博士論文

鳴禽の歌系列に履歴および時間帯が与える影響

上村卓也

目次

第 1 章	序論	3
図表		7
第 2 章	歌の自動認識	9
導入		9
方法		12
結果		22
考察		25
図表		32
第 3 章	歌文法の推定と時間帯による変化	44
導入		44
方法		47
結果		51
考察		57
図表		61
第 4 章	実験による検証	75
導入		75
方法		77
結果		83

考察		87
図表		90
第 5 章	総合討論	97
図表		104
引用文献		105
謝辞		113
付録		114

第 1 章　序論

多くの動物種が音声によるコミュニケーションを行う。その中でも連続した発声は系列発声と呼ばれる。系列発声には精密な運動制御・運動学習や聴覚知覚が必要であるため、系列発声について研究することで運動機能全般について示唆を得られることも多い。また、ヒトの発話も系列発声の一種であるため、様々な種の系列発声が広く研究されている。

私は博士過程において、運動制御・運動学習の神経機構について実際の行動に基づいたモデルを構築するために、系列発声の一種である鳴禽の歌を用いて研究を行った。鳴禽の歌は複雑で精密に制御された系列発声である(Brainard & Doupe, 2002; Scharff & Adam, 2013)。多くの種において歌は複数種類の要素が無音区間を挟んで並ぶように構成されている（図 1.1）。さらに歌を観察すると、要素の出現順序に規則が存在することがわかる（図 1.2）。歌における要素の出現規則を歌文法と呼ぶ。よって歌を正確にうたうためには、複雑な音響構造を持った要素の発声を細かく制御し、さらに要素の順序を歌文法に従って制御しなければならない。また、鳴禽は歌を学習によって獲得し、学習完了後も能動的に歌を維持している(Bolhuis, Okanoya, & Scharff, 2010)。要素の音響構造と歌文法の両方とも、学習によって獲得され、能動的に維持されるため、両方とも歌の重要な側面であると考えられている(Menyhart, Kolodny, Goldstein, DeVoogd, & Edelman, 2015)。

鳴禽の中でも、ジュウシマツの歌文法は特に複雑である。ジュウシマツの歌では、要素が歌文法に従って確率的に出現すると考えられている(Okanoya, 2004a)。歌文法は確率的ではあるが完全に無

作為ではない。さらに、要素の出現確率は系列のパターンによって変化する。また、歌文法が学習によって獲得されることや、歌の停止が歌文法の確率的な部分で起こりやすいことから(Seki, Suzuki, Takahasi, & Okanoya, 2008)、歌文法が何らかの形で生体内に保持されており、それが行動に影響を与えていることがわかる。したがって、複雑な歌文法が鳴禽の生体内でどのように表現され、どのように維持されているのかを調べることにより、より広範囲にわたる複雑な運動の制御・学習に関わる神経機構の理解に繋げることができる。よって私は、ジュウシマツの歌文法を対象として、その生成と維持の神経機構について研究を行った。

研究は、次の 3 段階に分けて行った。まず、大量の歌を効率良く解析するための手法を確立した。歌の維持について研究するためには、長期間にわたって記録した歌を解析しなければならない。そこで本研究では、機械学習を用いた歌の自動認識手法を確立した。さらに、認識精度を正しく評価するための指標を新たに考案した。結果として、研究のために実用性の高い自動認識器を作成し、その性能を正しく評価することができるようになった。この手法は本質的には動物種に依存しないので、系列発声全般の研究の発展に寄与すると期待できる。

次に、観測された要素系列を説明する歌文法のモデルを作成し、発声のパターンについて詳細に解析した。ジュウシマツの歌文法は複雑が故に、観測された要素系列から背後にある歌文法を推定することが簡単ではない。よって、多くの個体で観測された系列を正確にわかりやすく記述できるモデルが望まれる。これまでにもいくつかの研究で、歌文法のモデル化が試みられてきた(Katahira, Suzuki, Okanoya, & Okada, 2011; Kershenbaum et al., 2014; Markowitz, Ivie, Kligler, & Gardner, 2013;

Sasahara, Kakishita, Nishino, Takahasi, & Okanoya, 2006)。本研究では、これらの研究結果を考慮に入れ、観測された要素系列を最も良く予測するモデルを複数の候補から選択するという方法を用いた。選択したモデルを用いて歌を解析することで、体内の状態によって歌文法が変化することと、歌の生成に直前の系列の履歴を用いている可能性が示唆された。これまでに、脳内の神経回路を神経細胞単位でモデル化し神経活動をシミュレーションすることにより同様の示唆を得るという研究がいくつか行われてきた(Doya & Sejnowski, 1995; Hanuschkin, Diesmann, & Morrison, 2011)。神経細胞の活動を粒度の細かい現象、系列発声を粒度の粗い現象とすると、過去の研究は粒度の細かいモデルによるボトムアップな研究ということができる。ボトムアップな仮説では神経細胞や細胞内分子の性質について深く議論できる一方、モデルの状態と観測された行動との対応をつけることが難しい。それに対して、本研究は、観測された要素系列を元に構築した粒度の大きいモデルによるトップダウンな研究である。本研究では行動の規則に基づいた仮説を扱うので、発現した行動との対応がわかりやすい。さらに本研究では、作成したモデルの粒度を下げ、神経機構に迫るモデルを構築することにより、行動と対応する神経機構についての示唆を得ることができた。

　最後に、モデルから予測された神経活動が、実際の生体内でも観測されることを確認した。鳴禽の脳には、歌の生成・学習に重要な神経核が存在する(Brainard & Doupe, 2002)。これらの神経核は歌神経核と呼ばれている。歌神経核間の結合様式は、哺乳類の運動制御系と類似している。特に、ヒトの発話に関わる神経基盤と似ている部分が多い(Bolhuis et al., 2010)。本研究では特に歌の生成と維持に重要であると考えられている歌神経核として、哺乳類の運動前野と基底核に対応する歌神経核を対象とした。歌文法のモデルから、直前に生成した歌の履歴が重要であることが予測されたので、

自身の歌に対するこれらの神経核の反応を調べた。結果として、これらの神経核の活動が自身の歌の局所的な構造に依存して変化することがわかった。さらに、歌の履歴情報を表現する神経活動が存在することがわかり、モデルによる仮説を支持する結果となった。

博士論文は、次の 5 章から構成した。まず本章で導入を記述し、次章から 3 章かけて、前述の 3 段階の研究について記述した、最後の章に結論と考察を記述した。

図 1.1 | ジュウシマツの歌

スペクトログラムの下に、要素の分類と区間を示した。要素区間を黒色の棒で、分類をアルファベットで表した。図には歌の一部のみを示した。実際の歌は図に示したものより長いことが多い。

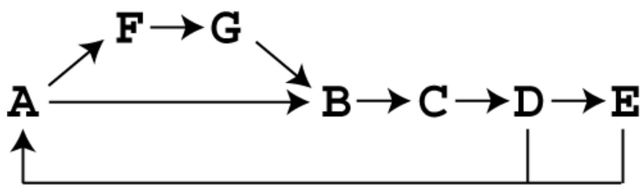


図 1.2 | 歌文法

図 1.1 の歌の、要素系列の規則。連続して出現する要素を、矢印で繋いだ。

第 2 章　歌の自動認識

導入

動物の発声に関する行動学的研究や生理学的研究では、数日間や数ヶ月間にわたる発声の解析が必要になることが珍しくない(Aronov & Fee, 2012; Tumer & Brainard, 2007)。例えば、ジュウシマツは 1 日に合計数十分の歌をうたい、その中に数万の要素が含まれる。複数個体による数日間の歌を扱う場合、数十万要素を含む数十時間の歌を解析しなければならない。よって、発声を効率良く解析するために、正確で頑健な自動認識器が強く望まれる。

鳴禽の歌のような動物の系列発声を対象とする研究においては、発声の音響構造だけでなく時間情報も重要であることが多い。この点において、生物学的研究のための系列発声の自動認識は、産業応用のためのヒトの音声自動認識と異なり、難しい。ヒトの音声自動認識では、音声を文字列に変換することが優先され、単語や音素の境界を検出することは重要ではないことが多い(Benzeghiba et al., 2007; Hinton, Li, et al., 2012)。一方、ヒト以外の動物の発声はヒトの発話と比べて要素の種類とその組み合わせのパターンが少ない。よって、パターンの複雑さの点では、ヒト以外の動物の発声の自動認識はヒトの音声自動認識よりも簡単である。したがって、ヒトの発話のための自動認識器を闇雲に利用するのではなく、生物学的研究のために特化した発声の自動認識手法を確立することが、今後の研究発展のために重要である。

鳴禽の歌を含む多くの動物の系列発声は、次の 3 つの特徴を持っている。1 点目は、歌が離散的な

要素からなることである(Rohrmeier, Zuidema, Wiggins, & Scharff, 2015)。類似する要素に 1 種類の分類を割り当てることで、歌を要素分類の記号列に変換することができる。同じ種類の要素は、運動神経による同じ命令群が筋肉の似た活動を引き起こすことで生成されると考えられている(Doya & Sejnowski, 1995; Fee, Kozhevnikov, & Hahnloser, 2004; Sober, Wohlgemuth, & Brainard, 2008; Yu & Margoliash, 1996)。また、同じ種類の要素からなる聴覚刺激は聴覚野において似た活動を引き起こす(Lewicki & Arthur, 1996)。2 点目は、歌の時間構造が精密に制御されている点である。特定の要素分類の特定の時刻に同期した神経活動が、歌の発声時と知覚時において見つかっている(Amador, Perl, Mindlin, & Margoliash, 2013; Chi & Margoliash, 2001; Hahnloser, Kozhevnikov, & Fee, 2002; Koumura, Seki, & Okanoya, 2014)。また、鳴禽の歌が階層的な時間構造から構成されていることを示す研究もある(Glaze & Troyer, 2006, 2007; Tachibana, Koumura, & Okanoya, 2015)。よって、鳴禽の歌を解析するときは、要素の開始時刻や終了時刻などの時間情報を正確に抽出することが重要である。3 点目は、要素系列が確率的な規則に従って出現することである。通常、要素系列の出現規則は個体によって異なり、学習により獲得される(Berwick, Okanoya, Beckers, & Bolhuis, 2011; Markowitz et al., 2013; Menyhart et al., 2015; Okanoya, 2004a)。この規則は歌文法と呼ばれている。

本研究では、これらの 3 つの特徴である、要素が分類できること・時間情報が重要であること・歌文法が存在すること、に着目し、多層ニューラルネットワークと隠れマルコフモデル（hidden Markov model、HMM）を合わせた認識器を用いて鳴禽の歌の自動認識を行った。多層ニューラルネットワークは深層ニューラルネットワーク（deep neural network、DNN）とも呼ばれている。DNN はデータを効果的に表現する特徴を学習により自動で構成し、頑健な認識を可能にする(Bengio, Courville, & Vincent, 2013)。つまり DNN は特徴量抽出器と分類器を兼ねている。本研究では、長時間の連続した音声を扱うため、畳み込み DNN（convolutional DNN、CDNN）を用いた(Fukushima, 1980; Lecun, Bottou, Bengio, & Haffner, 1998; Zeiler & Fergus, 2014)。CDNN と HMM の組み合わせは、ヒトの発話認識でも高い性能を達成している(Hinton, Li, et al., 2012)。HMM は確率的な文法規則によって生成された系列を扱うのが得意である。要素の境界は HMM または音圧と長さの閾値によって検出した。

鳴禽の歌の自動認識の研究は過去にもいくつかある。過去の研究では、動的時間伸縮法(Anderson, Dave, & Margoliash, 1996; Fantana & Kozhevnikov, 2014; Kogan & Margoliash, 1998)・HMM(Kogan & Margoliash, 1998)・サポートベクターマシン（support vector machine、SVM）(Tachibana, Oosugi, & Okanoya, 2014)が用いられている。機械学習では一般に、データを分類ごとに良く分離するような特徴量空間で表現することが重要である。過去の研究では、音声スペクトラム(Anderson et al., 1996; Fantana & Kozhevnikov, 2014; Kogan & Margoliash, 1998)・メル周波数ケプストラム係数（Mel-frequency cepstral coefficient, MFCC）(Kogan & Margoliash, 1998)・少数の音響特徴量(Tchernichovski, Nottebohm, Ho, Pesaran, & Mitra, 2000)・スペクトラムやケプストラムとその変化を含む大量の音響特徴量(Tachibana et al., 2014)、が歌を記述する特徴量として用いられてきた。しかし、MFCC は元々ヒトの発話認識のために作成されたもので、鳴禽の歌に用いるための十分な理由が無い。また、過去の研究で用いられた特徴量が他の種の発声にもうまく適用できるかどうかはわからない。SVM は大量の特徴量群から分類に有効なものを自動的に選択することができるが、はじめに用いる大量の特徴量群を手動で決定しなければならないという問題は解決できない。手動での特徴量抽出を介さずにデ

ータから自動で良い特徴量を抽出できる方法が理想的であり、DNNはこのような特徴を備えている。

方法

データの取得

オスの成鳥ジュウシマツ（孵化後 120 日齢以上）を 1 羽ずつ防音箱に入れた。2 日以上かけて環境に馴化させた後、14 時間の明期の間の音声を記録した。マイク（PRO 35、Audio-Technica Corporation、日本）、増幅器（MicTube Duo、Alesis、アメリカ合衆国）、録音装置（OCTA-CAPTURE、Roland、日本）を用い、32 kHz のサンプリング周波数・16 bit の深度で音声を記録した。明期と暗期は LED 照明によりそれぞれ 14 時間と 10 時間となるように制御した。餌と水は自由に摂ることができるようにした。

データ

13 羽の音声を録音した。記録した音声から、連続した発声を音声スペクトログラムの目視により手動で抽出した。音声スペクトログラムは長さ 512・移動幅 32 の短時間フーリエ変換を用いて作成したもののうち、1 kHz から 8 kHz の周波数帯を用いた。スペクトラムの計算には、パラメータ W = 4/512 の 0 次の離散扁長回転楕円体列（discrete prolate spheroidal sequences、DPSS）を窓関数として用いた(Percival & Walden, 1993)。DNN の効率よい学習のために、スペクトログラムから平均値を引き、標準偏差で割った。平均値と標準偏差は、訓練用データ（後述）から計算した。

要素の種類・区間は、スペクトログラムの目視により手動で決定した。手動による要素の決定の

客観性は、本研究の結果である汎化誤差の小ささによってある程度担保できる。発声の中に含まれていた地鳴きには、1 つの分類を割り当てた。どの分類にも属さないような要素や、複数の分類の中間に位置するような要素のように、手動で分類を決定できなかった要素は、分類不能とした。要素数が全体の 1%未満であった要素分類も、分類不能とした。

歌を、地鳴きまたは 300 ms 以上の無音区間によって区切られた、8 要素以上の系列と定義した。分類不能な要素が 1%以上あった個体（2 羽）は解析から除いた。そして、分類不能な要素をデータから除くために、歌を分類不能な要素の前後で分割した。区切られた歌の中で要素数が 3 未満であったものは解析から除いた。さらに、計算の効率化のためにデータ中の歌の長さをなるべく均等にするために、要素数が 15 以下になるように要素系列を分割した。

3 分割交差検証を行うため、要素系列を 3 群に分割した。機械学習では一般に、訓練用データが多いほど高い汎化性能が得られる。過去の鳴禽の歌の自動認識の研究でもそのような結果が得られている(Tachibana et al., 2014)。本研究では、非検証用データ（全体の 2/3）から無作為に選んだ 2 分と 8 分の訓練用データによる結果を比較した。要素系列の合計長は個体ごとに大きく異なっていたので（最小値=13.3 分、最大値=78.4 分）、非検証用データ全てを用いた訓練は行わなかった。ハイパーパラメータの調整が必要になった場合、訓練用データをさらに分割し、訓練用データ内における交差検証で汎化誤差が最小となった値を用いた。歌は個体によって大きく異なったので、各個体の歌を独立に評価した。

歌の自動認識のための 3 段階

問題の見通しを良くするために、歌の自動認識を 3 つの問題に分割した。それぞれの問題は導入で述べた歌の 3 つの性質と対応している。1 つ目の問題として、連続した音声から要素の区間を正確に検出しなければならない。この段階を「区間検出」と呼ぶ。2 つ目の問題として、それぞれの要素を与えられた種類（または背景ノイズ）に分類しなければならない。この段階を「局所分類」と呼ぶ。区間検出と局所分類の組み合わせは、二次元物体認識における物体位置決定または semantic segmentation に対応する。3 つ目の問題として、歌文法を考慮して局所分類の出力を配列しなければならない。この段階を「全体配列」と呼ぶ。全体配列の段階では、局所分類における誤りを文法の情報を用いて修正する。局所分類と全体配列は、それぞれボトムアップな処理とトップダウンな処理と考えることができる。

歌認識全体としては、これら 3 段階の複数通りの組み合わせが可能である。本研究では、次の 3 通りの組み合わせで認識を行い、結果を比較した（図 2.1）。1 つ目の構成では、区間検出、局所分類、全体配列を順に行った（図 2.1a）。2 つ目の構成では、局所分類の次に区間検出と全体配列を行った（図 2.1b）。3 つ目の構成では、局所分類と全体配列の後、区間検出と（2 回目の）全体配列を行った（図 2.1c）。3 つ目の構成における 2 回の全体配列には、それぞれ異なる手法を用いた。今後は、これら 3 構成を「区間検出→局所分類→全体配列」、「局所分類→区間検出＆全体配列」、「局所分類＆全体配列→区間検出＆全体配列」と呼ぶ。

音圧と長さの閾値による要素区間の検出

区間検出→局所分類→全体配列の構成においては、要素の開始時刻と終了時刻は音圧と長さの閾値によって決定した（図 2.2、付録 A)。まず、音圧がある閾値より大きい区間を、音区間として抽出した（図 2.2、オレンジの棒)。音圧は対数振幅スペクトラムの 1 kHz から 8 kHz の周波数帯の合計を、スペクトログラムの移動幅 1 ms ごとに計算した。次に、ある閾値より短い無音区間の両側にある音区間を結合した。最後に、ある閾値以上の長さの音区間を要素とした（図 2.2、青い棒)。この 3 つの閾値は、訓練用データ内で要素の分類を無視したマッチング誤差（後述）が最小となるように決定した。結果として、無音区間長の最適な閾値は 0 であった。つまり、無音区間を挟む 2 つの音区間が結合されることはなかった。

CDNN による局所分類

全 3 構成において、固定長の時間窓内のスペクトログラムを、CDNN によって与えられた要素分類のうちのひとつに分類した（図 2.3)。区間検出→局所分類→全体配列と局所分類→区間検出&全体配列の構成では、次のような構造の CDNN を用いた。入力側から出力側へ、入力層、畳み込み層・cascaded cross channel parametric (CCCP) pooling 層・max-pooling 層を 3 層ずつ、全結合層、ソフトマックス層とした。CDNN では各層の値を 3 階テンソルで表し、ある層の値が次の層の入力となる。また、パラメータは重みWとバイアスbであり、Wは行列、bはベクトルで表される。ある層への入力値をX、その層の値をYとすると、それぞれの層の値は次のように決まる。

入力層：

$$Y = X \tag{1}$$

畳み込み層：

$$Y_{cij} = f\left(b_c + \sum_{c',0\le i'<h_f,0\le j'<w_f} W_{i'j'}X_{c',i+i',j+j'}\right) \tag{2}$$

max-pooling 層：

$$Y_{cij} = \max_{h_s i\le i'<h_s i+h_f, w_s j\le j'<w_s j+w_f} X_{ci'j'} \tag{3}$$

ただし 3 階テンソルXの各要素をX_{cij}で、行列Wの各要素をW_{ij}で、ベクトルbの各要素をb_cで表した。CDNN では 3 階テンソルのcはチャネル、iは行、jは列に対応する。畳み込み層ではfで活性化関数を、h_fとw_fでフィルターの高さと幅を表した。max-pooling 層ではh_fとw_fでフィルターの高さと幅を、h_sとw_sで移動の高さと幅を表した。CCCP pooling 層は$h_f = w_f = 1$のフィルターを持つ畳み込み層で実装され、全結合層は畳み込み層により実装される。CCCP pooling 層は畳み込み層の活性化関数として働く小さなネットワークと考えることができる(Lin, Chen, & Yan, 2014)。ソフトマックス層は、活性化関数がソフトマックス関数である畳み込み層により実装される。ソフトマックス関数は次の式で表される。

$$f_{softmax}(X)_i = \frac{\exp(X_i)}{\sum_{i'}\exp(X_{i'})} \tag{4}$$

畳み込み層のフィルターの大きさは初めの 2 層で$h_f = w_f = 5$、最後の層で$h_f = w_f = 4$とした。畳み込み層と CCCP pooling 層のチャネル数は 16 とした。畳み込み層・CCCP pooling 層・全結合層の活性化関数は全て rectified 線形素子とした。Rectified 線形素子は入力が 0 未満のとき 0 を、0 位上のとき入力値を出力する関数で、次の式で表される。

$$f(x) = \begin{cases} 0 & (x < 0) \\ x & (x \ge 0) \end{cases} \tag{5}$$

Max-pooling 層のフィルターの大きさと移動幅は共に$h_f = w_f = 2$、$h_s = w_s = 2$とした。入力層の高さは、1 kHz から 8 kHz の周波数帯と対応させて 112 とした。全結合層の次元は 240 とした。全結合層のフィルター幅は入力層の窓幅によって決定される。区間検出→局所分類→全体配列と局所分類→区間検出＆全体配列の構成では、入力層の窓幅は 96（111 ms）としたので、全結合層のフィルター幅は$w_f = 9$となった。入力層の窓幅は、およそ 1 要素が収まるように決定した。全結合層のフィルター高は入力の高さ 112 を全て収めるため、$h_f = 11$とした。局所分類＆全体配列→区間検出＆全体配列の構成では、複数の要素にまたがる歌文法を捉えるため、ソフトマックス層の直前にもう 1 層の全結合層を追加した（図 2.3b）。入力層の窓幅を 288（303 ms）とするため、追加した全結合層のフィルター幅は$w_f = 25$にした。これは連続するおよそ 3 要素を収める幅である。全結合層を追加した狙いは、下層の全結合層で表現されているであろう個々の要素の局所的な情報を、3 要素にまたがる大局的な歌文法の情報として統合するためである。下層の全結合層で個々の要素の情報を狙い通り表現するために、まず上層の全結合層を含まない CDNN を訓練し、その後、上層の全結合層を追加して CDNN 全体を訓練した。

パラメータ（結合の重みWとバイアスb）は交差エントロピー損失関数を用いた確率勾配法によって更新した。交差エントロピー損失関数による誤差Cは次の式で表される。

$$C = -\ln Y_k \tag{6}$$

ただしkはある時刻における正解の要素分類、Y_kはソフトマックス層のk次元目の値を表す。確率勾配法とは、各パラメータを、誤差の偏微分の定数倍変化させるという操作を繰り返すことにより、パラメータの訓練を行う手法である。パラメータの更新は、次の式に従って行われる。

$$W \leftarrow W - \alpha \frac{\partial C}{\partial W} \tag{7}$$

$$b \leftarrow b - \alpha \frac{\partial C}{\partial b} \tag{8}$$

ただしWはある重みを、bはあるバイアスを、αは学習率を表す。学習率αは訓練用データごとに、次のように決定した。まず、訓練用データ内の 2/3 を小訓練用データとし、残りの 1/3 を小検証用データとして、0.001 から 0.04 の範囲で学習率の初期値を探索した。学習率の初期値は、32 回のパラメータ更新の間に小検証用データにおいて最良の結果を得た値とした。次に、小検証用データにおける誤差が減少しなくなるまで、初期値の学習率を用いて訓練を行った。誤差が減少しなくなったら、学習率を半分にし、さらに訓練を続ける、という操作を 3 回繰り返した。小訓練用データと小検証用データを 3 通りに分割し、3 通りのパラメータを得た。認識の段階では、3 通りのパラメータによる CDNN の出力値を平均した。訓練は、小訓練用データからランダムに選んだ 32 s 未満のデータを順に用いて行った。CDNN の訓練の手順を付録 B に記述した。

DNN の重みWとバイアスbは He, Zhang, Ren, & Sun (2015)に従って初期化した。重みの初期値は、平均値が 0、標準偏差が

$$\sqrt{\frac{\text{その変数へ入力する結合数}}{2}}$$

であるガウス分布から抽出した。バイアスの初期値は 0 とした。学習率の初期値を決定する段階と同時に、パラメータの初期値を複数通りのサンプルから最適なものに決定した。

区間検出→局所分類→全体配列の構成では、ソフトマックス層の次元は要素の種類数とした。他

の 2 構成では、後に HMM による区間検出を行うため、ソフトマックス層の次元は 3×要素の種類数＋背景雑音とした。

HMM による区間検出と全体配列

区間検出→局所分類→全体配列の構成では、局所分類の出力と歌文法を HMM によって組み合わせた（図 2.1a）。一般に、HMM と DNN を組み合わせる場合、DNN の出力は隠れ状態の事後確率として扱われる(Bourlard & Morgan, 2012; Hinton et al., 2012)。本研究では、歌文法を 2 次マルコフ過程で記述した（図 2.4a）(Yamashita, Okumura, Okanoya, & Tani, 2011)。2 次マルコフ過程では、次に出現する要素分類の確率が、直前の 2 要素によって決まる。そのために、要素分類の生成を介して隠れ状態が遷移し、DNN の出力値が要素分類の事後確率として扱われるように、HMM の構造を改変した。

$$\text{次の状態} \sim P\left(\text{次の要素分類} \middle| \text{現在の隠れ状態}\right)$$

$$\text{DNN の出力} = P\,(\text{次の要素分類}|\text{スペクトログラム})$$

隠れ状態間の遷移確率は、訓練用データを用いて計算したものに定数を加えて合計が 1 になるように平滑化したものを用いた。平滑化の定数は訓練用データ内の交差検証によって決定した。初期状態とその次の状態への遷移確率は一様分布に従うとした。閾値によって決定した音区間ごとに DNN の出力を平均し、要素分類の事後確率とした。与えられたスペクトログラムの生成確率を最大化する隠れ状態系列を Viterbi アルゴリズムによって決定し、要素分類の系列に変換した（付録 C）。スペクトログラムの生成確率を最大化する隠れ状態系列は、次の式で表される。

求める隠れ状態系列

$$= \underset{隠れ状態_{1\text{ から }n}}{\operatorname{argmax}} P\left(隠れ状態_0\right)\prod_{i\in[1,n]} P\left(スペクトログラム_i \middle| 要素分類_i\right) P\left(要素分類_i \middle| 隠れ状態_{i-1}\right)$$

ただし、iは要素の位置を、nは要素の個数を表す。

局所分類→区間検出＆全体配列と局所分類＆全体配列→区間検出＆全体配列の構成では、全体配列と同時に区間検出を行った（図 2.1b, c）。要素の区間を正確に検出するために、各要素を 3 区間に分割し（図 2.5）、各隠れ状態を 4 状態に分割した。分割した 4 状態のうち初めの 3 状態から要素の 3 区間が生成され、最後の 1 状態から要素間の無音区間が生成されるとした（図 2.4b）。分割した 4 状態間の遷移は、自身への遷移と左から右への遷移のみとした。最後の 2 状態からは次の要素分類への遷移を可能にし、遷移確率として訓練用データを用いて計算した平滑化遷移確率を用いた。分割した状態間の遷移確率は一様分布に従うとした。

一般に、HMM と DNN を組み合わせた音声認識では、DNN の出力は音声データが与えられたときの事後確率 P (要素分類|スペクトログラム)として扱われる。この場合、事後確率はベイズの定理を用いて、要素分類が与えられたときの音声データの生成確率 P (スペクトログラム|要素分類)に変換されることが多い(Bourlard & Morgan, 2012; Hinton et al., 2012)。本研究ではこの変換を行うかどうかを、訓練用データ内の交差検証によって決定した。

評価

認識器の性能は交差検証によって、次の 2 種類の指標を用いて評価した。まず要素分類の精度を

評価するために、出力された分類系列と正解の分類系列の Levenshtein 距離を計算した。Levenshtein 距離はある記号列を別の記号列へ変換するのに必要な最小の挿入・削除・置換の数である。評価には、Levenshtein 距離を正解系列の長さで割ったものを用いた。本論文ではこの指標を Levenshtein 誤差と呼ぶ。Levenshtein 誤差はヒトの発話自動認識における単語誤り率と同様の指標である。

Levenshtein 誤差は 2 つの記号列の差を測るためのものであり、要素の区間に関する誤差を捉えることはできない。要素の分類を無視して区間の情報のみが必要であることは少ないため、本研究では要素分類と区間検出の精度を合わせて評価する指標を考案した。分類と区間の誤差を合わせて評価するための単純な指標として、正解と異なる分類を振られた区間の合計長を考えることができる。しかし、この指標では、例えば同じ分類に属する連続 2 要素と、長い 1 要素を区別することができない。別の指標として、出力された系列中の各要素にとって、正解系列中の同じ分類に属する最も近い要素との開始時刻間の距離と終了時刻間の距離の合計を考えることができる。この指標は、片方の系列に含まれる要素分類が全てもう片方にも含まれていないと正しく機能しない。本研究では、要素の分類と区間を制約なく評価できる新しい指標を考案した（図 2.6)。まず、それぞれの正解の要素区間について、出力系列中の同じ分類を振られた要素で、区間の重複が最も長いものを対応付けた。この段階で対応付けられる相手の要素が存在しないこともある。出力系列中の要素区間内で、正解系列中の対応付けられた要素区間と重複した区間を、正しく認識された要素区間とした。次に、正解系列中の無音区間のうち、無音区間であると認識された区間を、正しく認識された無音区間とした。最後に、正しく認識された要素区間と正しく認識された無音区間の合計長を、正解系列の長さで割って 1 から引いた。この指標をマッチング誤差と呼ぶ。

統計検定

異なる条件における汎化誤差は、Wilcoxon 符号順位検定を用いて個体内で比較した。必要に応じて Bonferroni 補正を行った。本論文では、可読性を上げるために補正後の *p* 値を記述したので、*p* 値が 1 を超える場合があるが、その場合は 1 とした。

計算

計算と音声の記録は全て java で書いたプログラムによって行った。計算に用いたプログラムのソースコードを、付録 D に示した。CDNN の計算はグラフィックプロセッサ（GTX 970 または 980、NVIDIA、アメリカ合衆国）上で行った。CDNN の畳み込み層の計算には cuDNN ライブラリを用いた。

結果

データ

13 羽のジュウシマツの歌を記録した。歌に含まれる全ての要素を手動で検出し、分類した。手動で分類不可能な要素が 1%より多く含まれていた個体（2 羽）を除き、残った 11 羽の歌を認識器の評価に用いた。歌の合計長の平均値±標準偏差は、44.1±20.6 分であった。要素の総数は 19548.8±10037.6、要素の種類数は 8.5±3.9 であった。個体によって歌が大きく異なったので、認識器の評価は個体ごとに行った。

区間検出→局所分類→全体配列による歌認識

区間検出→局所分類→全体配列の構成では、音圧と長さの閾値によって要素の区間を決定した（図 2.2）。閾値は訓練用データ内で要素の分類を無視したマッチング誤差が最小になるように決定した。そして、CDNN によってスペクトログラムを 111 ms の時間窓ごとに分類した（図 2.3）。時間窓の幅は、およそ 1 要素の長さと対応するようにした。最後に、局所分類の出力値を検出された音区間ごとに平均し、HMM によって歌文法と組み合わせた。歌文法は訓練データから 2 次マルコフモデルによって推定した（図 2.4a）。

Levenshtein 誤差の平均値は 2 分の訓練用データで 1.86%、8 分の訓練用データで 1.57%であった（表 2.1）。一方、マッチング誤差の平均値は 2 分と 8 分の訓練用データでそれぞれ 4.57%と 4.42%もあり、個体によっては約 10%になるものもあった（図 2.7a）。区間検出→局所分類→全体配列の構成は要素区間の正確な検出が得意でないのかもしれない。実際に、マッチング誤差が最大であった個体の認識結果では、短い無音区間を検出することができていなかった（図 2.8b）。マッチング誤差の下限は、要素の分類を無視したマッチング誤差によって測った区間検出の精度となる（図 2.7c）。区間検出がうまくいった個体では、その後の要素の分類と配列はうまくいった（図 2.8a）。訓練用データが多いほうが Levenshtein 誤差もマッチング誤差も小さくなった（Wilcoxon 符号順位検定により、Levenshtein 誤差で $W = 61$、$p = .0098$、マッチング誤差で $W = 66$、$p < .001$）。

局所分類→区間検出＆全体配列による歌認識

局所分類→区間検出＆全体配列の構成では、まず区間検出→局所分類→全体配列の構成と同じ

CDNN を用いて局所分類を行った（図 2.3a）。次に、HMM を用いて区間検出と全体配列を行った。要素の区間を正確に検出するために、それぞれの要素を長さの等しい 3 小区間に分割した（図 2.5）。また、HMM 内の隠れ状態を 4 つに分割し、初めの 3 状態が要素の 3 小区間を生成し、最後の状態が無音区間を生成するとした（図 2.4b）。最後の 2 状態には、次の要素の先頭の隠れ状態への結合を付与した。この結合は、ある要素から次の要素への遷移に対応する。各要素を 3 小区間以上に分割することは重要である。要素を分割しない場合、HMM は短い無音区間を挟む 2 要素と長い 1 要素を区別できない（図 2.5b の要素 A と C）。要素を 2 小区間に分割する場合、隣り合う 2 小区間の分類に誤りがあると、そこに要素の区切りがあると誤って認識してしまう（図 2.5c の要素 B）。

区間検出→局所分類→全体配列の構成ではうまくいかなかった歌でも、分割された要素の小区間が CDNN によって正しく認識され、その結果 HMM によって全体配列と区間検出が正しく行われた（図 2.9）。Levenshtein 誤差の平均値は訓練用データ 2 分で 1.51%、8 分で 1.09%であり（表 2.1）、区間検出→局所分類→全体配列の構成とは有意な差はなかった（p 値を 3 倍に補正した Wilcoxon 符号順位検定により、訓練用データ 2 分で $W = 35$、$p > 1$、8 分で $W = 36$、$p > 1$）。マッチング誤差の平均値は訓練用データ 2 分で 2.25%、8 分で 2.06%であり（表 2.1）、区間検出→局所分類→全体配列の構成よりも小さかった（訓練用データ 2 分・8 分で共に $W = 66$、$p = .0029$）。よって、局所分類→区間検出＆全体配列の構成は要素の区間を正確に認識することに適している。訓練用データが多いほうが Levenshtein 誤差もマッチング誤差も小さくなった（Wilcoxon 符号順位検定により、Levenshtein 誤差で $W = 63$、$p = .0049$、マッチング誤差で $W = 66$、$p < .001$）。

局所分類＆全体配列→区間検出＆全体配列による歌認識

局所分類＆全体配列→区間検出＆全体配列の構成では、CDNN によって局所分類と同時に全体配列も行った。複数の要素にまたがる情報を捉えるため、CDNN の入力時間窓を、およそ 3 要素の長さである 303 ms とした（図 2.3b）。区間検出と 2 回目の全体配列は、局所分類→区間検出＆全体配列の構成と同様に HMM によって行った。

本構成によっても、局所分類→区間検出＆全体配列の構成と同様に要素を正確に認識できた（図 2.10）。Levenshtein 誤差は局所分類→区間検出＆全体配列の構成よりも小さく（p 値を 3 倍に補正した Wilcoxon 符号順位検定により、訓練用データ 2 分で $W = 63$、$p = .015$、8 分で $W = 65$、$p = .0059$）、マッチング誤差には有意な差は無かった（訓練用データ 2 分で $W = 34$、$p > 1$、8 分で $W = 36$、$p > 1$）。区間検出→局所分類→全体配列の構成と比較すると、Levenshtein 誤差とマッチング誤差は局所分類＆全体配列→区間検出＆全体配列の構成のほうが小さかった（両誤差・両訓練用データ長で $W = 66$、$p = .0029$）。したがって、入力時間窓の長い CDNN が複数要素にわたる情報を捉えることができた可能性が高い。訓練用データが多いほうが Levenshtein 誤差もマッチング誤差も小さくなった（Wilcoxon 符号順位検定により、Levenshtein 誤差で $W = 66$、$p < .001$、マッチング誤差で $W = 65$、$p = .0020$）。

考察

本研究では、鳴禽の歌の自動認識の手法として 3 つの構成を比較し、そのうちの 2 構成で約 2%という実用的に十分低い汎化誤差を得た（表 2.1）。全ての構成で、局所分類は CDNN によって行い、全体配列は HMM を用いて行った。本研究によって、CDNN と HMM の組み合わせが、ヒトの発話

認識だけでなく鳴禽の歌の自動認識にも有効であることが示された。

区間検出→局所分類→全体配列の構成によるマッチング誤差は、他の 2 構成よりも大きかった。この結果から、区間検出は音圧と長さの閾値による手法よりも HMM を用いて行ったほうが良いことがわかる。Levenshtein 誤差は局所分類＆全体配列→区間検出＆全体配列の手法よりも大きかったが、実用に耐えうる小ささであった（表 2.1）。区間検出→局所分類→全体配列の構成の利点として、HMM によって区間検出を行わないことにより、CDNN で扱う分類数が他の 2 構成と比較して約 1/3 になり、局所分類の計算が速くなるという点がある。計算が速いということは、パラメータをより細かく調整できるということであり、より良い汎化性能につながる可能性がある。局所分類→区間検出＆全体配列の構成と局所分類＆全体配列→区間検出＆全体配列の構成を比較すると、Levenshtein 誤差が後者のほうが小さかった。したがって、目的によって次のように手法を使い分けると良いことがわかる。要素の正確な分類が最優先の目的で、要素の区間はあまり重要でない場合は、区間検出→局所分類→全体配列の構成を用いるのが良い。要素の分類と区間検出の両方を正確に行う必要があるときは、局所分類＆全体配列→区間検出＆全体配列の構成を用いるのが良い。

本研究で検討した手法は、要素の正確な分類と区間検出が必要で、要素系列が確率的である全ての発声に適用することができる。DNN は手動による特徴量抽出無しでデータから良い特徴量を学習することができる(Bengio, Courville, et al., 2013; Zeiler & Fergus, 2014)。また、HMM は認識の際に要素系列の確率的な規則を考慮するのに適している。これらの手法は本質的に音声データの特別な特徴量に依存しない。

本研究のもう 1 つの成果は、マッチング誤差の考案である。マッチング誤差を用いることで、要素の分類と区間の認識精度を合わせて評価することができた。また、全ての要素に同じ分類を割り当てることにより、マッチング誤差を用いて区間認識のみの精度を測ることもできる。

認識精度を向上させる可能性のある他の手法

DNN において汎化誤差を減少させる可能性のある手法がいくつか提案されている。その 1 つに drop-out 法がある(Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012)。drop-out 法では訓練の反復ごとに一定割合の変数を無効にする。この手続きは、複数の DNN を同時に訓練し、認識の段階ではその平均値を出力として用いることに対応している。本研究では異なるデータによって訓練した DNN の出力を認識の段階で平均した。画像認識についての過去の研究では、異なる構造を持った複数の DNN の出力を平均すると汎化性能が向上した(Ciresan, Meier, & Schmidhuber, 2012)。しかし、この手法では訓練する DNN の数に比例して計算時間が長くなる。DNN のパラメータ更新については、resilient 伝播法を用いるとより頑健なパラメータ学習ができるかもしれない(Igel & Hüsken, 2000; Riedmiller & Braun, 1993)。この手法ではパラメータの更新量を適応的に変化させる。さらに、訓練用データを改変して擬似的に数を増やすという手法もある(Ciresan et al., 2012; Lecun et al., 1998)。画像認識では、ガウシアンノイズを加えたり、拡大縮小したり、傾けたり、局所的に歪めたりするという改変が汎化性能の向上に有効である。音声認識において有効な改変手法を検討することは、今後の有用な研究となり得る。例えば、入力を時間軸方向に局所的に拡大縮小するのか、周波数軸方向に拡大縮小するのか、その両方か、どれが効果的なのかはまだわかっていない。

本研究では、ジュウシマツの歌文法を 2 次マルコフモデルで表現した(Yamashita et al., 2011)。しかし、ジュウシマツの歌文法を記述する他のモデルとして、HMM(Katahira et al., 2011)、単純化 renewal プロセス(Kershenbaum et al., 2014)、k-reversible 言語(Sasahara et al., 2006)が提案されている。またカナリアの歌文法を確率接尾辞木で記述した研究もある(Markowitz et al., 2013)。本研究の第 3 章では、可変次数マルコフモデルにより歌文法を記述した。これらのモデルを用いることで汎化性能が向上する可能性がある。K-reversible 言語・確率接尾辞木・可変次数マルコフモデルは本研究の枠組みに簡単に取り入れることができる。単純化 renewal プロセスも、1 要素分類の繰り返し数が有限であれば取り入れることができる。HMM による全体配列における文法の表現に HMM を用いるためには、階層型 HMM を用いることができるかもしれない。

判別前向き学習(Lecun et al., 1998)や相互情報量最大化(Bahl, Brown, De Souza, & Mercer, 1986)の手法によって、DNN と HMM を合わせて訓練すると、認識精度を上昇させることができるかもしれない。しかし、本研究のように要素区間の正解が与えられている問題でこれらの手法がどの程度有効かはわからない。

本研究では音声スペクトログラムの計算のために 0 次 DPSS を窓関数として用いた。複数の窓関数によるスペクトラムの平均値を用いると、背景雑音に頑健なスペクトログラムの計算ができるかもしれない(Percival & Walden, 1993; Tchernichovski et al., 2000)。

調整していないハイパーパラメータ

訓練過程には大量のハイパーパラメータが存在する。全てのハイパーパラメータを細かく調整することは実効的に不可能なので、本研究では予めいくつかのハイパーパラメータの値を固定した。よって、これらのハイパーパラメータの値を細かく調整すると、認識精度が向上するかもしれない。DNN の訓練では、訓練用データの一部について認識精度が改善しなくなると、確率勾配法の学習率を半分にして訓練を再開した。半分という割合は恣意的に決めたものであるため、より良い訓練スケジュールがあるかもしれない。また、層の数、畳み込み層のフィルターの大きさ、全結合層の次元など、CDNN の構造もあらかじめ固定した。

その他の調整していないハイパーパラメータは、DNN の訓練用データの長さ、フーリエ変換の長さ、スペクトログラムの移動幅、DPSS 窓関数のパラメータである。これらのハイパーパラメータは訓練用データ内の交差検証による汎化誤差を最小にするように決定することができる。固定したハイパーパラメータの値は方法の節に記述した。

手法の限界

本研究で用いた手法の欠点として明らかなのは、訓練のために要素の分類と区間を事前に決定しなければならない点である。訓練用データに必要な量は 2 分程度と短いが、何十もの個体を解析しなければならないときは手間になるかもしれない。また、これは同時に、訓練用データに含まれない要素分類を認識できないことも意味する。つまり、この手法は、ジュウシマツのような学習完了後に新たな要素分類を獲得しない種の発声にしか適用することができない。

本研究では、記録した音声データから歌の区間を手動で抽出した。これは、音声データのほとんどは本研究の興味の対象外である背景雑音で、長時間の不要な音声を含むデータの解析には計算時間が長くかかるからである。本質的には、歌の事前抽出は必要ないと予想できるが、これは厳密に評価しなければならない。歌の区間を検出する手法として他に考えられるのは、本研究で用いた手法を応用して、歌の区間全体に 1 つの分類を割り当て、歌と歌の間の無音区間に別の分類を割り当てるという手法である。歌区間を検出する段階では正確な時間解像度は必要ないので、時間解像度を落として計算を速くすることができる。

応用の可能性

本研究では、歌の自動認識を個体ごとに行った。訓練用データに複数個体の歌を含めることで、新たな個体の歌を認識できる認識器を構築できるかもしれない。そのためには、様々な個体から記録した多様性のある訓練用データが必要である。また、個体間で一貫した方式で正解の要素分類を決定する必要がある。

本研究では CDNN を教師あり認識器として用いた。つまり訓練の際に入力の音声データと正解の要素分類・区間を用いて、音声データと正解の要素分類・区間を対応づけるパラメータを学習した。正解の要素分類・区間を用いずに、CDNN を教師なし認識器として用いることもできる(Bengio, Thibodeau-Laufer, Alain, & Yosinski, 2013)。CDNN を教師なし認識器として用いると、音声データを効率良く表現するための特徴を自動で抽出することができる。この方法により、個体ごとの歌の音響的特徴を抽出し、比較することができるかもしれない。さらに、この方法を複数個体の歌に適用す

ることで、ジュウシマツという種の歌に特有の音響構造や歌文法を抽出できるかもしれない。

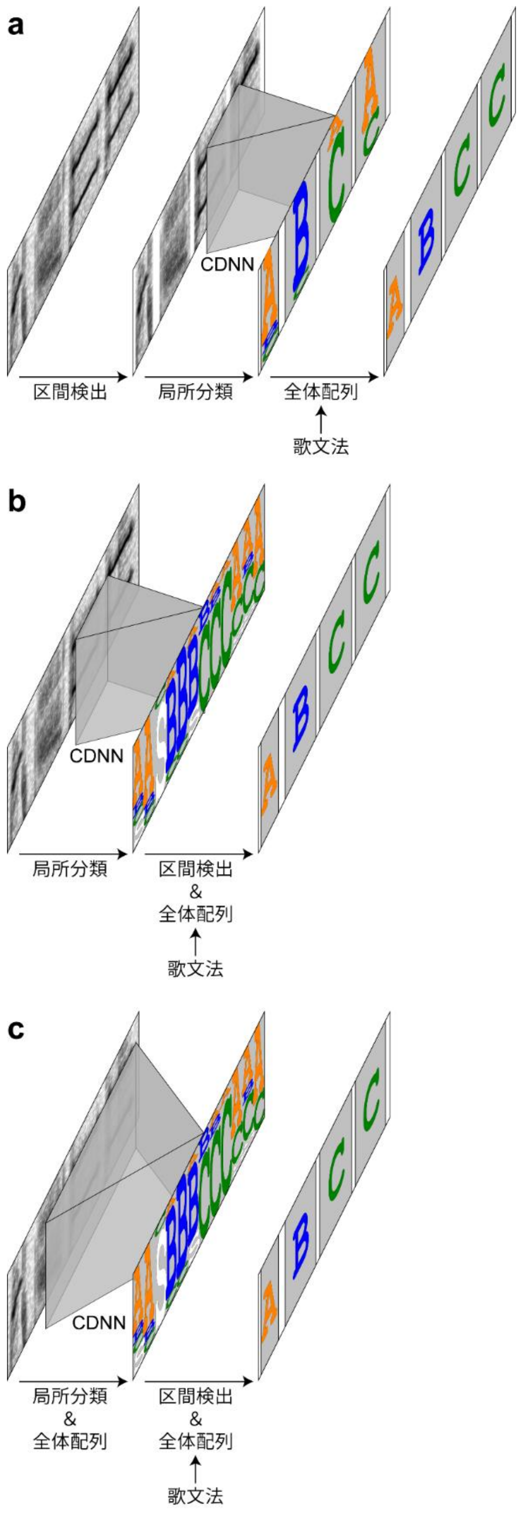


図 2.1 | 歌認識のための 3 構成

(a)区間検出→局所分類→全体配列の構成。スペクトログラム・音区間・DNN の出力値・認識結果を斜めに示した。要素区間を灰色の長方形で、要素分類をアルファベットで表した。要素間の無音区

間を白色の長方形で表した。(b)局所分類→区間検出＆全体配列の構成。(c)局所分類＆全体配列→区間検出＆全体配列の構成。

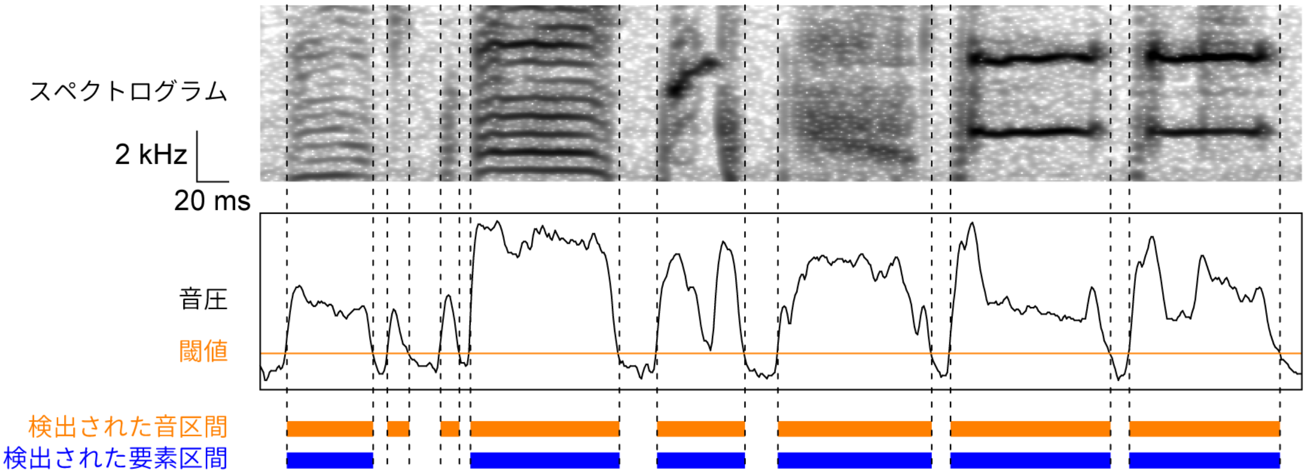


図 2.2 | 閾値による区間検出

上から、スペクトログラム、音圧、検出された音区間、検出された要素区間。音圧の閾値も音圧の図に示した。

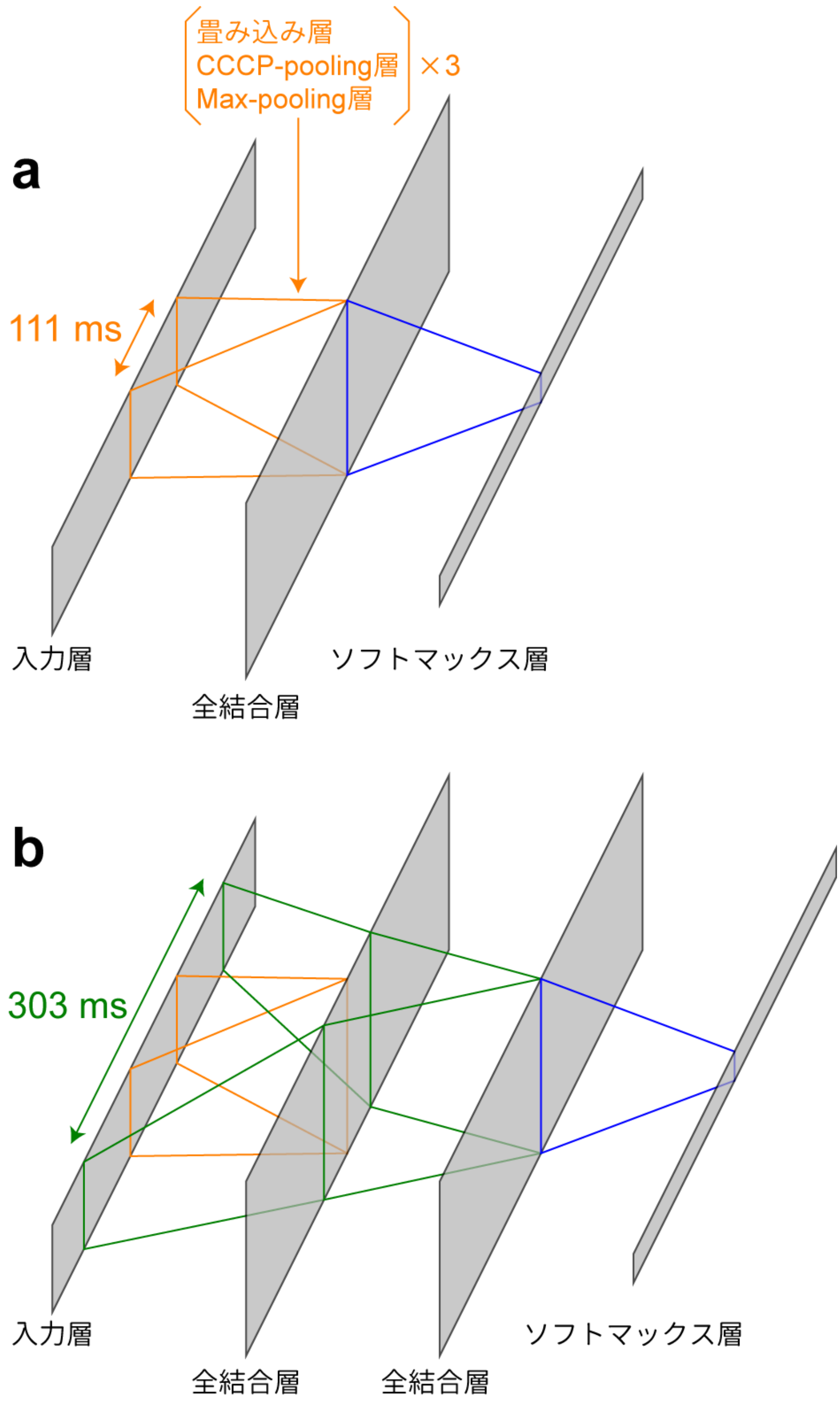


図 2.3 | CDNN

(a)区間検出→局所分類→全体配列と局所分類→区間検出＆全体配列の構成で用いた CDNN の構造。

(b)局所分類＆全体配列→区間検出＆全体配列の構成で用いた CDNN の構造。(a)の構造に全結合層を 1 層追加したものである。

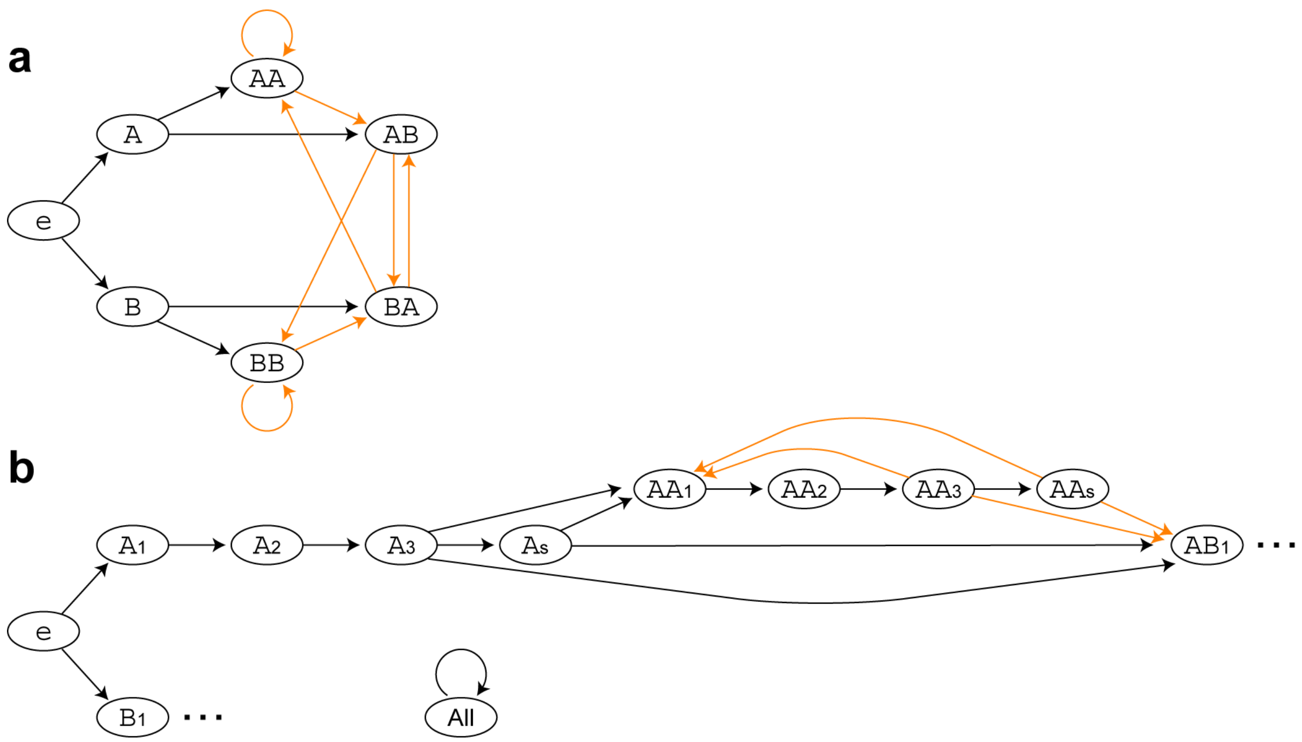


図 2.4｜HMM で用いた歌文法

(a)区間検出→局所分類→全体配列の構成で用いた HMM の歌文法。隠れ状態を楕円で表した。簡単のため、要素分類が A と B のみの例を示した。歌の開始を状態 e で表した。オレンジ色の矢印で示した結合の遷移確率には、訓練用データを用いて計算した確率を用いた。黒色の矢印で示した結合の遷移確率は、一様分布に従うとした。(b)局所分類→区間検出＆全体配列と局所分類＆全体配列→区間検出＆全体配列の構成で用いた HMM の歌文法。(a)の隠れ状態を 4 分割したものを、下付きの 1、2、3、s で表した。全ての状態に自身への遷移を持たせた。グラフ全体が複雑であるため、一部のみを示した。

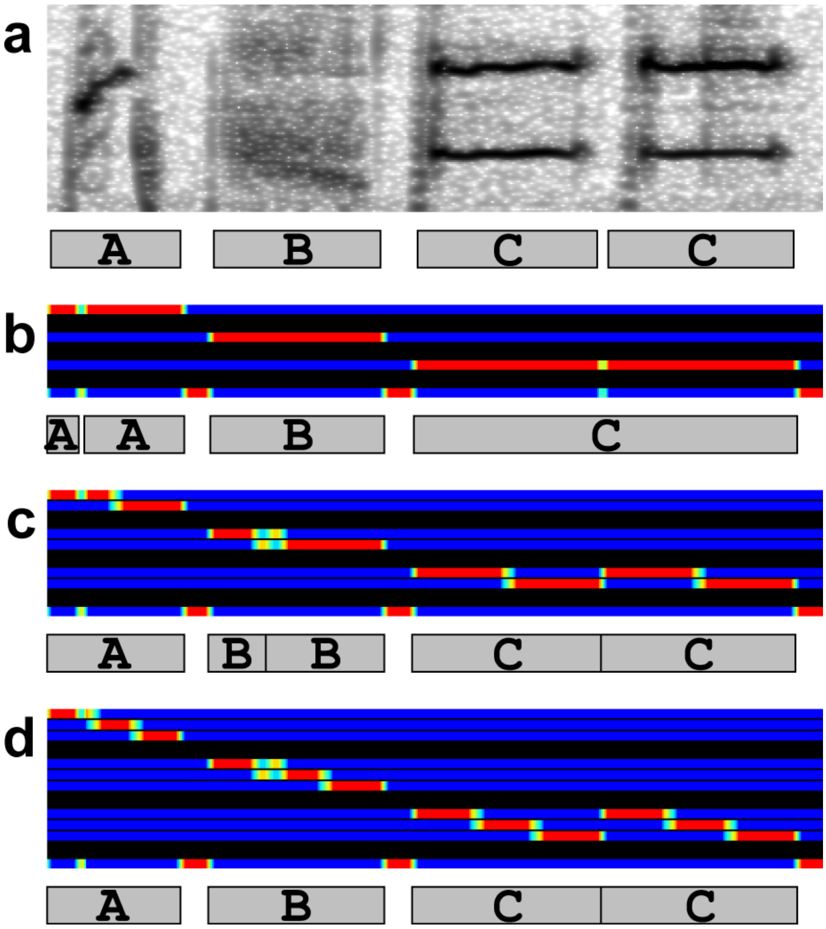


図 2.5｜要素の分割

(a)上段：スペクトログラム。下段：正解の要素分類と区間。要素区間を灰色の長方形で、分類をアルファベットで表した。(b)要素を分割しないときの認識結果の例。上段：CDNN の出力。下段：出力された要素分類と区間。CDNN の出力では、上から要素分類ごとに出力値を示した。(c)要素を 2 分割したときの認識結果の例。CDNN の出力では、上から要素分類ごとに、要素内の小区間ごとに、出力値を示した。(d)要素を 3 分割したときの認識結果の例。

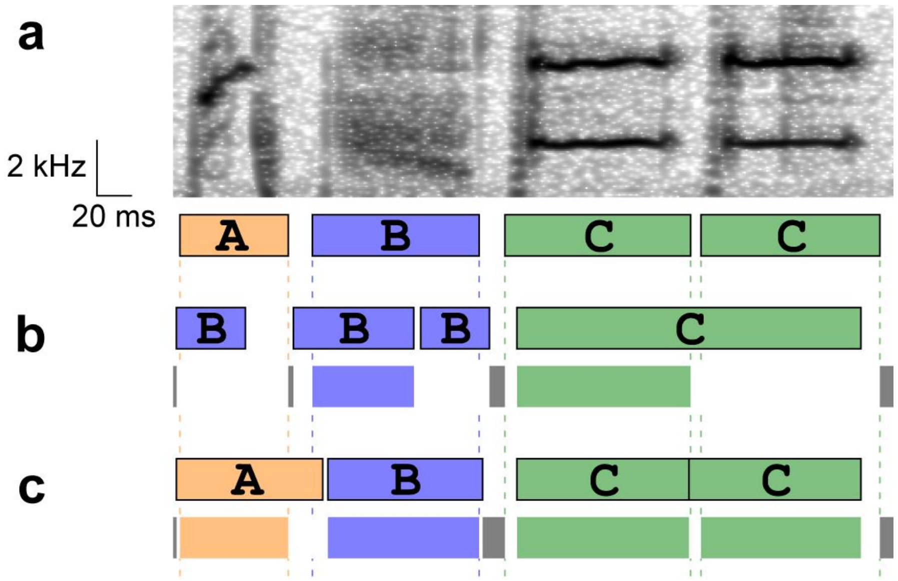


図 2.6｜マッチング誤差

(a)上段：入力スペクトログラム。下段：正解の要素区間。要素の分類をアルファベットで表した。

(b)上段：認識結果の例。下段：正しく認識された要素区間（色付きの長方形）と正しく認識された無音区間（灰色の長方形）。(c)別の認識結果の例。この例では(b)よりもマッチング誤差が小さい。

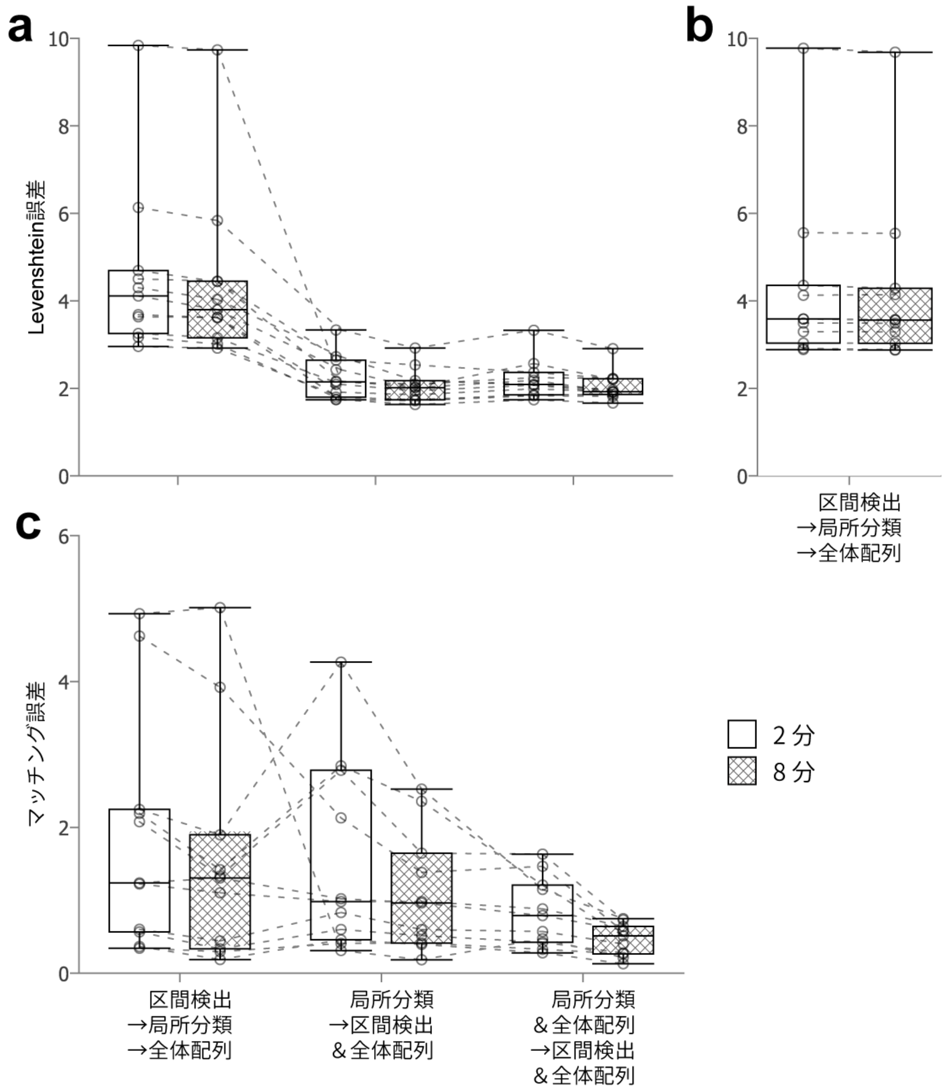


図 2.7｜汎化誤差

(a)Levenshtein 誤差。訓練用データ 2 分の結果を白色の箱ひげ図、8 分の結果を格子模様の箱ひげ図で示した。各個体の誤差値を円で表し、破線で繋いだ。(b)区間検出→局所分類→全体配列による区間検出の誤差。(c)マッチング誤差。

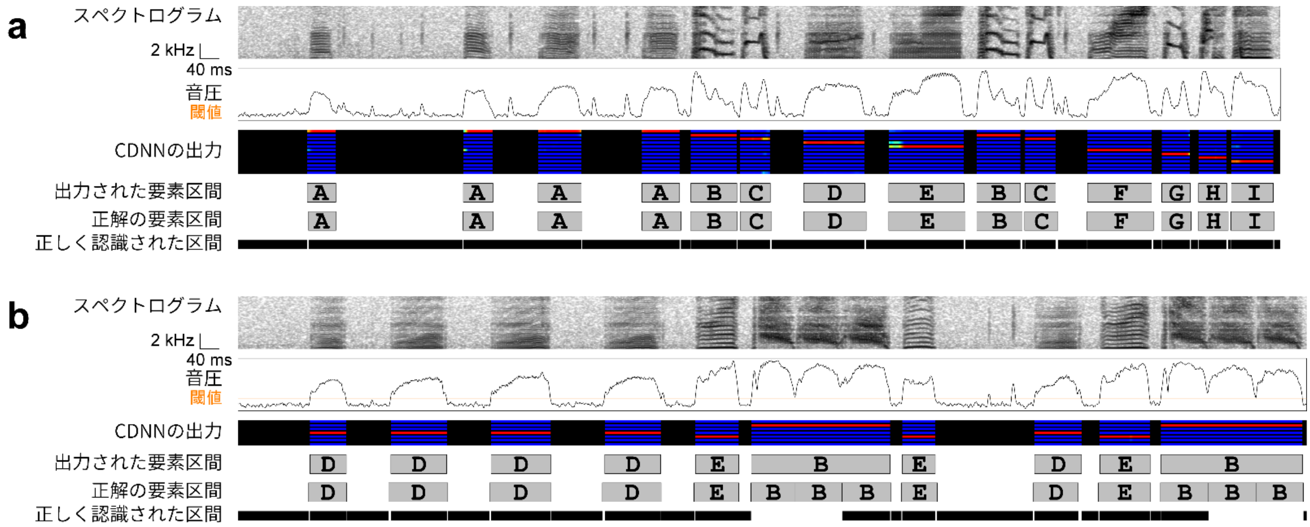


図 2.8｜区間検出→局所分類→全体配列の構成による認識結果

上から、スペクトログラム、音圧と閾値、CDNN の出力、出力された要素分類と区間、正解の要素分類と区間、正しく認識された区間。CDNN の出力では、上から要素分類ごとに出力値を示し、区間検出により検出された無音区間を黒く塗りつぶした。出力された要素区間と正解の要素区間は、灰色の長方形で表した。要素分類をアルファベットで表した。(a)と(b)に異なる個体の結果を示した。

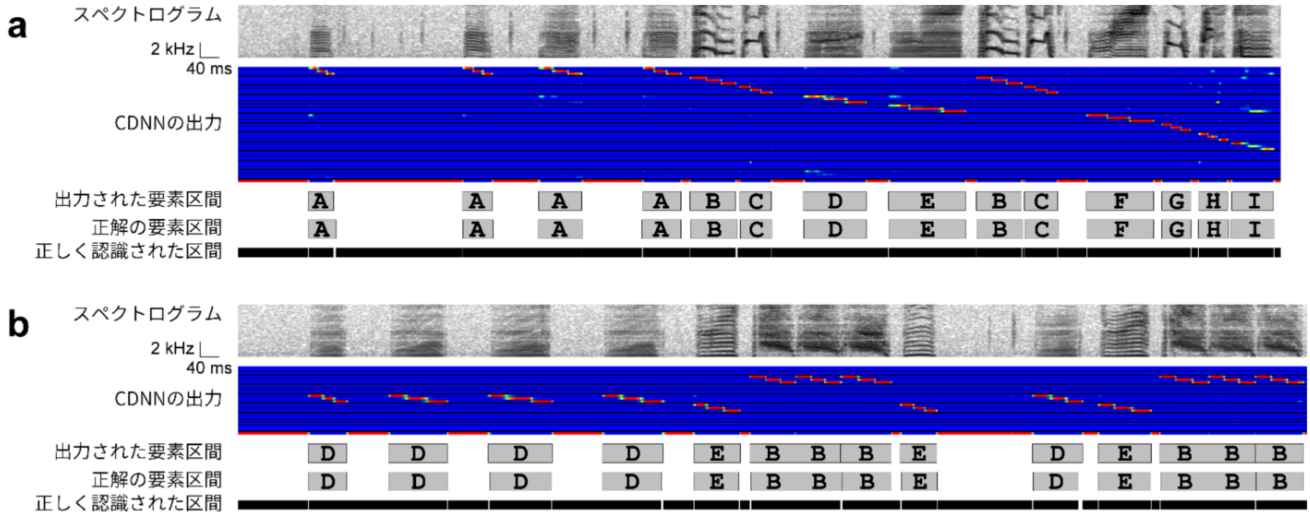


図 2.9 | 局所分類→区間検出＆全体配列の構成による認識結果

上から、スペクトログラム、CDNN の出力、出力された要素分類と区間、正解の要素分類と区間、正しく認識された区間。CDNN の出力では、上から要素分類ごとに、要素内の小区間ごとに、出力値を示した。出力された要素区間と正解の要素区間は、灰色の長方形で表した。要素分類をアルファベットで表した。(a)と(b)に異なる個体の結果を示した。

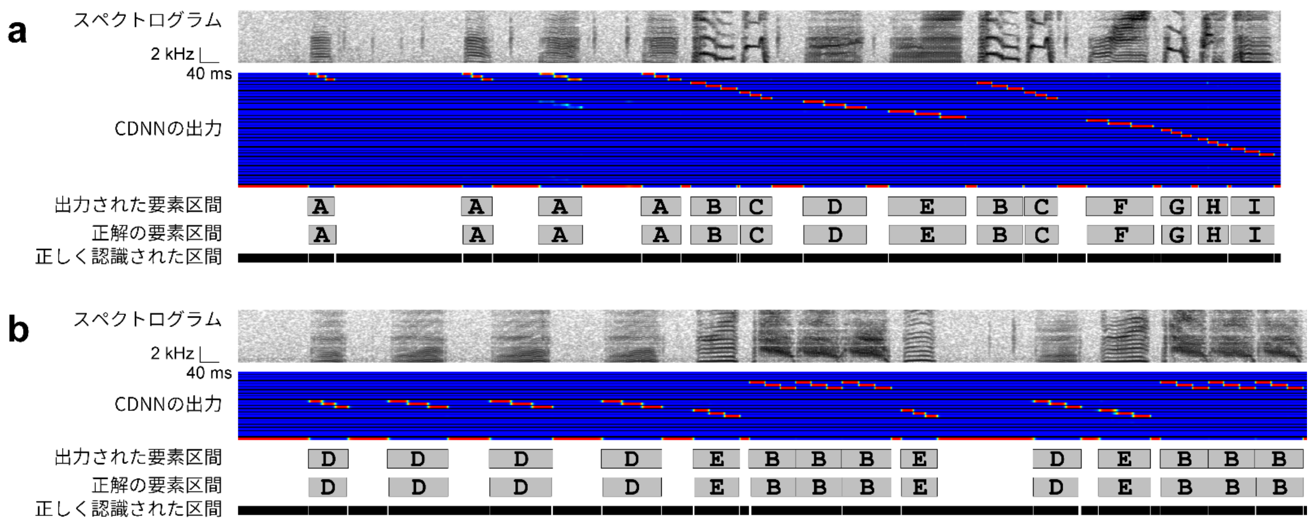


図 2.10｜局所分類＆全体配列→区間検出＆全体配列の構成による認識結果

上から、スペクトログラム、CDNN の出力、出力された要素分類と区間、正解の要素分類と区間、正しく認識された区間。CDNN の出力では、上から要素分類ごとに、要素内の小区間ごとに、出力値を示した。出力された要素区間と正解の要素区間は、灰色の長方形で表した。要素分類をアルファベットで表した。(a)と(b)に異なる個体の結果を示した。

表 2.1｜汎化誤差の平均値

	Levenshtein 誤差		マッチング誤差	
訓練用データ長	2 分	8 分	2 分	8 分
区間検出→局所分類→全体配列	1.86%	1.57%	4.57%	4.42%
局所分類→区間検出＆全体配列	1.51%	1.09%	2.25%	2.06%
局所分類＆全体配列→区間検出＆全体配列	0.84%	0.46%	2.21%	2.06%

第 3 章　歌文法の推定と時間帯による変化

導入

鳴禽の歌文法は学習によって獲得される(Lipkind et al., 2013; Menyhart et al., 2015; Okanoya, 2004a)。幼鳥は、育ての親の歌を手本とし、自身の歌を手本に近づけるように学習する。歌学習は孵化後約 120 日で完了するが、学習完了後に歌文法が全く変化しないわけではない。歌文法は学習完了後も加齢とともに徐々に変化する(James & Sakata, 2014)。また、成鳥の聴覚を剥奪すると、歌文法が数日で急激に変化する(Okanoya & Yamaguchi, 1997)。歌の特定のタイミングで雑音を提示することにより、雑音が罰として働き雑音を避けるように歌文法が変化する(Warren, Charlesworth, Tumer, & Brainard, 2012)。これらの研究から、歌文法は学習完了後も自身の歌の聴覚フィードバックを用いて能動的に維持されると考えられている。歌文法と同様に、要素の音響構造も学習完了後に能動的に維持される(Ali et al., 2013)。音響構造のばらつきが学習の方向と効率を予測することから(Sober & Brainard, 2012)、ばらつきを利用して最適な音響構造を探索していると考えられている。そこで私は、歌文法についても同様に、変動を利用して最適な歌文法を探索しているのではないかと考えた。この可能性について調べるために、数日間にわたって記録した歌を解析し、歌文法の変動を定量した。

歌文法の変動を定量するためには、まず歌文法をわかりやすく正確に記述する必要がある。これまでに、様々な手法により歌文法がモデル化されてきた(Katahira et al., 2011; Kershenbaum et al., 2014; Markowitz et al., 2013; Menyhart et al., 2015; Sasahara et al., 2006)。ジュウシマツの歌文法には出現する要素の種類が確率的に決まる分岐点が存在する（図 1.2）。多くの研究で、直前の系列が与えられた

ときの次の要素分類の条件付き確率 P(要素分類|直前の系列)で歌文法を表現している。条件付き確率による表現はモデルの構築や評価が容易で、直感的にもわかりやすいため、本研究でも同様のモデル化を行った。さらに本研究では、ジュウシマツの歌の特徴を考慮して次の 3 点について検討した。1 点目は、要素分類の出現確率が何要素前までの系列の影響を受けるかという点である。次の要素分類が直前の系列のみに依存するモデルをマルコフモデルと呼び、条件付き確率 P(要素分類|直前の系列)における直前の系列の長さをマルコフモデルの次数と呼ぶ。過去の研究では、歌文法を 1 次マルコフモデルにより表現することが多かった。しかし実際は、ジュウシマツの歌文法は 1 次マルコフモデルによる表現では不十分である(Katahira et al., 2011)。ジュウシマツの歌文法が 3 次マルコフモデルによって十分表現できるとした研究がある(Hosino & Okanoya, 2000)。他の種では、カナリアの歌文法を、次数が系列に依存するモデルで表現した研究がある(Markowitz et al., 2013)。本論文では、次数が系列に依存するマルコフモデルを可変次数マルコフモデルと呼び、次数が一定のマルコフモデルを固定次数マルコフモデルと呼ぶ。本研究では、1 次マルコフモデルと可変次数マルコフモデルのどちらが歌文法の表現に適しているかを検討した。固定次数マルコフモデルでは、記憶すべき条件付き確率が次数に対して指数関数的に大きくなる。よって本研究では、2 次以上の固定次数マルコフモデルは検討しなかった。2 点目は、歌開始部の扱いである。ジュウシマツの歌は少数種類の要素の繰り返しから始まることが多い。歌開始部における要素の繰り返し中に、要素の音響的特徴が徐々に安定し、その安定化と相関する神経活動が存在する(Rajan & Doupe, 2013)。よって、歌開始部には歌の準備という特別な機能があると考えられている。本研究では、歌開始部をその他の部分と別に扱うモデルか、他の部分と同様に扱うモデルか、どちらが歌文法のモデルとして適しているかを検討した。3 点目は、繰り返しの扱いである。ジュウシマツの歌には 1 種類の要素の繰り返しが頻繁に

出現する。繰り返しは、同じ種類の要素への分岐と扱うことで、マルコフモデルの枠組みで表現できる。鳴禽の運動前野の神経回路網をモデル化した研究によると、生体内では繰り返しをその他の分岐と同様に表現できる(Wittenbach, Bouchard, Brainard, & Jin, 2015)。一方、繰り返しをポアソン分布で表現し、繰り返し以外の分岐を 1 次マルコフモデルで表現したモデルは、繰り返しも含め全ての分岐を 1 次マルコフモデルで表現するよりも観測された系列に近い系列を生成するという研究がある(Kershenbaum et al., 2014)。この研究では、繰り返し以外の分岐を 1 次マルコフモデルで表現したという点が不十分であった可能性がある。本研究では、繰り返しと分岐を同様にマルコフモデルにより表現するか、分岐をマルコフモデルで表現し繰り返しを経験分布で表現するか、どちらが歌文法のモデルとして適するかを検討した。

マルコフモデルによる歌文法は、直前の系列を離散的に記憶し、それぞれに対応する要素分類の出現確率を出力するというモデルである。しかし、生体内で歌文法がどのように実装されているかはわからない。鳴禽の運動前野において 1 つの要素が複数の神経細胞によるフィードフォワードネットワークにより生成されるという仮説がある(Doya & Sejnowski, 1995; Fee et al., 2004; Hanuschkin et al., 2011; Wittenbach et al., 2015)。このモデルでは、複数の神経細胞が同時に活動することで要素の一部を生成する命令となり、直後に別の細胞集団が活動することにより同じ要素の次の部分を生成する命令となる。そして、要素間の分岐は複数の細胞集団間の結合によって表現される。このモデルを用いると神経細胞単位での要素発声制御をシミュレーションすることが可能であるが、モデルの粒度が細かすぎて歌文法全体を捉えることが難しい。一方マルコフモデルでは観測された現象をわかりやすく表現することはできるが、神経機構について言及することはできない。本研究では、

両者の中間にあたるモデルとして、単純なニューラルネットワークにより要素分類の出現確率を表現するというモデルを考案した。

方法

データの取得

オスの成鳥ジュウシマツを単独で防音箱に入れ、防音箱の環境に 2 日以上かけて馴化させた後、連続した 3 日または 4 日間の音声を記録した。録音は明期の 14 時間に行った。明期と暗期のサイクルは LED 照明によってそれぞれ 14 時間と 10 時間とした。餌・水は自由に摂ることができるようにした。14 羽の個体から音声を記録した。

音声データから要素系列への変換

音声スペクトログラムの目視により連続した発声を手動で抽出した。第 2 章で述べた手法により、歌要素を検出した。自動認識の結果を全て目視により確認し、稀にあった誤りを手動で修正した。目視により分類を判定できなかった要素や、複数の分類の中間に位置するような要素は、分類不能な要素とした。地鳴きには 1 種類の分類を割り当てた。歌を、地鳴きまたは 300 ms 以上の無音区間に区切られた 8 要素以上からなる要素系列と定義した。分類不能な要素が 1%以上存在した個体（2 羽）の歌は解析から除外し、残りの 12 羽の歌について以降の解析を行った。歌の解析は個体ごとに行った。

データ集合

歌の集合をランダムに 4 分割し、4 分割交差検証により文法モデルを評価した。ハイパーパラメータの調整が必要であった場合、訓練用データを更に 4 分割し、訓練用データ内で 4 分割交差検証による尤度を最大とする値を用いた。

歌文法の推定

本研究では、歌文法のモデルとして、1 次マルコフモデルか可変次数マルコフモデルか、歌開始部を別に扱うかどうか、繰り返しを別に扱うかどうかの、合計 8 種類のモデルから観測された要素系列を最も良く予測するモデルを選択した。可変次数マルコフモデルを構築するために、まず観測された要素系列を表現する確率接尾辞木を構成した。確率接尾辞木は可変次数マルコフモデルとほぼ等価なモデルで、節に部分列とその部分列が与えられたときの条件付き確率が付与される。また、根に付与される部分列は長さ 0 の空の列で、節の位置が深いほど付与される部分列が長くなる。確率接尾辞木は Bejerano & Yona (2001)の手順で構成した（付録 E）。ハイパーパラメータは、節の候補として採用する部分系列の最小出現確率P_{min}を$P_{min} \in [1/|\Sigma|^3, 1/|\Sigma|]$、ある部分系列をその最大長接尾辞の下位の節として採用するための、その部分系列が与えられたときの条件付き確率の最小値C_{min}を$C_{min} \in [1/|\Sigma|^2, 2/|\Sigma|]$、その際の条件付き確率の変化の割合の最小値$r$を$r \in [1.2, 2]$、節として採用する部分系列の最大長$L$を$L \in [1, 20]$、条件付き確率の平滑化定数$\gamma$を$\gamma \in [1 \times 10^{-6}, 1/|\Sigma|\,]$の範囲から訓練用データ内の交差検証により決定した。可変次数マルコフモデルは、確率接尾辞木のそれぞれの節に付与された部分列と条件付き確率を用いて構築した。

ある部分系列についての、可変次数マルコフモデルによる尤度は、次のように計算される。

$$尤度 = \left(\prod_{i=1}^{n} P(\sigma_i | suffix_{\mathrm{i}-1}) \right)^{\frac{1}{n}}$$

ただしnを部分系列の長さ、σ_iをi番目の要素分類、$suffix_{\mathrm{i}-1}$をσ_iの直前の部分系列の中で、モデルに含まれる最長のものとする。

歌文法を 1 次マルコフモデルでモデル化するときは、文法モデルは節に付与される部分系列の最大長Lを 1 とした確率接尾辞木から構成した。歌開始部を別に扱うときは、確率接尾辞木の各節に、歌開始部から始まる部分列を持つ葉を、葉を接続する条件を満たしたとき接続した。繰り返しを別に扱うときは、要素系列中で同じ分類の要素が複数回繰り返している系列を 1 要素で置き換え、確率接尾辞木を構成した。そして、各要素分類について繰り返し回数の分布を別に計算した。繰り返しでない要素も繰り返し回数 1 として扱った。予測の段階で尤度を計算するときは、繰り返しを省略した系列に対するマルコフモデルの尤度と、繰り返し回数の分布による尤度を掛け合わせた。繰り返し回数の分布は、指数分布と足しあわせ、合計が 1 となるようにすることで平滑化した。ここでは次のような指数分布を用いた。

$$P(x) = P_1 r^{-(x-1)}$$

ただしxを繰り返し回数、P_1を繰り返し回数が 1 回である確率、rを繰り返しが 1 回増えるごとに確率が減少する割合とした。ハイパーパラメータP_1とrは$P_1 \in [1 \times 10^{-3}, 1 \times 10^{-1}]$、$r \in [1.1, 2]$の範囲から訓練用データ内の交差検証により決定した。平滑化に指数分布を用いることにより、訓練用データ中に存在しないいかなる繰り返し回数についても確率が 0 にならないようにした。歌開始部

を別に扱う場合、歌開始部から始まる繰り返しと、それ以外の繰り返しのそれぞれについて繰り返し回数の分布を計算した。

歌文法の分割単位の推定

歌文法が時間帯によって変化するかどうか、変化するとしたらどのように変化するのかを調べるために、音声を記録した期間を分割し、条件付き確率を区間ごとに計算した。分割は可変次数マルコフモデルの分岐点ごとに行った。系列ごとに、訓練用データ内の交差検証により最適な分割数を決定した。区間の長さは、全期間・1 日（14 時間の明期）ごと・14 時間を 1 から 28 の区間数に分割、の範囲で探索した。

系列の履歴を入力とするモデル

出現する要素分類の確率を、直前の系列の履歴情報をもとに予測するモデルを作成した。履歴情報の強さが時間とともに指数関数的に減衰すると仮定することで、モデルはその要素が生成された時刻を推定できる。具体的には、要素ごとの履歴情報を表現するベクトルをxとすると、

$$x_i = \sum_{t_i<T} \exp\left(\frac{-(T-t_i)}{\tau_i}\right)$$

ただしTは現在の時刻、t_iは分類がiである要素の開始時刻、τ_iは要素分類iについての時定数である。このベクトルを入力とし、各要素分類についての確率を表すベクトルを出力する、中間層 1 層の全結合ニューラルネットワークにより歌文法を表現した。履歴情報の減衰時定数は、計算時間の短縮のため全要素分類で等しいとし、$\tau_i \in [50\ \mathrm{ms}, 300\ \mathrm{ms}]$の範囲から訓練用データ内での交差検証により決定した。中間層の次元は、事前に少数のデータを用いて検討し、128 とした。ニューラルネットワ

ークの出力層は、離散確率分布を表現するためにソフトマックス層とした。学習と予測には、要素の開始時刻における入力ベクトル$\boldsymbol{x}$とその要素の分類の組のみをデータとして用いた。ネットワークのパラメータは第 2 章と同様に(He et al., 2015)の方法により初期化した。パラメータの更新には(Kingma & Ba, 2014)の方法を用いた。訓練用データ内をさらに分割して作成した小検証用データに対する予測誤差が改善しなくなるまでパラメータを更新した。訓練の際には交差エントロピー損失関数を用いて予測誤差を計算した。

統計検定

モデル間の尤度は、Wilcoxon 符号順位検定を用いて個体内で比較した。必要に応じて Bonferroni 補正を行った。本論文では、可読性を上げるために補正後の p 値を記述したので、p 値が 1 を超える場合があるが、その場合は 1 とした。

結果

データ

14 羽の個体から連続した 3 日または 4 日間の音声を記録した。歌に含まれる全ての要素を第 2 章で述べた手法により検出・分類した。分類不可能な要素が 1%より多く含まれていた個体（2 羽）を解析から除き、残りの 12 羽の歌を以降の解析に用いた。1 日あたりの歌の合計長の平均値±標準偏差は 45.5±22.3 分、歌の長さの個体内平均値は 7.5±2.4 s であった。1 日あたりの要素の総数は 20701.8±10551.9、要素の種類数は 7.8±3.6 であった。個体によって歌が大きく異なったので、解析は個体ごとに行った。

歌の要素系列

歌の音声を第 2 章の方法により要素系列に変換した。ある個体の歌の要素系列の一部を図 3.1 に示した。この個体では、歌の開始部には A の繰り返しの後に BCCC が多く出現していた。開始部以外にも F の繰り返しが頻繁に見られた。また、BCCBD という部分列が頻出した。以上より、要素の出現順序には何かしらの規則があることが予想される。しかし、要素系列が膨大かつ複雑なので、要素系列を観察しただけでは全体の規則はわからない。よって、要素の出現順序に関する規則を確率モデルで表現することが有効であると考えられる。

歌開始部の系列パターン

図 3.1 より、歌開始部に特徴的な系列パターンが存在すると予想できる。これを確認するために、歌を要素数が均等になるように 3 分割し、それぞれに含まれる要素の種類を数えた（図 3.2a）。この個体では、歌の初めの 1/3 に要素 A と C が特に多く出現していた。歌を 3 分割したときのそれぞれの区間における要素分類の分布（$F_{1/3}$、$F_{2/3}$、$F_{3/3}$）が、歌全体における要素分類の分布（F_0）とどの程度異なっているかを調べるために、$F_{1/3}$ から $F_{3/3}$ に対する F_0 の Kullback‐Leibler divergence（KLD）を計算した。この個体では、F_0 の $F_{1/3}$ に対する KLD（$D(F_0|F_{1/3})$）は 0.23 bit、$D(F_0|F_{2/3})$では 0.02 bit、$D(F_0|F_{3/3})$では 0.06 bit であった。つまり、歌の後半の 2/3 では、前半の 1/3 よりも要素分類の分布が歌全体の分布と近かった。各個体で同様の計算をしたところ、$D(F_0|F_{1/3})$が、$D(F_0|F_{2/3})$と $D(F_0|F_{3/3})$よりも大きかった（図 3.2b、p 値を 3 倍に補正した Wilcoxon 符号順位検定により、$D(F_0|F_{1/3})$と $D(F_0|F_{2/3})$で $W = 71$、$p = .029$、$D(F_0|F_{1/3})$と $D(F_0|F_{3/3})$で $W = 78$、$p = .0015$）。一方、$D(F_0|F_{2/3})$と $D(F_0|F_{3/3})$に有意な差はなかった（$W = 54$、$p = .80$）。よって、歌のはじめの 1/3 では要素分類の分布が歌全体とは異

なることが示された。要素分類の分布が異なるので、歌文法も異なる可能性が高い。

繰り返し回数の分布

記録した歌には、同じ種類の要素の繰り返しが多く見られた。図 3.1 に示した個体における要素分類ごとの繰り返し回数の分布を図 3.3a に示した。繰り返しの無い要素分類（D と G）も、繰り返し回数のばらつきが大きい要素分類（A、C、F）もあった。全個体の歌における、繰り返し回数の最頻値の分布を図 3.3b に示した。繰り返しの無い要素分類が最も多く、それ以外の要素分類では最頻値は 2 回から 7 回の範囲でばらついていた。繰り返し回数の多い系列は単純な 1 次マルコフモデルでは表現できない。よって繰り返しの扱いについて検討する必要がある。

歌文法の推定

ここまでの結果から、歌文法を確率モデルによって記述するためには、直前の系列への依存性・歌開始部に特徴的な系列パターン・1 種類の要素の繰り返し、について考慮すべきであることがわかった。よって本研究では、歌文法のモデルとして、1 次マルコフモデルか可変次数マルコフモデルか、歌開始部の系列を別に扱うかどうか、繰り返しを別に扱うかどうか、のそれぞれについてどちらかを選んだ合計 8 種類のモデルを計算した。

図 3.1 の個体から推定した条件付き確率を表 3.1 に示した。1 次マルコフモデルでは、直前に要素が存在する場合は、要素分類の出現確率が直前の要素分類によって変化する。可変次数マルコフモデルでは、直前の系列内でモデルに含まれる最長の部分列を元に要素分類の出現確率が決まる。可

変次数マルコフモデルでは、要素分類の出現確率が直前の最長 12 要素に依存するという結果になった。もしこの歌文法を 12 次マルコフモデルで表現しようとすると、この個体の歌には要素が 7 種類出現したので、$7^{12} \sim 1.4\times10^{10}$ の条件付き確率を記憶しなければならない。一方、可変次数マルコフモデルでは、記憶すべき部分列の最大数は 79 であった。よって、可変次数マルコフモデルでは系列の規則を利用して記憶する部分列を減らしていることがわかる。

歌開始部の系列を別に扱うモデルでは、直前の部分列が歌開始部かどうかによって、要素分類の出現確率を変化させた。繰り返しを別に扱うモデルでは、同じ要素分類への遷移を禁止したマルコフモデルと、繰り返し回数の分布（図 3.3 と図 3.4）を組み合わせて要素分類の出現確率を決定した。歌開始部の系列も繰り返しも別に扱うモデルでは、繰り返しの分布は歌開始部とそれ以外の部分とで分けて計算した。

予測精度の比較

推定した歌文法が観測された系列をどの程度正確に予測するかを調べるために、交差検証による予測精度を計算した。訓練用データによって推定した歌文法を用いて、検証用データの尤度を計算した（図 3.5）。結果として、歌開始部を別に扱い、繰り返しを分岐の一種として扱う、可変次数マルコフモデルにおいて尤度が最大となった（p 値を 7 倍に補正した Wilcoxon 符号順位検定により、歌開始部を同様に扱うモデルとの比較で、$W = 77$、$p = .0068$、その他のモデルとの比較で $W = 78$、$p = .0034$）。尤度が大きいほど予測精度が高いことを意味するが、尤度が取りうる最大値は exp（その文法の条件付きエントロピー）であるので、確率的な文法では尤度が 1 になることは無い。

時間帯による歌文法の変化

次に、歌文法が時間帯によって変化するのか、変化するとしたらどのように変化するのかを調べた。ある個体の歌のある分岐点について、1 日（14 時間の明期）を 4 分割し、それぞれの区間で条件付き確率を計算した（図 3.6)。条件付き確率が時間帯によって変化しているようにみえる。また、変化のパターンが日をまたいで類似しているようにみえる。

全ての分岐点について系統的に調べるために、データを時間帯によって分割し（図 3.7d)、それぞれの区間ごとに条件付き確率を計算した。分割の細かさは、分岐点ごとに、訓練用データ内の交差検証による尤度が最大となるように決定した。最適な分割の細かさは図 3.8 のように分布した。全ての個体において少なくとも 1 つの分岐点で最適な分割数が 2 以上であった。つまり、分割した時間帯ごとに歌文法を計算したほうが観測された系列を良く説明した。さらに、最適な細かさで時間帯に分割した歌文法を用いて条件付き確率を計算し、検証用データでの尤度を計算したところ、分割しないモデルよりも上昇した（図 3.7e、p 値を 3 倍に補正した Wilcoxon 符号順位検定により、W = 75、p = .0073)。つまり、時間帯に分割することで、条件付き確率をより良く予測できた。この結果から、歌文法が時間帯によって変化することが示唆された。

次に、変化のパターンをより詳細に調べるために、データを日ごとに分割して条件付き確率を計算したところ（図 3.7c)、分割しないモデルよりも交差検証による尤度が上昇した（p 値を 3 倍に補正した Wilcoxon 符号順位検定により、W = 76、p = .0044)。よって、条件付き確率が日ごとに異なることが示唆された。最後に、異なる日の同じ時間帯をまとめて条件付き確率を計算したところ（図

3.7b)、分割しないモデルよりも交差検証による尤度が上昇した（W = 77、p = .0029)。よって、時間帯による変動が日をまたいで類似することが示唆された。まとめると、歌文法が日によって異なり、時間帯による変化が日をまたいで共通することを示唆する結果が得られた。異なる分割の方法で尤度は有意に異なっていたが、その差は小さかった。よって時間帯によって変動するが、大きく変動することはなかった。

系列の履歴を用いたニューラルネットワークによる要素分類の予測

　モデル選択の結果、ジュウシマツの歌文法では、次に出現する要素の確率は 10 要素程度過去の系列に依存し、歌開始部は他の部分と異なる文法になっており、繰り返しを分岐の一種として扱う、ということがわかった。しかし、ここで検討した文法モデルは要素分類の出現規則を表現したものにすぎず、歌文法の生体内における実装までは言及できない。そこで、生体内における実装として可能な形で歌文法を表現するために、系列の履歴を入力とするニューラルネットワークにより歌文法をモデル化した。ここまでのモデルでは過去の系列中の要素を離散的に扱っていた。ニューラルネットワークによるモデルでは履歴情報の強さが徐々に減衰すると仮定し、過去の系列を要素分類数次元のベクトルとして連続的に表現した（図 3.9)。履歴情報の減衰時定数は訓練用データ内による交差検証を用いた汎化性能を最大化するように決定した。履歴情報の減衰時定数の最適値は、197.4 ±159.2 ms となった。交差検証による尤度は、可変次数マルコフモデルによる汎化性能よりも全個体で大きくなった（図 3.10、Wilcoxon 符号順位検定により、W = 78、p < .001)。全個体で尤度の平均値±標準偏差は 0.83±0.08 であった。したがって、履歴情報を入力とするニューラルネットワークによるモデルは、生体内で実装可能であり、かつ、観測された系列を可変次数マルコフモデルよ

りも良く予測した。

考察

可変次数マルコフモデル

ジュウシマツの歌文法が可変次数マルコフモデルで表現できたという結果は、ジュウシマツの歌が複数要素からなるチャンクから構成されているという過去の研究の結果と無矛盾である(Okanoya, 2004a)。一般に、固定次数マルコフモデルでは次数が増えると記憶すべき部分系列が指数関数的に増大するため、生体が大きな次数の固定次数マルコフモデルを用いて歌を生成しているとは考えにくい。複雑な歌文法を可変次数マルコフモデルによって表現することで、系列の履歴を長く遡るモデルでも記憶する部分系列の数を抑えるコンパクトな表現ができた。(Bejerano & Yona, 2001; Ron, Singer, & Tishby, 1996)。

歌開始部

ジュウシマツの歌は開始部に特有の要素の繰り返しから始まることが多い。しかし、個体によっては歌開始部に特有の要素が複数種類あったり、その要素が歌の後半にも出現したりと、歌開始部だけを単純に別扱いすればよいというわけではない。本研究では、歌開始部の系列に固有の条件付き確率を割り当てることで、歌開始部とそれ以降の系列を同じ可変次数マルコフモデルの枠組みの中で表現した。本研究の結果は、歌開始部には歌の準備という特別な機能があることを示唆する過去の研究と無矛盾である(Rajan & Doupe, 2013)。

繰り返し

ジュウシマツの歌には同じ種類の要素の繰り返しが頻繁に出現する。繰り返しは、単純なマルコフモデルや、遷移確率が繰り返しに従って徐々に変化するような適応的マルコフモデルを用いてもうまく表現できない(Wittenbach et al., 2015)。本研究では繰り返しを分岐の一種として扱い、繰り返しを含んだ直前の系列によって次の要素の出現確率が変化するというモデルを用いることで、観測された系列を精度良く予測することができた。記憶すべき部分系列に繰り返しを含めると、それだけ記憶容量が多く必要になる。本研究では可変次数マルコフモデルを用いることで必要な記憶容量を少なくすることができた。本研究の結果は、繰り返しと分岐を同じ神経回路で生成できるという過去の研究と無矛盾である(Wittenbach et al., 2015)。

歌文法の変化

歌文法を時間帯ごとに計算することで、歌文法が徐々に変化することがわかった。さらに、異なる分割パターンで予測性能を計算することで、歌文法が日によって変化し、時間帯による変化が日をまたいで類似することがわかった。過去の研究によると、脳の温度が概日リズムを持って変化する(Aronov & Fee, 2012)。本研究でも明期と暗期の時間は 14 時間と 10 時間に保たれていたため、脳の温度に限らず、個体の状態が概日リズムを持って変化していた予想できる。歌文法が時間帯によって変化するということは、歌文法が個体の内部状態によって変化することを示しているかもしれない。鳴禽は自身の歌を能動的に維持していると考えられている(Charlesworth, Warren, & Brainard, 2012)。様々な状態でうたうことで、文法にばらつきをもたせ、探索学習を効果的にしているのかもしれない。

歌文法は時間帯によって変動していたが、その変動の強度は小さかった。これは、歌学習完了後は歌が大きく変動することがないからかもしれない。一般に、学習が完了すると、探索範囲が大きすぎないほうが良いパフォーマンスを発揮することができる。成鳥の歌文法の変動強度が小さいのは、探索と利用のバランスが最適化された結果によるものかもしれない。

系列の履歴を入力とするニューラルネットワークによる歌文法

過去の系列の履歴を入力とするニューラルネットワークを用いて歌文法を表現することにより、観測された系列を可変次数マルコフモデルによる歌文法よりも精度よく予測することができた。ニューラルネットワークでは中間層の値のパターンにより内部状態を表現するため、系列の履歴が与えられたときの要素分類の条件付き確率を、可変次数マルコフモデルとは異なる形でコンパクトに表現できる。実際、本研究で用いたニューラルネットワークは中間層の次元が 128 であり、可変次数マルコフモデルに必要な記憶容量と同程度であった。尤度が約 0.8 ということは、確率的な系列ではあるが次に出現する要素分類をかなり決定的に予測できるということである。本研究によって、系列の履歴を入力とするニューラルネットワークにより歌文法を表現できるということがわかった。

ここでの系列の履歴として具体的に考えられるのは、生体内における聴覚フィードバック、体性感覚フィードバック、または脳内フィードバックの複合である。本研究ではニューラルネットワークの入力へのそれぞれの貢献度を分離することはできない。

本研究を含め過去の多くの研究で、マルコフモデルなど系列を離散的に扱うモデルが歌文法の表

現に用いられてきた。これらのモデルでは観測された現象を表現するのみで、神経機構について言及することができない。一方、神経細胞のフィードフォワードネットワークにより歌の生成機構を説明しようとするモデルでは、神経細胞単位でモデル化するため、歌文法全体を表現しにくい。計算モデルを用いた研究の良いところは、観測された現象とそれを説明する細かい神経機構の間をつなぐような、中間の粒度のモデルを作成することができる点である。本研究では、ニューラルネットワークという生体内で実装可能なモデルを用いて、観測された要素系列を説明できるモデルを構築することに成功した。

AAAAAAABCCCBABCCC-DCEEFFFFFFFFAABCCBDGCE...
AAAABCCCABCCCBABCCCABCEEFF-AABCCBDGCEEFF...
AAAAAABCCCBABCCBDCEEFFFFFFAABCCBDGCEEFFF...
AAAABCCCBABCCBDCEEFFFFFAABCCBDGCEEFFFFAA...
AAAABCCCBABCCBDCEEFFFFFFAABCCBDGCEEFFFFF...
AAAAAABCCCBABCCCBABCCBDCEEFFFFFFF-AABCCB
AAABCCCBABCCBDAABCCBDCEEFFFF
AAAAABCCC

図 3.1 | 要素系列

ある個体から記録した歌の要素系列の一部を示した。上 5 行の歌は長かったため、前半の一部のみを示した。赤色 : 歌開始部に見られた要素 A の繰り返し。青色 : 歌開始部に見られた部分列 BCCC。紫色：頻出した部分列 BCCBD。緑色：F の繰り返し。

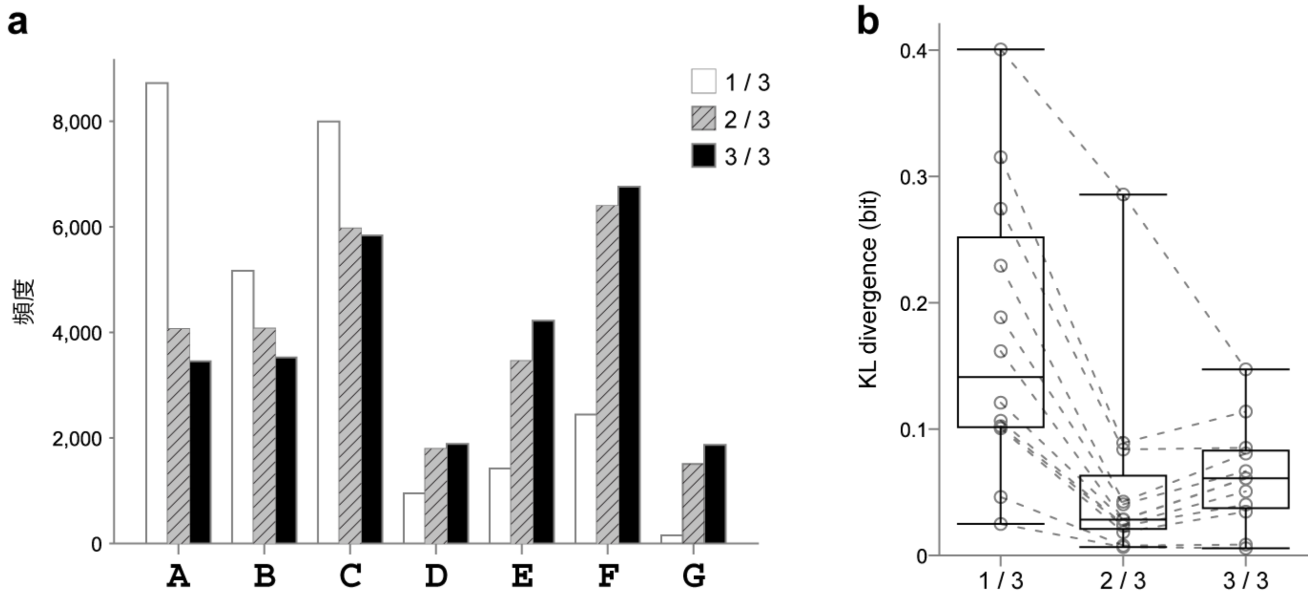


図 3.2 | 歌を 3 分割したときの要素分類の分布

(a)ある個体の歌を、要素数が均等になるように 3 分割し、それぞれの区間における要素分類の分布。要素の分類をアルファベットで、区間の位置を色で表した。白色：はじめの 1/3 の系列。灰色斜線：中間の 1/3 の系列。黒色：おわりの 1/3 の系列。(b)それぞれの区間における要素分類の分布に対する、歌全体における要素分類の分布の Kullback–Leibler divergence。各個体の値を円で示し、破線で繋いだ。

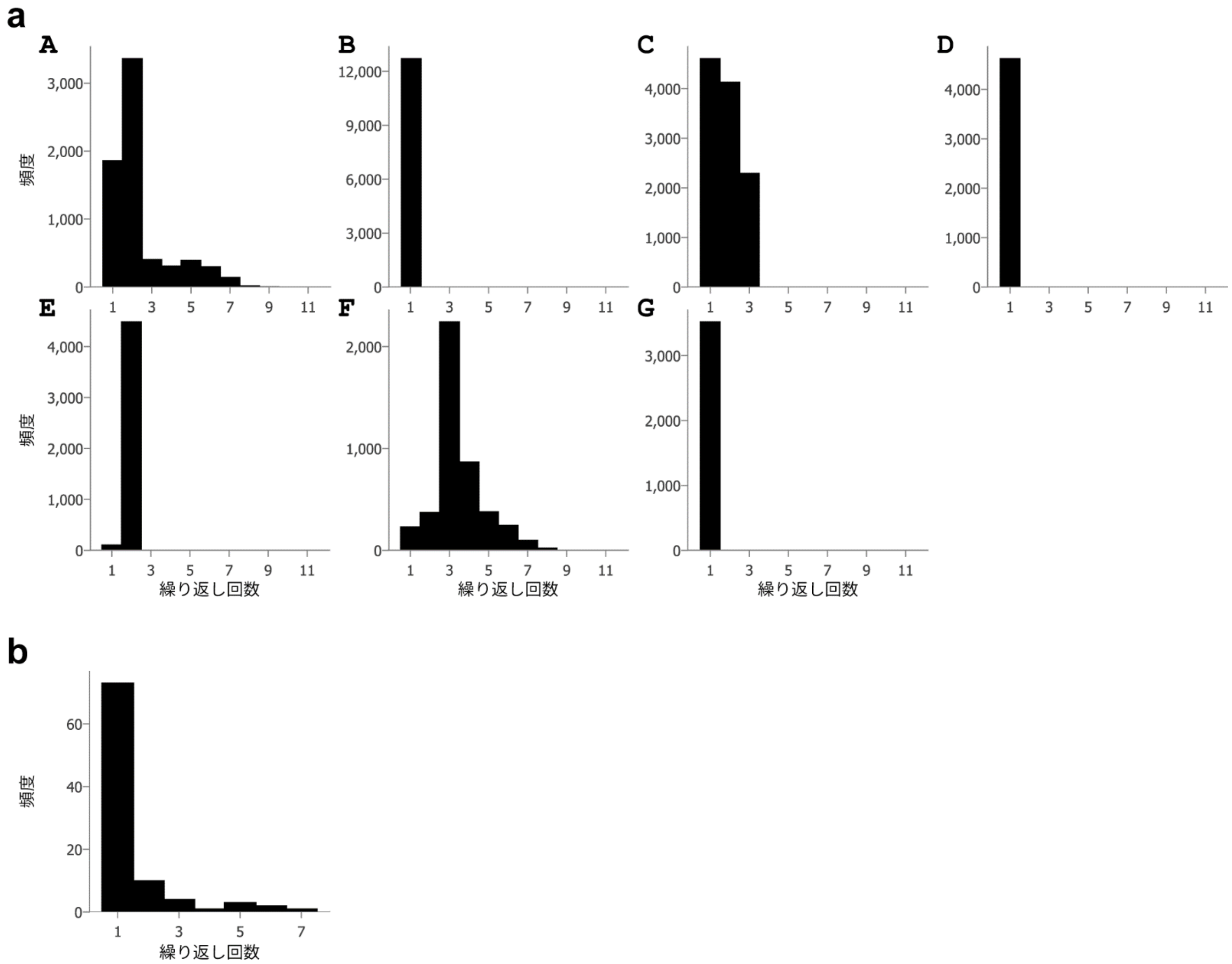


図 3.3 | 繰り返し回数の分布

(a)ある個体における 1 種類の要素の繰り返し回数の分布。図の左上に要素分類をアルファベットで示した。(b) 全個体における、繰り返し回数の最頻値の分布。各個体で要素分類ごとに 1 つの最頻値が計算できる。その値を全個体でまとめて集計したもの。繰り返しの無い要素も繰り返し回数 1 として扱った。

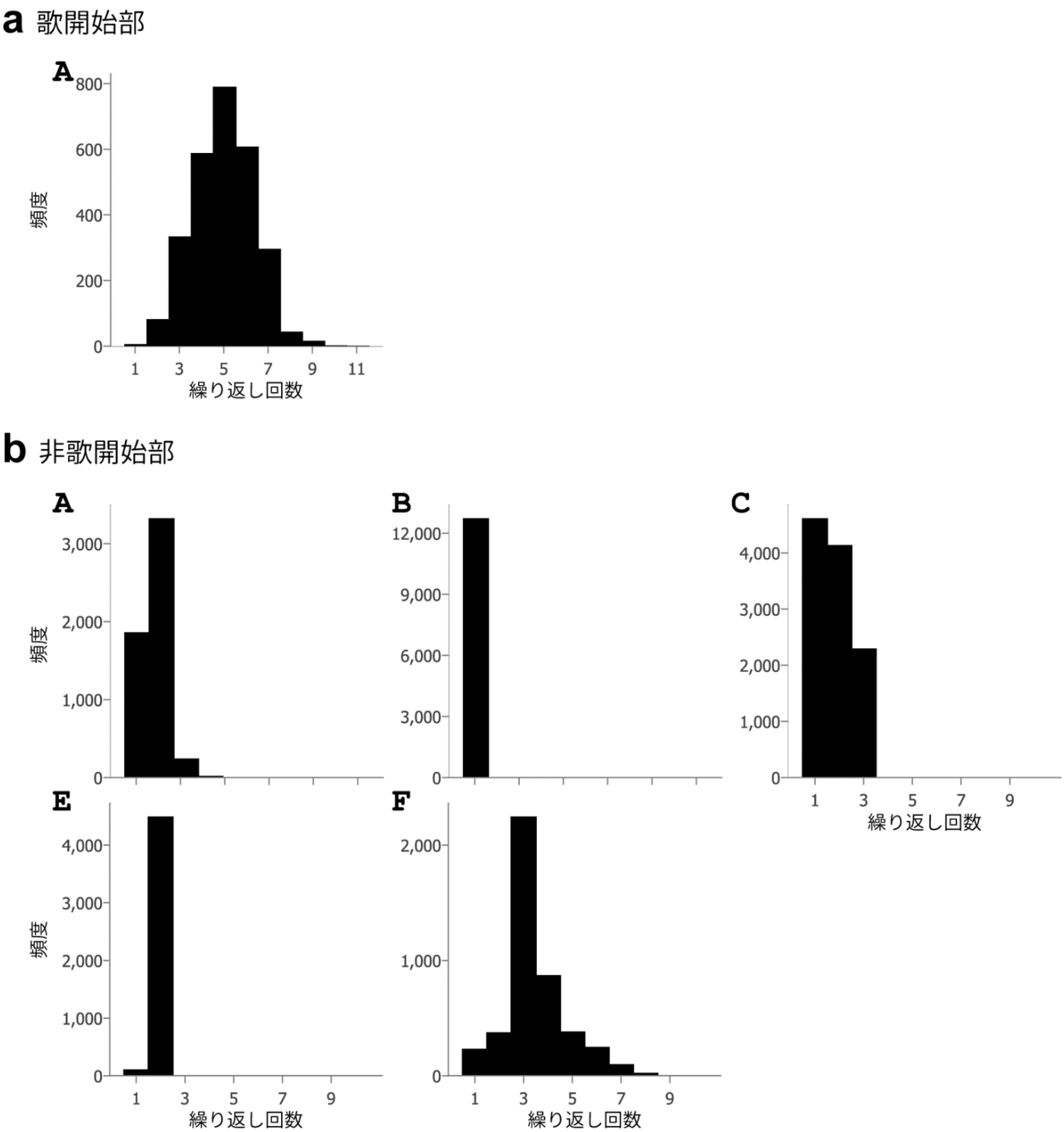
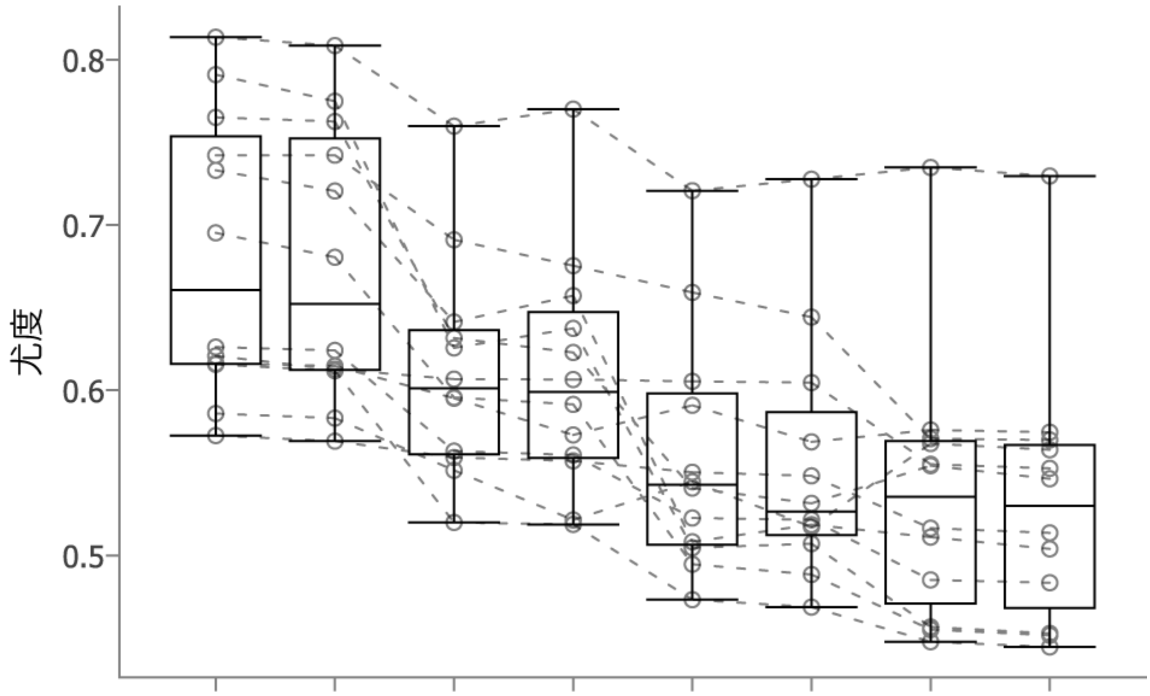


図 3.4 | 歌開始部を別扱いしたときの繰り返し回数の分布

ある個体における、歌開始部を別扱いしたときの繰り返し回数の分布。要素分類を図の左上にアルファベットで示した。繰り返しの無かった要素分類は省略した。(a)歌開始部の繰り返し回数の分布。(b)歌開始部以外の部分の繰り返し回数の分布。



可変次数	✓	✓	✓	✓				
歌開始部を別扱い	✓			✓		✓	✓	
繰り返しを別扱い			✓	✓	✓	✓		

図 3.5｜モデルごとの尤度

個体ごとに計算した交差検証による尤度を箱ひげ図で示した。横軸はモデルの種類を表す。それぞれの設定ごとに、あてはまるものにチェックを示した。モデルは、尤度の平均値が大きい順に並べた。

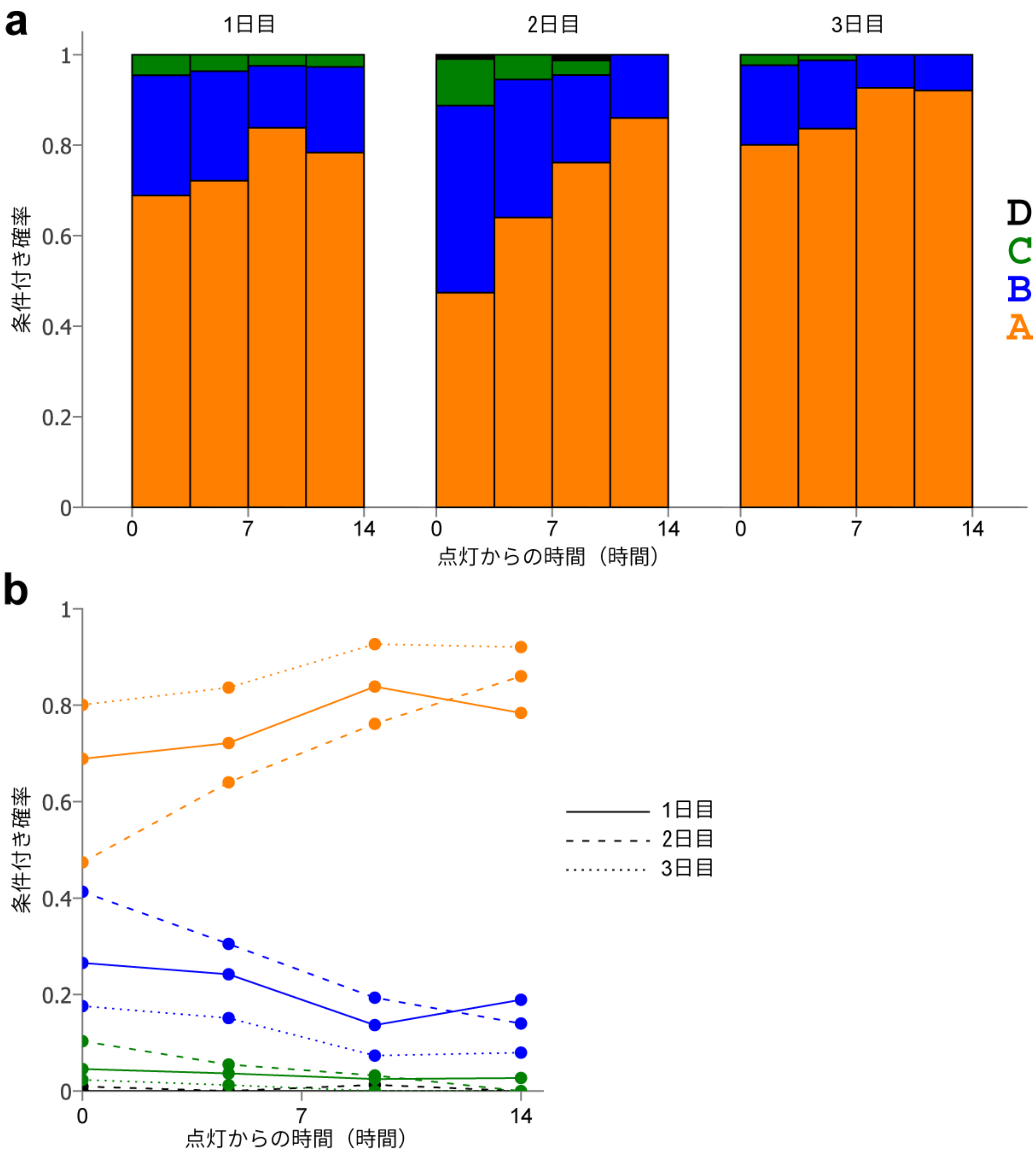


図 3.6 | 時間帯ごとの条件付き確率

ある個体のある分岐点における条件付き確率を、日ごとに 1 日を 4 分割して計算した。要素 A が出現する確率をオレンジ色、要素 B を青色、C を緑色、D を黒色で表した。(a)条件付き確率を日ごとに並べた。(b)全ての日の条件付き確率を重ねて示した。

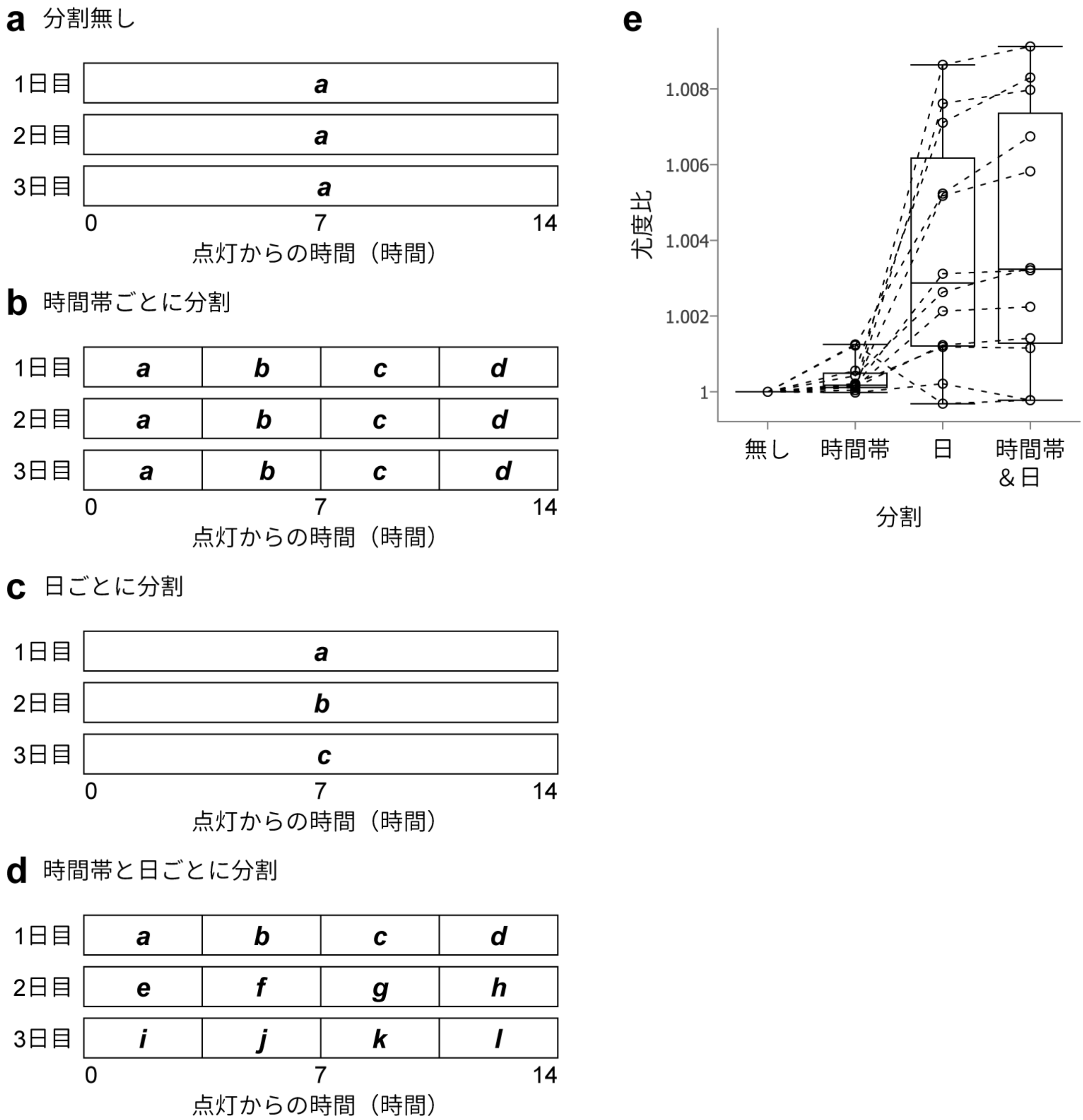


図 3.7 | 時間帯による変化の検討

(a-d)分割の方法。同じアルファベットの区間の系列を用いて、条件付き確率を計算した。(a)分割しない場合。(b)時間帯ごとに分割する場合。異なる日の同じ時間帯の系列を用いて、条件付き確率を計算した。(c)日ごとに分割する場合。条件付き確率を日ごとに計算した。(d)時間帯と日ごとに分割する場合。全ての区間で独立に条件付き確率を計算した。(d)分割したモデルによる尤度。分割しな

い場合の尤度との比を示した。

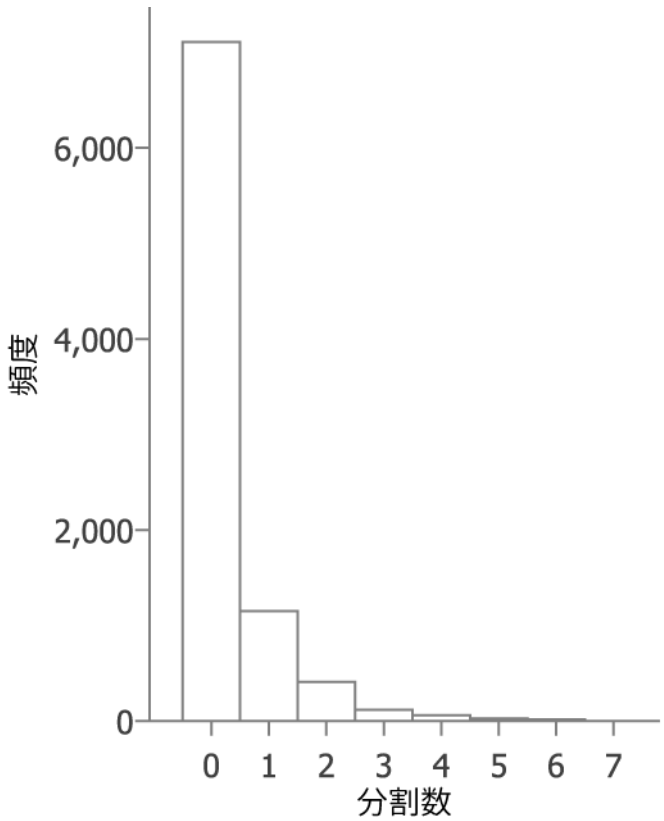


図 3.8｜明期の期間の最適な分割数の分布

可変次数マルコフモデルのそれぞれの分岐点において、明期の 14 時間を分割する最適な分割数の分布を、全個体まとめて示した。分割数 0 は、分割しないことを表す。分割数 1 は、1 日単位で分割することを表す。分割数 2 以上は、明期の 14 時間をその数に分割することを表す。

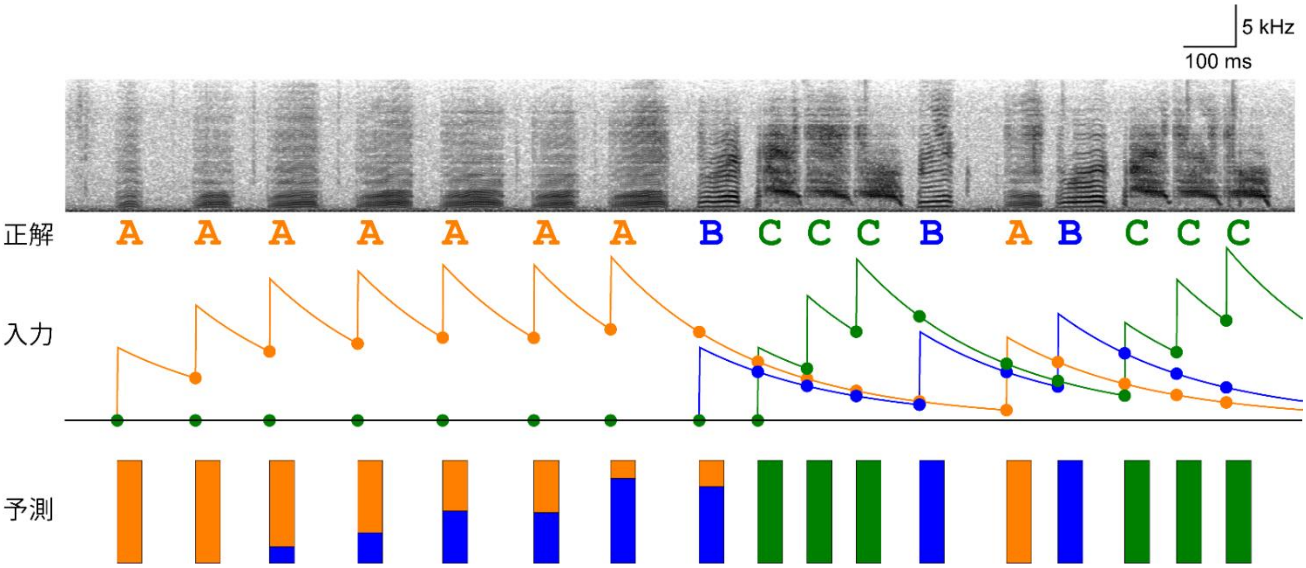


図 3.9｜フィードバックによる要素分類の出現確率の予測

上から、ある個体の歌、正解の要素分類、フィードバックによる入力値の時間変化、要素分類の出現確率の予測値。時刻に沿ったフィードバックの値を実線で示し、実際にモデルへ入力した値を円で示した。出現確率の予測値は、各要素分類の出現確率を帯の高さで示した。要素 A、B、C をそれぞれオレンジ色、青色、緑色で表した。

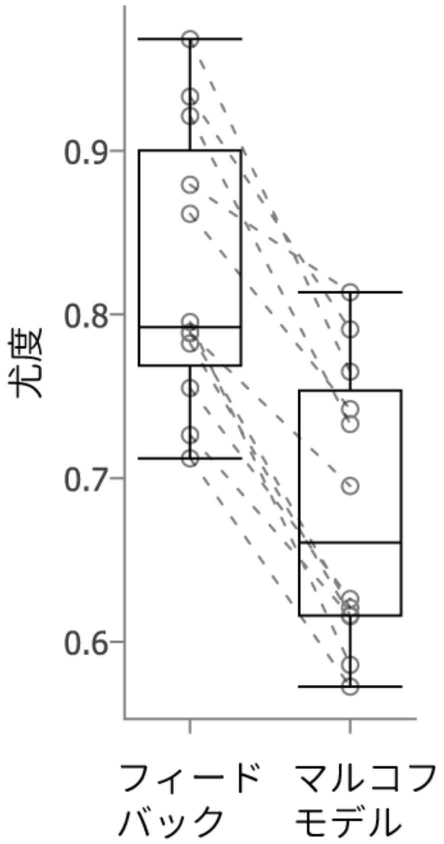


図 3.10｜フィードバックによるモデルと可変次数マルコフモデルの尤度

各個体の尤度値を円で示し、破線で繋いだ。マルコフモデルの尤度は図 3.5 と同一である。

表 3.1 | 推定された条件付き確率

各行に直前の系列を、列に次の要素分類を示した。(a)歌開始部は他の部分と同様、繰り返しは分岐と同様とした 1 次マルコフモデル。(b) 歌開始部は他の部分と同様、繰り返しは分岐と別とした 1 次マルコフモデル。(c) 歌開始部は他の部分と別、繰り返しは分岐と同様とした 1 次マルコフモデル。(d) 歌開始部は他の部分と別、繰り返しは分岐と別とした 1 次マルコフモデル。(e)歌開始部は他の部分と同様、繰り返しは分岐と同様とした可変次数マルコフモデル。(f) 歌開始部は他の部分と同様、繰り返しは分岐と別とした可変次数マルコフモデル。(g) 歌開始部は他の部分と別、繰り返しは分岐と同様とした可変次数マルコフモデル。(h) 歌開始部は他の部分と別、繰り返しは分岐と別とした可変次数マルコフモデル。歌開始部の部分系列には、先頭に X を付けた。条件付き確率を青から赤へのカラースケールにより表した。繰り返しを禁止したマルコフモデルでは、同じ種類の要素へ遷移する条件付き確率は 0 となる。条件付き確率が 0 である場合は表を空欄とした。0 より大きく 0.01 より小さい条件付き確率は背景を白色にした。

a 次数 一次
歌開始部 同
繰り返し 同

	A	B	C	D	E	F	G
	0.20	0.16	0.24	0.06	0.11	0.19	0.04
A	0.59	0.40	<0.01	<0.01	<0.01	<0.01	<0.01
B	0.12	<0.01	0.51	0.36	<0.01	<0.01	<0.01
C	<0.01	0.32	0.44	<0.01	0.23	<0.01	<0.01
D	<0.01	<0.01	0.24	<0.01	<0.01	<0.01	0.76
E	<0.01	<0.01	<0.01	<0.01	0.50	0.49	<0.01
F	0.24	<0.01	<0.01	<0.01	<0.01	0.76	<0.01
G	<0.01	<0.01	1.00	<0.01	<0.01	<0.01	<0.01

b 次数 一次
歌開始部 同
繰り返し 別

	A	B	C	D	E	F	G
	0.14	0.27	0.23	0.10	0.10	0.09	0.07
A		0.99	<0.01	0.01	<0.01	<0.01	<0.01
B	0.12		0.51	0.36	<0.01	<0.01	<0.01
C	0.01	0.58		<0.01	0.42	<0.01	<0.01
D	<0.01	<0.01	0.24		<0.01	<0.01	0.76
E	0.01	<0.01	<0.01	<0.01		0.99	<0.01
F	1.00	<0.01	<0.01	<0.01	<0.01		<0.01
G	<0.01	<0.01	1.00	<0.01	<0.01	<0.01	

c 次数 一次
歌開始部 別
繰り返し 同

	A	B	C	D	E	F	G
	0.20	0.16	0.24	0.06	0.11	0.19	0.04
A	0.59	0.40	<0.01	<0.01	<0.01	<0.01	<0.01
B	0.12	<0.01	0.51	0.36	<0.01	<0.01	<0.01
C	<0.01	0.32	0.44	<0.01	0.23	<0.01	<0.01
D	<0.01	<0.01	0.24	<0.01	<0.01	<0.01	0.76
E	<0.01	<0.01	<0.01	<0.01	0.50	0.49	<0.01
F	0.24	<0.01	<0.01	<0.01	<0.01	0.76	<0.01
G	<0.01	<0.01	1.00	<0.01	<0.01	<0.01	<0.01
X	1.00	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
XA	1.00	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01

d 次数 一次
歌開始部 別
繰り返し 別

	A	B	C	D	E	F	G
	0.14	0.27	0.23	0.10	0.10	0.09	0.07
A		0.99	<0.01	0.01	<0.01	<0.01	<0.01
B	0.12		0.51	0.36	<0.01	<0.01	<0.01
C	0.01	0.58		<0.01	0.42	<0.01	<0.01
D	<0.01	<0.01	0.24		<0.01	<0.01	0.76
E	0.01	<0.01	<0.01	<0.01		0.99	<0.01
F	1.00	<0.01	<0.01	<0.01	<0.01		<0.01
G	<0.01	<0.01	1.00	<0.01	<0.01	<0.01	
X	1.00	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
XA		1.00	<0.01	<0.01	<0.01	<0.01	<0.01

e 次数　可変
歌開始部　同
繰り返し　同

	A	B	C	D	E	F	G
	0.20	0.16	0.24	0.06	0.11	0.19	0.04
A	0.59	0.40	<0.01	<0.01	<0.01	<0.01	<0.01
B	0.12	<0.01	0.51	0.36	<0.01	<0.01	<0.01
C	<0.01	0.32	0.44	<0.01	0.23	<0.01	<0.01
D	<0.01	<0.01	0.24	<0.01	<0.01	<0.01	0.76
E	<0.01	<0.01	<0.01	<0.01	0.50	0.49	<0.01
F	0.24	<0.01	<0.01	<0.01	<0.01	0.76	<0.01
G	<0.01	<0.01	1.00	<0.01	<0.01	<0.01	<0.01
AA	0.47	0.53	<0.01	<0.01	<0.01	<0.01	<0.01
AB	<0.01	<0.01	1.00	<0.01	<0.01	<0.01	<0.01
BA	0.06	0.94	<0.01	0.01	<0.01	<0.01	<0.01
BC	<0.01	<0.01	1.00	<0.01	<0.01	<0.01	<0.01
BD	<0.01	<0.01	0.23	<0.01	<0.01	<0.01	0.77
CB	0.25	<0.01	<0.01	0.75	<0.01	<0.01	<0.01
CC	0.01	0.73	0.27	<0.01	<0.01	<0.01	<0.01
CE	<0.01	<0.01	<0.01	<0.01	1.00	<0.01	<0.01
DC	<0.01	<0.01	<0.01	<0.01	1.00	<0.01	<0.01
EE	<0.01	<0.01	<0.01	<0.01	<0.01	1.00	<0.01
FA	1.00	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
FF	0.33	<0.01	<0.01	<0.01	<0.01	0.67	<0.01
GC	<0.01	<0.01	<0.01	<0.01	1.00	<0.01	<0.01
BCC	<0.01	0.64	0.36	<0.01	<0.01	<0.01	<0.01
CBD	<0.01	<0.01	0.23	<0.01	<0.01	<0.01	0.77
CCB	0.25	<0.01	<0.01	0.75	<0.01	<0.01	<0.01
CCC	0.02	0.98	<0.01	<0.01	<0.01	<0.01	<0.01
EFF	0.05	<0.01	<0.01	<0.01	<0.01	0.95	<0.01
FAA	0.06	0.94	<0.01	<0.01	<0.01	<0.01	<0.01
FFF	0.51	<0.01	<0.01	<0.01	<0.01	0.49	<0.01
ABCC	<0.01	0.64	0.36	<0.01	<0.01	<0.01	<0.01
BCCB	0.01	<0.01	<0.01	0.99	<0.01	<0.01	<0.01
CCBD	<0.01	<0.01	0.23	<0.01	<0.01	<0.01	0.77
CCCB	0.74	0.01	<0.01	0.26	<0.01	<0.01	<0.01
EFFF	0.51	<0.01	<0.01	<0.01	<0.01	0.49	<0.01
FFAA	0.06	0.94	<0.01	<0.01	<0.01	<0.01	<0.01
FFFF	0.51	<0.01	<0.01	<0.01	<0.01	0.49	<0.01
AABCC	<0.01	0.71	0.29	<0.01	<0.01	<0.01	<0.01
ABCCB	0.01	<0.01	<0.01	0.99	<0.01	<0.01	<0.01
BABCC	<0.01	0.40	0.60	<0.01	<0.01	<0.01	<0.01
BCCBD	<0.01	<0.01	0.13	<0.01	<0.01	<0.01	0.86
BCCCB	0.74	0.01	<0.01	0.26	<0.01	<0.01	<0.01
CCCBD	0.01	<0.01	0.96	<0.01	<0.01	<0.01	0.03
EEFFF	0.51	<0.01	<0.01	<0.01	<0.01	0.49	<0.01
EFFFF	0.48	<0.01	<0.01	<0.01	<0.01	0.52	<0.01
FFFAA	0.06	0.94	<0.01	<0.01	<0.01	<0.01	<0.01
AAABCC	<0.01	0.16	0.84	<0.01	<0.01	<0.01	<0.01
ABCCBD	<0.01	<0.01	0.13	<0.01	<0.01	<0.01	0.86
ABCCCB	0.74	0.01	<0.01	0.26	<0.01	<0.01	<0.01
BABCCB	0.06	<0.01	<0.01	0.94	<0.01	<0.01	<0.01
CBABCC	<0.01	0.39	0.60	<0.01	<0.01	<0.01	<0.01
CEEFFF	0.51	<0.01	<0.01	<0.01	<0.01	0.49	<0.01
EEFFFF	0.48	<0.01	<0.01	<0.01	<0.01	0.52	<0.01
FFFFAA	0.04	0.96	<0.01	<0.01	<0.01	<0.01	<0.01
AABCCBD	<0.01	<0.01	0.02	<0.01	<0.01	<0.01	0.98
AABCCCB	0.88	0.01	<0.01	0.11	<0.01	<0.01	<0.01
BABCCBD	<0.01	<0.01	0.85	<0.01	<0.01	<0.01	0.14
BABCCCB	0.52	<0.01	<0.01	0.48	<0.01	<0.01	<0.01
CBABCCB	0.06	<0.01	<0.01	0.94	<0.01	<0.01	<0.01
CCBABCC	<0.01	0.39	0.60	<0.01	<0.01	<0.01	<0.01
CEEFFFF	0.48	<0.01	<0.01	<0.01	<0.01	0.52	<0.01
DCEEFFF	0.09	<0.01	<0.01	<0.01	<0.01	0.91	<0.01
EFFFFAA	0.07	0.93	<0.01	<0.01	<0.01	<0.01	<0.01
GCEEFFF	0.69	<0.01	<0.01	<0.01	<0.01	0.31	<0.01
AAABCCBD	<0.01	<0.01	0.09	<0.01	<0.01	<0.01	0.91
CCBABCCB	0.06	<0.01	<0.01	0.94	<0.01	<0.01	<0.01
CCCBABCC	<0.01	0.39	0.60	<0.01	<0.01	<0.01	<0.01
DCEEFFFF	0.22	<0.01	<0.01	<0.01	<0.01	0.78	<0.01
GCEEFFFF	0.85	<0.01	<0.01	<0.01	<0.01	0.15	<0.01
BCCCBABCC	<0.01	0.40	0.60	<0.01	<0.01	<0.01	<0.01
CCCBABCCB	0.05	<0.01	<0.01	0.95	<0.01	<0.01	<0.01
ABCCCBABCC	<0.01	0.39	0.60	<0.01	<0.01	<0.01	<0.01
BCCCBABCCB	0.05	<0.01	<0.01	0.95	<0.01	<0.01	<0.01
ABCCCBABCCB	0.06	<0.01	<0.01	0.94	<0.01	<0.01	<0.01
BABCCCBABCC	<0.01	0.73	0.27	<0.01	<0.01	<0.01	<0.01
AABCCCBABCCB	0.09	<0.01	<0.01	0.91	<0.01	<0.01	<0.01

f 次数　可変
歌開始部　同
繰り返し　別

	A	B	C	D	E	F	G
	0.14	0.27	0.23	0.10	0.10	0.09	0.07
A		0.99	<0.01	0.01	<0.01	<0.01	<0.01
B	0.12		0.51	0.36	<0.01	<0.01	<0.01
C	0.01	0.58		<0.01	0.42	<0.01	<0.01
D	<0.01	<0.01	0.24		<0.01	<0.01	0.76
E	0.01	<0.01	<0.01	<0.01		0.99	<0.01
F	1.00	<0.01	<0.01	<0.01	<0.01		<0.01
G	<0.01	<0.01	1.00	<0.01	<0.01	<0.01	
AB	<0.01		1.00	<0.01	<0.01	<0.01	<0.01
BC	0.01	0.99		<0.01	<0.01	<0.01	<0.01
BD	<0.01	<0.01	0.23		<0.01	<0.01	0.77
CB	0.25		<0.01	0.75	<0.01	<0.01	<0.01
DC	<0.01	<0.01		<0.01	1.00	<0.01	<0.01
GC	<0.01	<0.01		<0.01	1.00	<0.01	<0.01
BCB	0.25		<0.01	0.74	<0.01	<0.01	<0.01
CBD	<0.01	<0.01	0.23		<0.01	<0.01	0.77
ABCB	0.25		<0.01	0.75	<0.01	<0.01	<0.01
BCBD	<0.01	<0.01	0.23		<0.01	<0.01	0.77
ABCBD	<0.01	<0.01	0.23		<0.01	<0.01	0.77
BABCB	0.33		<0.01	0.67	<0.01	<0.01	<0.01
BABCBD	0.01	<0.01	0.89		<0.01	<0.01	0.10
CBABCB	0.33		<0.01	0.66	<0.01	<0.01	<0.01
BCBABCB	0.33		<0.01	0.67	<0.01	<0.01	<0.01
CBABCBD	0.01	<0.01	0.90		<0.01	<0.01	0.10
ABCBABCB	0.33		<0.01	0.66	<0.01	<0.01	<0.01
BCBABCBD	0.01	<0.01	0.90		<0.01	<0.01	0.10
ABCBABCBD	0.01	<0.01	0.90		<0.01	<0.01	0.10
BABCBABCB	0.11		<0.01	0.89	<0.01	<0.01	<0.01
BABCBABCBD	<0.01	<0.01	0.84		<0.01	<0.01	0.16

g 次数　可変
歌開始部　別
繰り返し　同

	A	B	C	D	E	F	G
	0.20	0.16	0.24	0.06	0.11	0.19	0.04
A	0.59	0.40	<0.01	<0.01	<0.01	<0.01	<0.01
B	0.12	<0.01	0.51	0.36	<0.01	<0.01	<0.01
C	<0.01	0.32	0.44	<0.01	0.23	<0.01	<0.01
D	<0.01	<0.01	0.24	<0.01	<0.01	<0.01	0.76
E	<0.01	<0.01	<0.01	<0.01	0.50	0.49	<0.01
F	0.24	<0.01	<0.01	<0.01	<0.01	0.76	<0.01
G	<0.01	<0.01	1.00	<0.01	<0.01	<0.01	<0.01
X	1.00	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
AA	0.47	0.53	<0.01	<0.01	<0.01	<0.01	<0.01
AB	<0.01	<0.01	1.00	<0.01	<0.01	<0.01	<0.01
BA	0.06	0.94	<0.01	0.01	<0.01	<0.01	<0.01
BC	<0.01	<0.01	1.00	<0.01	<0.01	<0.01	<0.01
BD	<0.01	<0.01	0.23	<0.01	<0.01	<0.01	0.77
CB	0.25	<0.01	<0.01	0.75	<0.01	<0.01	<0.01
CC	0.01	0.73	0.27	<0.01	<0.01	<0.01	<0.01
CE	<0.01	<0.01	<0.01	<0.01	1.00	<0.01	<0.01
DC	<0.01	<0.01	<0.01	<0.01	1.00	<0.01	<0.01
EE	<0.01	<0.01	<0.01	<0.01	<0.01	1.00	<0.01
FA	1.00	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
FF	0.33	<0.01	<0.01	<0.01	<0.01	0.67	<0.01
GC	<0.01	<0.01	<0.01	<0.01	1.00	<0.01	<0.01
XA	1.00	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
BCC	<0.01	0.64	0.36	<0.01	<0.01	<0.01	<0.01
CBD	<0.01	<0.01	0.23	<0.01	<0.01	<0.01	0.77
CCB	0.25	<0.01	<0.01	0.75	<0.01	<0.01	<0.01
CCC	0.02	0.98	<0.01	<0.01	<0.01	<0.01	<0.01
EFF	0.05	<0.01	<0.01	<0.01	<0.01	0.95	<0.01
FAA	0.06	0.94	<0.01	<0.01	<0.01	<0.01	<0.01
FFF	0.51	<0.01	<0.01	<0.01	<0.01	0.49	<0.01
XAA	0.97	0.03	<0.01	<0.01	<0.01	<0.01	<0.01
ABCC	<0.01	0.64	0.36	<0.01	<0.01	<0.01	<0.01
BCCB	0.01	<0.01	<0.01	0.99	<0.01	<0.01	<0.01
CCBD	<0.01	<0.01	0.23	<0.01	<0.01	<0.01	0.77
CCCB	0.74	0.01	<0.01	0.26	<0.01	<0.01	<0.01
EFFF	0.51	<0.01	<0.01	<0.01	<0.01	0.49	<0.01
FFAA	0.06	0.94	<0.01	<0.01	<0.01	<0.01	<0.01
FFFF	0.51	<0.01	<0.01	<0.01	<0.01	0.49	<0.01
AABCC	<0.01	0.71	0.29	<0.01	<0.01	<0.01	<0.01
ABCCB	0.01	<0.01	<0.01	0.99	<0.01	<0.01	<0.01
BABCC	<0.01	0.40	0.60	<0.01	<0.01	<0.01	<0.01
BCCBD	<0.01	<0.01	0.13	<0.01	<0.01	<0.01	0.86
BCCCB	0.74	0.01	<0.01	0.26	<0.01	<0.01	<0.01
CCCBD	0.01	<0.01	0.96	<0.01	<0.01	<0.01	0.03
EEFFF	0.51	<0.01	<0.01	<0.01	<0.01	0.49	<0.01
EFFFF	0.48	<0.01	<0.01	<0.01	<0.01	0.52	<0.01
FFFAA	0.06	0.94	<0.01	<0.01	<0.01	<0.01	<0.01
AAABCC	<0.01	0.16	0.84	<0.01	<0.01	<0.01	<0.01
ABCCBD	<0.01	<0.01	0.13	<0.01	<0.01	<0.01	0.86
ABCCCB	0.74	0.01	<0.01	0.26	<0.01	<0.01	<0.01
BABCCB	0.06	<0.01	<0.01	0.94	<0.01	<0.01	<0.01
CBABCC	<0.01	0.39	0.60	<0.01	<0.01	<0.01	<0.01
CEEFFF	0.51	<0.01	<0.01	<0.01	<0.01	0.49	<0.01
EEFFFF	0.48	<0.01	<0.01	<0.01	<0.01	0.52	<0.01
FFFFAA	0.04	0.96	<0.01	<0.01	<0.01	<0.01	<0.01
AABCCBD	<0.01	<0.01	0.02	<0.01	<0.01	<0.01	0.98
AABCCCB	0.88	0.01	<0.01	0.11	<0.01	<0.01	<0.01
BABCCBD	<0.01	<0.01	0.85	<0.01	<0.01	<0.01	0.14
BABCCCB	0.52	<0.01	<0.01	0.48	<0.01	<0.01	<0.01
CBABCCB	0.06	<0.01	<0.01	0.94	<0.01	<0.01	<0.01
CCBABCC	<0.01	0.39	0.60	<0.01	<0.01	<0.01	<0.01
CEEFFFF	0.48	<0.01	<0.01	<0.01	<0.01	0.52	<0.01
DCEEFFF	0.09	<0.01	<0.01	<0.01	<0.01	0.91	<0.01
EFFFFAA	0.07	0.93	<0.01	<0.01	<0.01	<0.01	<0.01
GCEEFFF	0.69	<0.01	<0.01	<0.01	<0.01	0.31	<0.01
AAABCCBD	<0.01	<0.01	0.09	<0.01	<0.01	<0.01	0.91
CCBABCCB	0.06	<0.01	<0.01	0.94	<0.01	<0.01	<0.01
CCCBABCC	<0.01	0.39	0.60	<0.01	<0.01	<0.01	<0.01
DCEEFFFF	0.22	<0.01	<0.01	<0.01	<0.01	0.78	<0.01
GCEEFFFF	0.85	<0.01	<0.01	<0.01	<0.01	0.15	<0.01
XAABCCBD	<0.01	<0.01	0.17	<0.01	<0.01	<0.01	0.83
BCCCBABCC	<0.01	0.40	0.60	<0.01	<0.01	<0.01	<0.01
CCCBABCCB	0.05	<0.01	<0.01	0.95	<0.01	<0.01	<0.01
XAAABCCBD	<0.01	<0.01	0.61	<0.01	<0.01	<0.01	0.39
ABCCCBABCC	<0.01	0.39	0.60	<0.01	<0.01	<0.01	<0.01
BCCCBABCCB	0.05	<0.01	<0.01	0.95	<0.01	<0.01	<0.01
ABCCCBABCCB	0.06	<0.01	<0.01	0.94	<0.01	<0.01	<0.01
BABCCCBABCC	<0.01	0.73	0.27	<0.01	<0.01	<0.01	<0.01
AABCCCBABCCB	0.09	<0.01	<0.01	0.91	<0.01	<0.01	<0.01

h 次数　可変
歌開始部　別
繰り返し　別

	A	B	C	D	E	F	G
	0.14	0.27	0.23	0.10	0.10	0.09	0.07
A		0.99	<0.01	0.01	<0.01	<0.01	<0.01
B	0.12		0.51	0.36	<0.01	<0.01	<0.01
C	0.01	0.58		<0.01	0.42	<0.01	<0.01
D	<0.01	<0.01	0.24		<0.01	<0.01	0.76
E	0.01	<0.01	<0.01	<0.01		0.99	<0.01
F	1.00	<0.01	<0.01	<0.01	<0.01		<0.01
G	<0.01	<0.01	1.00	<0.01	<0.01	<0.01	
X	1.00	<0.01	<0.01	<0.01	<0.01	<0.01	<0.01
AB	<0.01		1.00	<0.01	<0.01	<0.01	<0.01
BC	0.01	0.99		<0.01	<0.01	<0.01	<0.01
BD	<0.01	<0.01	0.23		<0.01	<0.01	0.77
CB	0.25		<0.01	0.75	<0.01	<0.01	<0.01
DC	<0.01	<0.01		<0.01	1.00	<0.01	<0.01
GC	<0.01	<0.01		<0.01	1.00	<0.01	<0.01
BCB	0.25		<0.01	0.74	<0.01	<0.01	<0.01
CBD	<0.01	<0.01	0.23		<0.01	<0.01	0.77
ABCB	0.25		<0.01	0.75	<0.01	<0.01	<0.01
BCBD	<0.01	<0.01	0.23		<0.01	<0.01	0.77
ABCBD	<0.01	<0.01	0.23		<0.01	<0.01	0.77
BABCB	0.33		<0.01	0.67	<0.01	<0.01	<0.01
XABCB	0.86		0.01	0.12	<0.01	<0.01	<0.01
BABCBD	0.01	<0.01	0.89		<0.01	<0.01	0.10
CBABCB	0.33		<0.01	0.66	<0.01	<0.01	<0.01
XABCBD	0.01	<0.01	0.79		<0.01	<0.01	0.21
BCBABCB	0.33		<0.01	0.67	<0.01	<0.01	<0.01
CBABCBD	0.01	<0.01	0.90		<0.01	<0.01	0.10
ABCBABCB	0.33		<0.01	0.66	<0.01	<0.01	<0.01
BCBABCBD	0.01	<0.01	0.90		<0.01	<0.01	0.10
ABCBABCBD	0.01	<0.01	0.90		<0.01	<0.01	0.10
BABCBABCB	0.11		<0.01	0.89	<0.01	<0.01	<0.01
BABCBABCBD	<0.01	<0.01	0.84		<0.01	<0.01	0.16
XABCBABCBD	0.01	<0.01	0.94		<0.01	<0.01	0.05

第 4 章　実験による検証

導入

第 3 章では、ジュウシマツの歌文法が直前の系列の履歴を用いたニューラルネットワークによって表現できることを示した。この結果から、同様の機能が生体内で実装されているかもしれないと予測できる。生体内では歌の履歴は聴覚フィードバック・体性感覚フィードバック・脳内フィードバックによって表現されていると考えられる。つまり、これらのフィードバック情報がジュウシマツの脳神経系において表現されているかもしれない。本研究では、この仮説を実験的に検証した。

鳴禽の脳には、歌の生成･知覚･学習･維持に特化した神経核が存在する(Bolhuis et al., 2010; Brainard & Doupe, 2002; Nottebohm, Stokes, & Leonard, 1976)。これらの神経核は歌神経核と呼ばれている。本研究では、歌神経核の中で、歌のフィードバックが表現されている可能性が高い神経核として HVC（正式名称）と Area X（正式名称）に着目した（図 4.1）。HVC は哺乳類の運動前野、Area X は基底核に対応する(Bolhuis et al., 2010)。Area X は Area X・視床背外側部の内側核（dorsal lateral nucleus of the medial thalamus、DLM）･前巣外套の外側巨大細胞神経核（lateral magnocellular nucleus of the anterior nidopallium、LMAN）からなる前脳経路と呼ばれる神経回路の一部である。前脳経路は歌の学習・維持に重要である(Kao, Doupe, & Brainard, 2005; Sakata & Brainard, 2008)。HVC は一次聴覚野と高次聴覚野から投射を受け(Kelley & Nottebohm, 1979; Lewicki & Arthur, 1996)、Area X に投射を送る(Nottebohm et al., 1976)。Area X は HVC と LMAN から投射を受ける。よって前脳経路内で、Area X、DLM、LMAN、Area X とループを作っている。また、HVC と LMAN は robust nucleus of the arcopallium

（RA）という哺乳類の運動野に対応する神経核へ投射する。よって聴覚情報は一次聴覚野と高次聴覚野から HVC に送られ、HVC で運動指令と統合され、RA と Area X へ同時に送られると考えられる。HVC からの情報は RA ではより細かい運動指令の出力に用いられ、Area X では歌の維持のために用いられる。よって HVC と Area X は、自身の歌の聴覚フィードバックだけでなく、運動指令の内的フィードバックも処理していると考えられている。

本研究では、まず HVC と Area X が自身の歌の聴覚刺激に対してどのような反応を示すかを調べた。HVC と Area X の神経細胞は、自身の歌（bird’s own song、BOS）に対して、BOS を反転した聴覚刺激よりも大きな平均発火頻度を示す(Doupe, 1997; Nakamura & Okanoya, 2004; Theunissen & Doupe, 1998)。他の聴覚刺激とくらべて BOS に対して大きな平均発火頻度を示す細胞を、BOS 選択的な神経細胞と呼ぶ。BOS 選択的な細胞は歌の生成・維持に重要であると考えられている(Doupe, 1997)。ジュウシマツの歌は個体ごとに異なるので、HVC と Area X の神経活動を個体内で比較することが重要である。また、平均発火頻度だけでなく、発火時刻のばらつきについても検討することが重要である。よって本研究では、HVC と Area X の BOS 選択的な神経細胞の平均発火頻度と発火時刻のばらつきを個体内で比較した。神経活動が BOS のどのような側面に依存するのかを調べるために、BOS・要素を反転した聴覚刺激・要素の順序を逆にした聴覚刺激・全体を反転した聴覚刺激、の 4 種類の聴覚刺激を用いた。

次に、第 3 章で作成したモデルから導かれる仮説を検証するために、BOS 刺激中の要素系列を予測するためのモデルへのフィードバック入力を計算し、記録した神経活動と比較した。生体内でモ

デルと同様の情報表現がされていれば、神経活動に系列のフィードバックの情報が含まれていることが期待できる。

方法

被験体

オスの成体ジュウシマツ 10 羽を用いた。全個体が孵化後 120 日齢以上であった。実験に用いるまでは、餌・水を自由に摂ることができる環境で飼育していた。明暗のサイクルは 14 時間・10 時間とした。

聴覚刺激

ジュウシマツを単独で防音箱に入れ、歌を記録した。記録にはマイク（ECM-MS907、Sony Corporation、日本）、オーディオインターフェイス（SE-U77、Onkyo Corporation、日本）、ソフトウェア（SASLab Pro、Avisoft Bioacoustics、ドイツ）を用いた。22.05 kHz のサンプリング周波数・16 bit の深度で録音した。個体ごとに、記録した音声から 3 s より短い発声を選び、聴覚刺激とした。歌要素を音声スペクトログラムの目視により手動で検出し分類した。音声スペクトログラムは長さ 256 の短時間離散フーリエ変換によって計算した。歌要素を、前後 3 ms にフェードイン・アウトを付けて記録した音声から切り出した。切り出した歌要素を元に、次の 4 種類の聴覚刺激を作成した。

BOS 選択的な神経活動が BOS のどのような構造に依存するのかを調べるために、記録した歌から歌要素を抽出し、それを再配置することにより BOS 刺激・局所反転刺激・逆順刺激・全反転刺激の

4 種類の聴覚刺激を作成した（図 4.2）。BOS 刺激では、歌要素を元の歌と同じ間隔で配置した。局所反転刺激では、時間的に反転した歌要素を元の歌と同じ間隔で配置した。逆順刺激では、歌要素を元の歌と逆順に配置した。全反転刺激では、時間的に反転した歌要素を元の歌と逆順に配置した。つまり、BOS 刺激では音声の局所的な変化も大局的な変化も元の歌と同じ、局所反転刺激では局所的な変化が元の歌と異なり大局的な変化は同じとした。また、逆順刺激では局所的な変化は同じで大局的な変化が異なり、全反転刺激ではどちらの変化も元の歌と異なっていた。

手術

被験体を 0.2 ml の 10%カルバミン酸エチルで麻酔し、定位装置（Model 900、David Kopf Instruments、アメリカ合衆国）に固定した。耳と嘴の先端が水平面から 45° 下になるように頭の角度を調整した。局所麻酔薬（Xylocaine、Astrazeneca K.K、日本）を頭皮に塗り、HVC と Area X の上にある頭皮、頭蓋、硬膜を取り除いた。ジュウシマツの HVC は脳表の中心にある分岐点から 2.0 mm 外、0.5 mm 前に、Area X は 1.7 mm 外、4.7 mm 前にある。

神経活動の記録

神経活動を記録するために、被験体を定位装置に固定し、電磁波を遮断した防音箱に入れた。神経活動は HVC と Area X から順に記録した。ジュウシマツの歌神経核は左優位であるので(Okanoya, Ikebuchi, Uno, & Watanabe, 2001)、神経活動は左半球から記録した。記録には、インピーダンス 2 MΩ のタングステン電極（#573200、A-M Systems Inc.、アメリカ合衆国）、増幅器（DAM80、World Precision Instruments アメリカ合衆国）、バンドパスフィルタ（Multichannel SR Filter 3315、NF Corporation、日

本)、アナログデジタル変換器（MICRO1401、Cambridge Electronic Design Ltd、イギリス)、ソフトウェア（Spike2、Cambridge Electronic Design Ltd）を用いた。バンドパスフィルタの通過帯域は 100 Hz から 25 kHz とした。記録のサンプリング周波数は 25 kHz とした。記録位置ごとに、4 種類の聴覚刺激をランダムな順序で 30 回ずつ再生した。聴覚刺激の提示の間隔は[3 s, 5 s]の範囲で毎回ランダムに選んだ。聴覚刺激の提示にはソフトウェア（Spike2），デジタルアナログ変換器（MICRO1401)、ローパスフィルタ（900C9L8B、Frequency Devices Inc.、アメリカ合衆国)、増幅器つきのスピーカー（SRS-Z1PC、Sony　Corporation、日本）を用いた。ローパスフィルタの遮断周波数は 10 kHz とした。聴覚刺激の音圧レベルは被験体の耳の位置で約 70 dB となるようにした。

電極位置の確認

神経活動の記録後、20 μA の電流を 20 s 間流すことにより、脳の電気的微小破壊を行った。微小破壊には電機刺激装置（SEN-3301、Nihon Kohden Corporation、日本）を用いた。次に、ペントバルビタールを過剰投与し、4%のパラホルムアルデヒドにより灌流固定した。脳を 50 μm のスライスに切り、cresyl violet により染色した。染色したスライスを顕微鏡下で観察し、電極の位置を確認した。

平均発火頻度についての解析

マルチユニット発火を、極小値が閾値を下回る細胞外電位と定義した。閾値は記録した電位変動の平均値±3×標準偏差とした。聴覚刺激の開始から終了の 0.1 s 後までの時間における平均発火頻度を計算した。刺激終了から 0.1 s までとしたのは、刺激終了後も神経活動が続くことが多くあったからである(Doupe, 1997)。自発発火頻度を刺激提示の試行ごとに計算した。自発発火頻度は、刺激

開始直前までと刺激終了後 1.6 s 後からのそれぞれ刺激の半分の時間における平均発火頻度の平均値とした。刺激終了後 1.6 s 後からとしたのは、刺激終了後 1.6 s では刺激による神経活動が完全に消えていたことを目視により確認できたからである。刺激提示中の発火頻度から自発発火頻度を引いたものを、刺激提示中の平均発火頻度の上昇とした。2 種類の刺激に対する平均発火頻度の上昇の差を、d'によって計算した(Green & Swets, 1966)。

$$d'(\text{刺激 A/刺激 B}) = \frac{2(\mu_A - \mu_B)}{\sqrt{\sigma_A^2 + \sigma_B^2}}$$

ただし、μ_Xとσ_X^2はそれぞれ刺激 X に対する平均発火頻度の上昇の平均値と分散である。

HVC と Area X において BOS 刺激提示中の平均発火頻度が全反転刺激提示中よりも大きいことがわかっている(Doupe, 1997; Nakamura & Okanoya, 2004)。本研究では、BOS 刺激提示中の平均発火頻度が自発発火頻度よりも大きく（t 検定により $p < 0.05$）、全反転刺激提示中よりも大きい（d'(BOS/全反転) > 0.5）神経活動を解析に用いた(Theunissen & Doupe, 1998)。全個体の HVC と Area X で、この基準を満たす記録点が 1 箇所以上あったので、刺激の種類と神経核についてd'を個体内で比較した。基準を満たす記録点が 1 神経核に複数あった場合、神経核内でd'の値を平均した。d'を個体内 2 要因分散分析により比較した。分散分析では、刺激の種類（BOS 刺激との比較で局所反転刺激・逆順刺激・全反転刺激）と神経核（HVC と Area X）について比較した。要因の主効果が有意であったため、全組み合わせについて対応のある総当り t 検定を行った。必要に応じて Bonferroni 補正を行った。本論文では、可読性を上げるために補正後の p 値を記述したので、p 値が 1 を超える場合があるが、その場合は 1 とした。

発火時刻についての解析

同じ種類の刺激を複数回提示したときの発火時刻の安定性を、瞬時発火頻度の試行間の相関係数によって調べた。まず、瞬時発火頻度 $r(t)$を、発火時刻の列をガウシアンカーネルにより平滑化したもので推定した。ガウシアンカーネルの半値全幅は 20 ms とした。次に、同じ種類の刺激を提示したときの瞬時発火頻度の試行間の相関係数CC_{ij}を計算した(Goldberg, Adler, Bergman, & Fee, 2010)。

$$CC_{ij} = \frac{< r_i(t) r_j(t) >_t}{< r_i(t)^2 >_t < r_j(t)^2 >_t}$$

ただし、iとjは試行を、$<\cdot>_t$はtについての平均値を表す。最後に、CC_{ij}を全試行ペアで平均し、平均相関係数CCとした。

$$CC = \frac{1}{N_{pairs}} \sum_{i<j} CC_{ij}$$

相関係数の値は平均発火頻度に依存する。発火時刻が完全にランダムな場合でも、平均発火頻度が大きいほど異なる試行において同じ時刻に発火しやすくなるからである。そこで、平均発火頻度に依存しない指標として、発火時刻がランダムであると仮定したときの相関係数の分布を元にz値を計算した(Olveczky, Andalman, & Fee, 2005)。まず、平均発火頻度は元のデータと同じで発火時刻がランダムであると仮定できるデータとして、元の発火時刻列の時刻をランダムにずらした発火時刻列を 1,000 パターン作成した。次に、それぞれのデータにおける瞬時発火頻度の相関係数を計算した。最後に、これらの相関係数の分布CC_{null}を元に、元のデータにおける相関係数のz値を計算した。

$$z = \frac{CC - CC_{null}\text{の平均値}}{CC_{null}\text{の標準偏差}}$$

同じ個体で 1 つの神経核から複数の記録を行った場合、各記録点におけるz値を平均した。平均発火頻度の解析と同様に、z値を個体内 2 要因分散分析により比較した。分散分析では、刺激の種類（BOS 刺激との比較で局所反転刺激・逆順刺激・全反転刺激）と神経核（HVC と Area X）について比較し

た。要因の主効果が有意であったため、全組み合わせについて対応のある総当り t 検定を行った。必要に応じて Bonferroni 補正を行った。本論文では、可読性を上げるために補正後の p 値を記述したので、書かれている p 値が 1 を超えることがある。

歌文法の解析

記録した音声から、連続した発声を音声スペクトログラムの目視により手動で抽出した。第 2 章に記述した方法により歌要素を検出・分類した。歌を 300 ms 以上の無音区間または地鳴きによって区切られた連続する 8 要素以上の音声とした。分類不能であった歌要素が全要素の 1%以上あった個体（2 羽）を解析から除き、残りの 8 個体について以降の解析を行った。歌を、1 系列に含まれる歌要素が 32 要素以下になるようにさらに分割した。16 分割交差検証により文法の予測精度を検証した。第 3 章と同様に訓練用データ内の交差検証により、フィードバックの最適な減衰時定数を決定した。

神経活動と文法モデルの比較

文法モデルの評価では、訓練用データごとに最適なフィードバックの減衰時定数を決定したので、個体ごとに複数の時定数が存在する。そこで、全データを用いた交差検証により、検証用データにおける尤度を最大にするような時定数を個体ごとに 1 つ決定した。そして、決定した時定数を用いて、BOS 刺激中のフィードバック値を計算した。

神経活動にフィードバックの情報が含まれているかどうかを検討するために、発火頻度を従属変数・フィードバック値を独立変数とした重回帰分析を行った。従属変数は、ある要素の開始点から

次の要素の開始点までの区間における要素ごとの平均発火頻度とした。独立変数には、従属変数と同じ区間内のフィードバック値を平均したものを用いた。さらに、BOS 刺激の直前の 100 ms 間の平均発火頻度とフィードバック値も用いた。このときのフィードバック値は聴覚刺激が始まっていないので全て 0 となる。フィードバック値の情報が発火頻度によってどの程度表現されているかの指標として、重回帰分析の決定係数を用いた。決定係数が 0.85 以上であった記録点について、結果の信頼性を確認するために、従属変数と独立変数をランダムに入れ替えたデータを用いた決定係数の分布と比較した。ランダムに入れ替えたデータは 10,000 サンプル用いた。

神経核間の差を検討するために、1 個体の複数の神経核から記録をとった場合は、神経核内で決定係数の値を平均した。神経核間の決定係数の差を Wilcoxon の符号順位検定を用いて比較した。

結果

聴覚刺激

記録した歌を改変し、BOS 刺激・局所反転刺激・逆順刺激・全反転刺激の 4 種類の聴覚刺激を作成した（図 4.2）。BOS 刺激と局所反転刺激の振幅包絡の相関係数は高く（平均値±標準偏差 = 0.63 ±0.16）、BOS 刺激と逆順刺激・全反転刺激との相関係数は低かった（BOS 刺激と逆順刺激では 0.022 ±0.14、全反転刺激では 0.002±0.19）（図 4.3）。よって、局所反転刺激では BOS 刺激の大局的な構造が保存されているが、逆順刺激と全反転刺激では保存されていないことが確認できた。

発火頻度

BOS 刺激へ選択性のあるマルチユニット神経活動を、10 羽のジュウシマツの HVC と Area X から記録した。脳切片を観察して電極の痕跡が HVC と Area X にあることを確認した（図 4.4）。聴覚刺激提示中の発火時刻と発火頻度から、HVC でも Area X でも、BOS 刺激提示中と逆順刺激提示中に発火頻度が鋭く上昇したことがわかる（図 4.5）。局所反転刺激提示中と全反転刺激提示中ではそのような反応は見られなかった。HVC のほうが Area X よりも上昇の程度が大きいように見えた。この結果は、HVC において Area X よりも発火時刻が揃っていたということを示唆する。

聴覚刺激が平均発火頻度に与える影響を調べるために、自発発火頻度と比較した刺激提示中の平均発火頻度の上昇を、刺激種類ごとに計算した。各刺激種類に対する神経活動の BOS 選択性を、平均発火頻度の上昇のd'を用いて測った（図 4.6a）。d'の値が大きいと、ある刺激種類と比べて BOS 刺激提示中に強く発火したことを意味する。HVC でも Area X でも、3 種類の刺激間でd'が有意に異なっていた（個体内 2 要因 ANOVA により、$F(2, 18) = 25.0$、$p < .001$）。続いて総当り t 検定を行ったところ、d'の大きさはd'(BOS/全反転)、d'(BOS/局所反転)、d'(BOS/逆順)の順であった（p 値を 3 倍に補正した対応あり t 検定により、d'(BOS/全反転)とd'(BOS/局所反転)の比較で $t(19) = 4.14$、$p = .0017$、d'(BOS/全反転)とd'(BOS/逆順)で $t(19) = 8.31$、$p < .001$、d'(BOS/局所反転)とd'(BOS/逆順)で $t(19) = 3.94$、$p = .0026$）。つまり、歌要素の反転は順序の反転よりも BOS 刺激への選択性に大きな影響を与えた。HVC と Area X における平均発火頻度の上昇の差や、交互作用は有意ではなかった（ANOVA により、神経核の主効果は $F(1, 9) = 1.10$、$p = .32$、交互作用は $F(2, 18) = 2.77$、$p = .090$）。本研究では BOS 選択性のある神経活動のみを対象とした。BOS 選択的な神経活動をd'(BOS/全反転)が 0.5 以上の神経活

動と定義したため、全神経活動においてd'(BOS/全反転)が大きかったのは自明である。

次に、聴覚刺激が発火時刻に与える影響を調べるために、同じ種類の刺激を提示したときの瞬時発火頻度の試行間の相関係数を計算した（図 4.6b）。瞬時発火頻度の相関係数は平均発火頻度に依存するので、平均発火頻度に依存しない指標として、発火時刻がランダムであると仮定した相関係数の分布から計算したz値を用いた。HVC でも Area X でも、相関係数のz値は刺激の種類によって有意に異なっていた（個体内 2 要因 ANOVA により、$F(3, 27) = 6.37$、$p = .0021$）。続いて総当り t 検定を行ったところ、相関係数のz値は BOS 刺激と逆順刺激提示中で、局所反転刺激と全反転刺激提示中よりも大きかった（p 値を 6 倍に補正した対応あり t 検定により、BOS と局所反転で $t(19) = 2.98$、$p = .046$、BOS と全反転で $t(19) = 3.07$、$p = .038$、逆順と局所反転で $t(19) = 3.58$、$p =.012$、逆順と全反転で $t(19) = 3.55$、$p =.013$）。他の刺激対では有意な差は見られなかった（BOS と逆順で $t(19) = 2.01$、$p =.31$、局所反転と全反転で $t(19) = 0.542$、$p > 1$）。つまり、歌要素が保たれた刺激に対する発火時刻が、歌要素が反転した刺激に対する発火時刻よりも試行をまたいで安定していた。さらに、HVC におけるz値のほうが Area X よりも大きかった（ANOVA により、$F(1, 9) = 9.94$、$p = .012$）。よって HVC における発火時刻のほうが Area X における発火時刻よりも安定していた。刺激の種類と神経核の交互作用は有意ではなかった（$F(3, 27) = 2.42$、$p = .088$）。

歌文法のモデル

歌文法と神経活動の関係を調べるために、歌文法を第 3 章と同様にフィードバックを入力とするニューラルネットワークによってモデル化した。交差検証による予測の尤度は、0.89±0.07 であった。

よって、このモデルにより歌文法を十分良く表現できたといえる。フィードバックの減衰時定数は、個体ごとに交差検証により決定した。最適な時定数は 170.9±87.9 であった。第 3 章の結果を異なる個体で再現できた。

歌文法と神経活動の関係

モデルから、直前の系列の情報が神経活動により表現されているかもしれないと予測できる。この仮説を検証するために、要素区間の平均発火頻度を従属変数、同区間内のモデルのフィードバック入力の平均値を独立変数とした重回帰分析を行った（図 4.7）。要素区間はある要素の開始から次の要素の開始までとした。最後の要素区間は、BOS の刺激中の最後の要素の開始から、終了の 100 ms 後までとした。さらに、BOS 刺激直前の 100 ms についても平均発火頻度とフィードバック入力を計算した。全個体の HVC と Area X の BOS 選択的な神経活動について重回帰分析を行い、重回帰分析の決定係数を計算した。結果として、8 羽中 4 羽で、決定係数が 0.85 を超える神経活動が存在した。さらに、決定係数が 0.85 を超えた神経活動について結果の信頼性を確認するために、従属変数と独立変数をランダムに入れ替えたデータを用いて、神経活動とフィードバック入力が対応しないと仮定したときの決定係数を計算した。元のデータによる全ての決定係数は、入れ替えたデータによる決定係数の上位 5%より大きかった。よって、決定係数が大きかった神経活動は、確かにフィードバック入力の情報を含んでいたといえる。

神経核間には決定係数の有意な差は無かった（Wilcoxon 符号順位検定により、W = 31、p = .36）。決定係数が 0.85 を超える神経活動が見つかった 4 個体の中では、2 個体で HVC と Area X の両方に

このような神経活動が見られ、1 個体で HVC のみ、1 個体で Area X のみに見られた。

考察

歌の構造と神経活動の関係

本研究では、まず、BOS 選択的な神経細胞が 4 種類の刺激提示中に示す反応について調べた。BOS 選択的な神経活動は、全反転刺激提示中よりも BOS 刺激提示中に大きな平均発火頻度示す神経活動であると定義した。全反転刺激は、BOS 刺激と局所的な構造と大局的な構造の両方が異なっている。平均発火頻度を刺激の種類間で比較したところ、平均発火率が歌の大局的な構造よりも局所的な構造に依存することがわかった。また、発火時刻の安定性を刺激の種類間で比較したところ、BOS の局所的な構造が保たれた聴覚刺激提示中に、発火時刻が試行をまたいで安定していた。

平均発火頻度と発火時刻の解析から、HVC と Area X の神経活動は歌の大局的な構造よりも局所的な構造に依存することがわかった。この結果は、次に出現する歌要素を予測するために必要なフィードバック強度の減衰時定数が歌要素約 2 個分であるという結果とも無矛盾である。ただし、d'(BOS/逆順)も 0 より大きかったので、歌要素の順序に対する選択性も弱いながらに存在した(Nakamura & Okanoya, 2004)。

神経核と神経活動の関係

発火時刻の安定性を異なる神経核で比較したところ、発火時刻が HVC において Area X よりも安定していた。鳴禽がうたっているときに、HVC において歌の構造に正確に同期した発火が見られる

(Hahnloser et al., 2002)。また、歌の維持のために、HVC は Area X に歌の時刻情報を送っており、Area X はその時刻情報とその他の入力を元に歌の修正についての計算をしていると考えられている(Fee & Goldberg, 2011)。HVC と Area X において発火時刻の安定性が異なっていたという結果は、HVC から Area X への情報伝達の際に発火時刻にノイズが加わったことが原因かもしれない。あるいは、Area X が HVC からの情報を時間的に積分しているからかもしれない。

モデルの検証

本研究の結果、HVC と Area X において、直前の系列のフィードバックの情報を含む神経活動が存在することがわかった。本研究ではフィードバックと神経活動の対応を重回帰分析によって調べた。重回帰分析では従属変数を独立変数のアフィン変換によって表現するため、ニューラルネットワークのような非線形なモデルと比べて制約が大きい。それにも関わらず神経活動を精度よく説明できたということは、モデルの入力値と神経活動が強く関係しているということを意味する。本研究によって、生体内においてフィードバックを用いて歌文法が表現されているというモデルによる予測を支持する結果が得られた。また、フィードバックの情報を含む神経活動が HVC と Area X の両方に見られたことから、両方の神経核が歌文法の表現に関わっていることが示唆された。

フィードバックの情報を含む神経活動が見つからなかった個体も存在した。これは、各個体で神経核内の全領域を網羅できなかったことが原因かもしれない。HVC と Area X の機能はフィードバックの表現だけではない。HVC には歌の細かい時間情報を表現するという機能があり、Area X には歌要素の音響構造の誤差情報を計算するという機能があると考えられている。よって、フィードバッ

クの情報を表現する神経細胞が記録点付近に存在しなかった可能性がある。

本研究の限界

本研究の結果は、麻酔下のジュウシマツから記録した神経活動によるものなので、HVC と Area X が発声時に同様の機能を持つことを証明することはできない。鳴禽の歌神経核では、覚醒時にフィードバックを用いて学習を行うが、麻酔下や睡眠状態で見られる神経活動は学習に用いられる情報と対応するという研究がある(Doupe & Konishi, 1991)。よって、麻酔下で確認できたフィードバックの情報が、発声時にも用いられている可能性は高いと考えられる。

歌文法のモデル推定の結果より、可変次数マルコフモデルでも、系列の履歴を入力とするニューラルネットワークによるモデルでも、複雑な歌文法をコンパクトに表現することが可能であった。系列の履歴と対応する神経活動が記録できたため、ジュウシマツが系列の履歴を入力とするニューラルネットワークによるモデルを用いて歌文法を表現している可能性を支持する結果となった。しかし、実際に歌文法がコンパクトに表現されていることを直接的に証明する結果は得られなかった。本研究では用いた聴覚刺激は 4 種類であったが､より多様な聴覚刺激を用いて同様の実験を行うと、歌文法をコンパクトに表現するような神経活動を記録できるかもしれない。歌文法をコンパクトに表現するような神経活動では、多様な聴覚刺激のうち、BOS に出現するものがそれぞれ異なって表現され、その他の聴覚刺激は区別されないかもしれない。

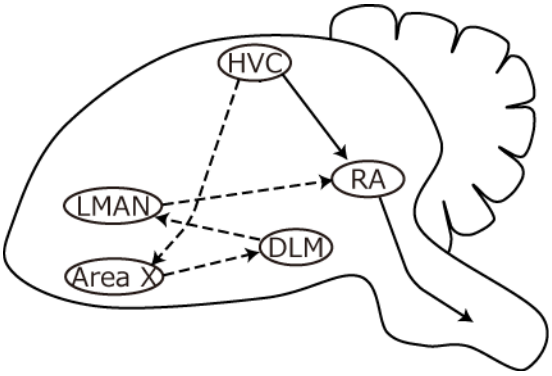


図 4.1｜鳴禽の歌神経核

歌の生成と維持に重要な神経核を、楕円により模式的に示した。神経核間の投射を矢印で示した。

前脳経路を破線で示した。図の左が吻側、上が背側を表す。

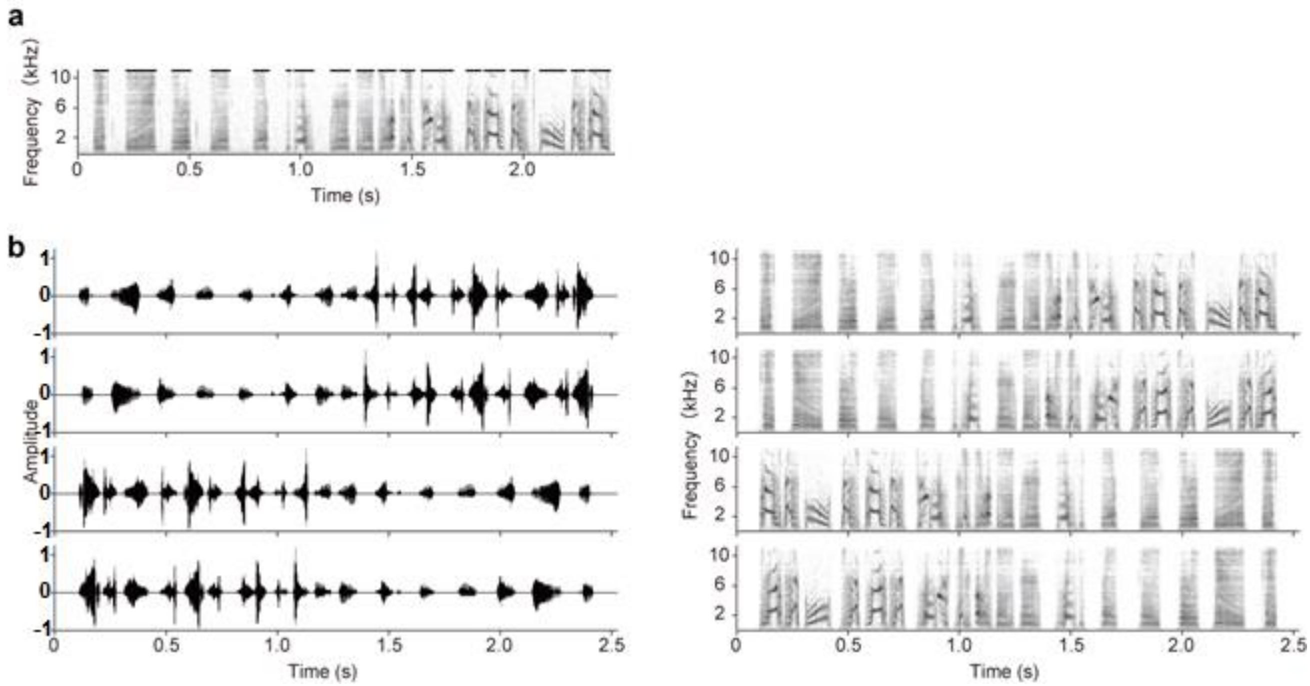


図 4.2｜4 種類の聴覚刺激

(a)ある個体の歌の一部。歌要素の区間を黒色の横棒で示した。(b)歌を元に作成した 4 種類の聴覚刺激。上から BOS 刺激、局所反転刺激、逆順刺激、全反転刺激を示した。左に音圧の変動を、右にスペクトログラムを示した。音圧の単位は任意単位である。

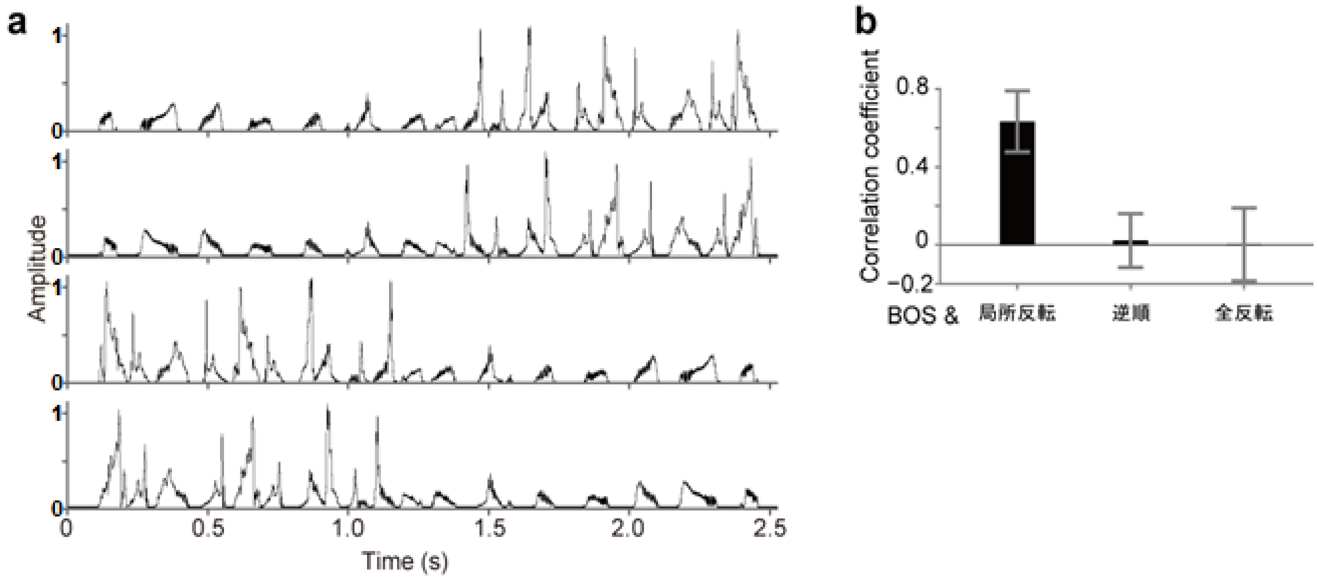


図 4.3｜聴覚刺激の振幅包絡

(a)4 種類の聴覚刺激の振幅包絡。上から BOS 刺激、局所反転刺激、逆順刺激、全反転刺激を示した。

(b)BOS 刺激とその他の刺激での、振幅包絡の相関係数。エラーバーは標準偏差を表す。音圧の単位は任意単位である。

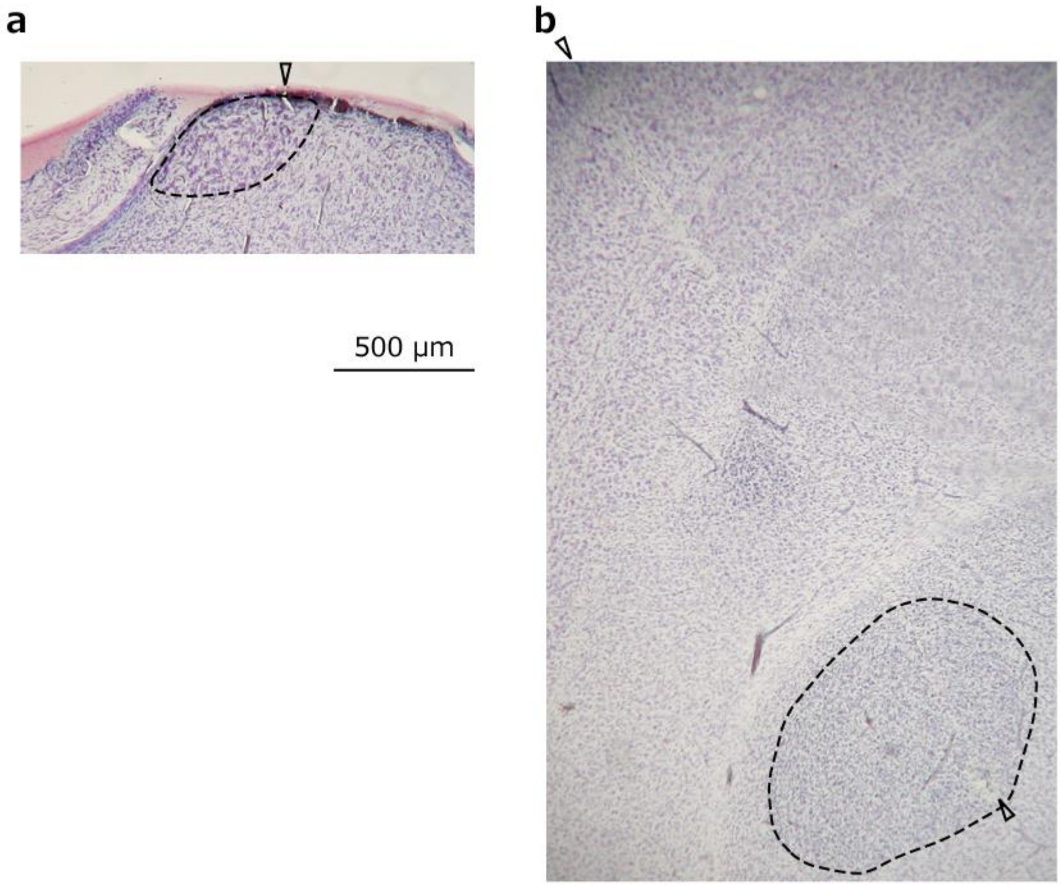


図 4.4｜脳切片

ある個体における染色した矢状断の脳切片。神経核を破線で囲み、電極の痕跡を白色の三角で示した。図の左が吻側、上が背側を表す。(a)HVC。 (b)Area X。

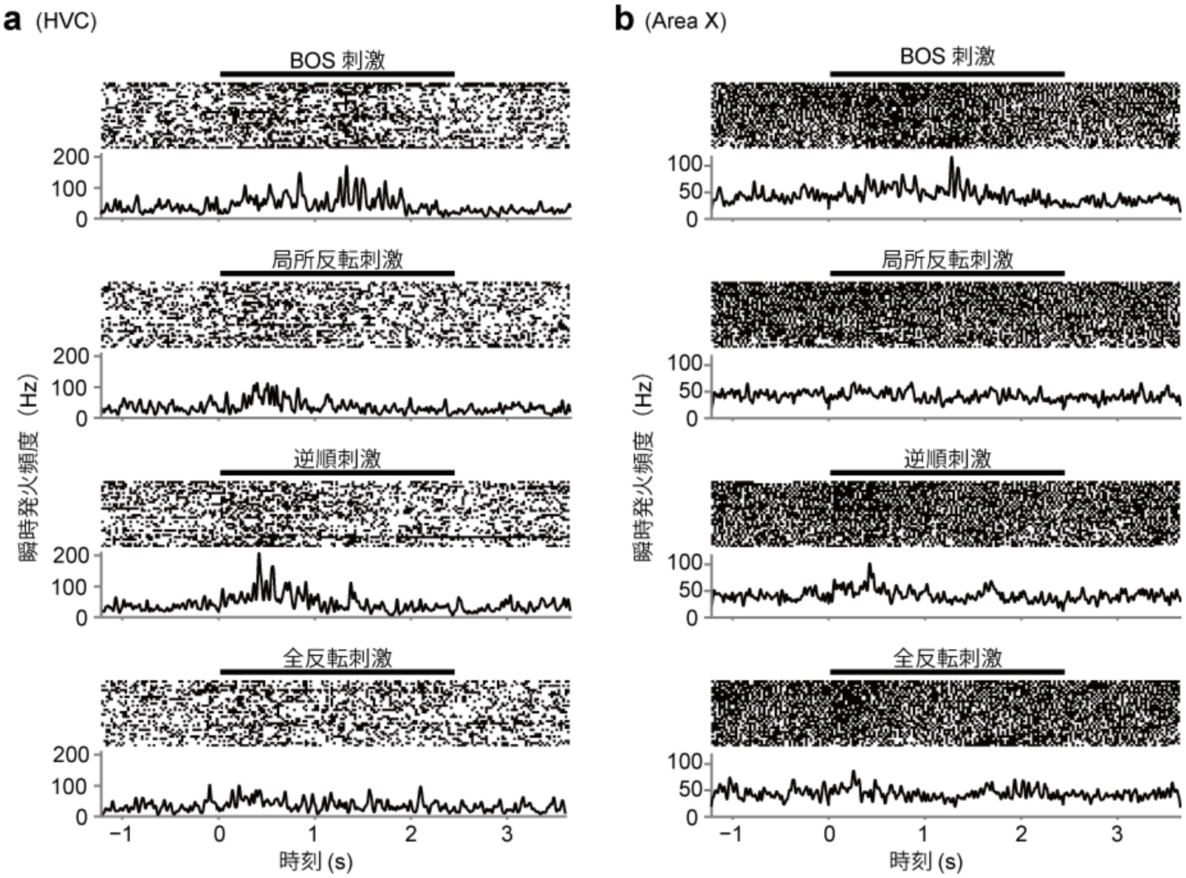


図 4.5｜発火時刻と発火頻度

ある個体における、それぞれの刺激提示中の発火時刻と発火頻度。上から BOS 刺激、局所反転刺激、逆順刺激、全反転刺激提示中の神経活動を示した。刺激の種類ごとに、上段に発火時刻のラスタープロット、下段に試行で平均した瞬時発火頻度を示した。刺激を提示した時間を黒色の横棒で示した。刺激開始時刻を時刻 0 とした。(a)HVC。(b)Area X。

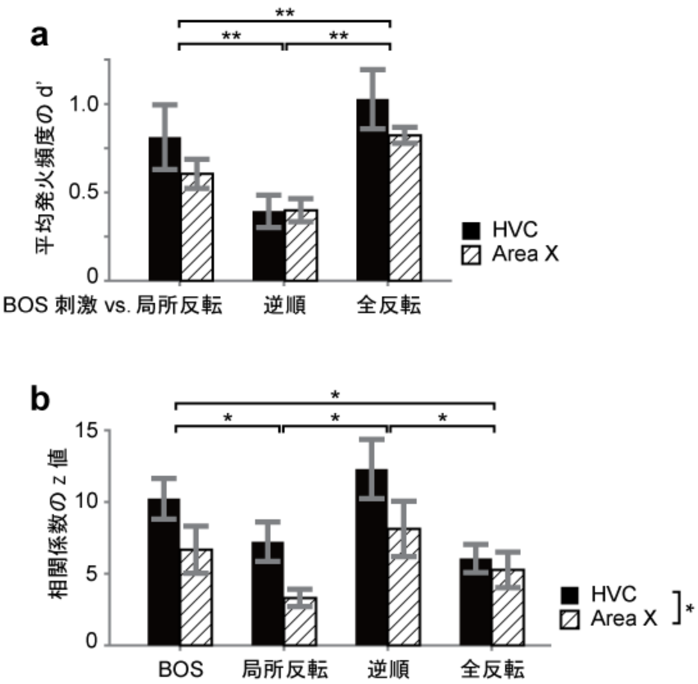


図 4.6｜平均発火頻度の上昇の d'と、瞬時発火頻度の平均相関係数の z 値

(a)平均発火頻度の上昇の *d'*。黒色の棒グラフで HVC の *d'*を、斜線の棒グラフで Area X の *d'*を示した。(b)瞬時発火頻度の平均相関係数の *z* 値。エラーバーは標準誤差を表す（$N = 10$）。*は *p* 値を補正した t 検定により $p < .005$ を、**は $p < .001$ を表す。

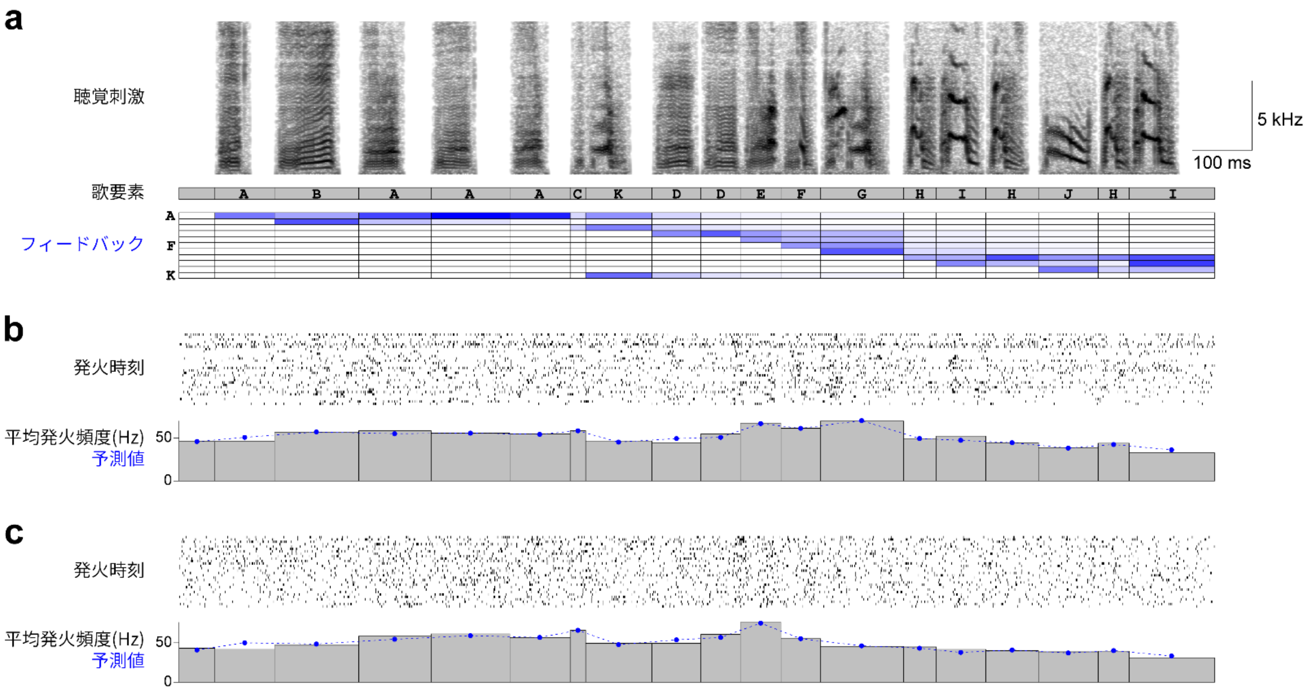


図 4.7｜直前の系列の情報を含んだ神経活動

(a)上から、ある個体の BOS 刺激のスペクトログラム、要素区間（ある要素の開始から次の要素の開始まで)、要素区間におけるフィードバック値の平均値。フィードバック値の大きさは白色から青色へのカラースケールで表した。(b)同個体の HVC における神経活動。上から、発火時刻のラスタープロット、要素区間における平均発火頻度（灰色長方形）と重回帰分析による予測値（青色)。発火時刻のラスタープロットでは、1 行に刺激提示の 1 試行を示し、その間に発火した時刻を黒色の点で示した。(c)同個体の Area X における神経活動。

第 5 章　総合討論

歌の自動認識

第 2 章の研究では、歌の自動認識を行った。ジュウシマツの歌の 3 特徴である要素の分類可能性・時間情報の重要性・確率的な歌文法の存在、に着目し、自動認識の問題をそれぞれに対応した小問題に分割することで問題の見通しを良くした。結果として、研究の場面で実用性のある認識器を構成することができた。本研究により、時間情報よりも要素の分類精度が重要であるときは要素の検出を先に行い、時間情報と分類精度のどちらも必要なときは深層ニューラルネットワークにより歌文法を考慮した分類を行った後、隠れマルコフモデルにより歌文法を考慮した要素の検出を行うと良いということがわかった。本研究の成果は、上述の 3 特徴をもつ系列発声であれば、他の鳴禽や鳴禽以外の種にも応用することができる。

歌文法の表現とその変化

第 3 章の研究では、まず、歌文法を表現するのに最も適したモデルを複数の候補の中から選択した。結果としてジュウシマツの歌では、歌開始部における要素の出現確率が他の部分と異なり、繰り返しを分岐の一種として扱うことができ、要素の出現確率が依存する直前の部分系列の長さがその部分系列によって変化する、ということがわかった。次に、選択されたモデルを用いて、歌文法の緩やかな変化について検討した。その結果、歌文法が時間帯や日によって変化し、時間帯による変化が日をまたいで類似することがわかった。時間帯による変化が日をまたいで類似することから、歌文法の変化に概日リズムが影響すると考えられる。しかし、歌文法が日をまたいで完全に一致す

るわけではなく、日によって異なっていたので、概日リズムだけではない日ごとの個体の状態も歌文法に影響していると考えられる。最後に、選択されたモデルを参考に、生体内で実装可能なモデルを構築した。結果として、新たに構築したモデルが、部分列を離散的に記憶する条件付き確率によるモデルよりも、観測された系列を良く予測した。本研究により、要素の出現確率が直前の要素の履歴を用いたニューラルネットワークによって予測できることがわかった。

神経活動との対応

第 4 章の研究では、履歴情報を入力とするニューラルネットワークモデルを実験的に検証した。歌の維持に重要である 2 種類の神経核（HVC と Area X）から、自身の歌を聴いているときの神経活動を記録した。まず、それぞれの神経核の性質を比較したところ、両神経核の活動が歌の局所的な構造に依存していることがわかった。また、発火時刻は HVC のほうが Area X よりもばらつきが小さかった。この結果から、Area X では HVC からの入力を時間的に積分して処理している可能性が示唆された。次に、それぞれの神経核が歌の系列のフィードバックを表現しているかどうかを検討した。その結果、一部の個体で直前の系列のフィードバックと相関する神経活動の変動が見られた。この結果から、前章で提案した系列の予測モデルの生物学的妥当性が支持された。

本研究の限界

本研究で用いた歌の自動認識の手法には、事前に手動で要素を検出・分類した歌のデータが必要である。言い換えると、その後の歌文法に関する解析結果が、全て初めの手動による歌認識の影響を受ける。歌の自動認識に用いたデータでは交差検証による汎化誤差が小さかったので、手動認識

の客観性はある程度担保される。それ以外のデータの手動認識も、歌の自動認識において手動認識を行った研究者と同じ者（私）が行ったので、客観性は歌の自動認識のデータと同程度であると推測できる。

本研究によって、歌文法が時間帯によって変化することがわかった。しかし、本研究からは、その変化が歌の維持に必要であるかどうかを証明することはできない。過去の研究から、歌の維持には自身の歌を聴くことが必要であることと(Okanoya & Yamaguchi, 1997)、要素内の音響的構造を維持するためには音響的構造のばらつきによる探索が必要であることがわかっている(Ali et al., 2013)。したがって、緩やかに変化する歌文法にも、様々な系列パターンを発声することにより探索の幅を広めるという機能があるかもしれない。

神経生理学の実験では、対象とした神経核に存在する全細胞の活動を記録することは現実的に不可能である。もし神経細胞の性質によって記録されやすさに差があった場合、本研究の結果は対象とした神経核にある一部の性質の細胞の活動のみを反映したものかもしれない。系列のフィードバックを表現していた細胞が一部の個体にしか見られなかったのも、神経核にある細胞を網羅できていなかったからかもしれない。

要素系列の変化

本研究により、要素系列の変化には、数時間単位の緩やかな変化と、直前の系列のフィードバックによる短時間の変化が存在することがわかった（図 5.1）。さらに、加齢による遅い変化もあるが

(James & Sakata, 2014)、これは筋肉や神経系の衰えによるもので、おそらく歌の維持とは無関係であるため、本研究では対象としなかった。数時間単位の緩やかな変化は、概日リズムや日ごとの個体の状態によるものであると考えられる。直前の系列のフィードバックによる短時間の変化は、フィードバックの情報が運動前野と基底核の活動に影響を与えることで、要素の出現確率を変化させていると考えられる。鳴禽は数時間単位の変化と数百ミリ秒単位の変化を組み合わせて、効果的に音声コミュニケーションを行っているのかもしれない。

要素系列の変化の神経機構

直前の系列によって、ある要素の次に出現する要素の確率が変化することがわかった。HVC に、ある要素のある部分に対応する運動指令を出力する神経細胞の集団が存在するという考えが有力である(Doya & Sejnowski, 1995; Fee et al., 2004)。この仮説では、要素の出現確率は、細胞集団間の結合強度によって表現される(Hanuschkin et al., 2011; Wittenbach et al., 2015)。そして、どれかの要素に対応する細胞集団の活動がある閾値を超えると、抑制性の介在神経細胞により他の要素に対応する神経細胞が側方抑制を受け、次に活動する細胞集団が 1 つに決まる。

HVC は履歴の情報として、聴覚野からの聴覚情報と、運動指令の遠心性コピーを受け取る(Bolhuis et al., 2010; Troyer & Doupe, 2000a, 2000b)。履歴情報が次の要素の出現確率に影響を与える仕組みとして、次の 2 つを考えることができる。1 つ目の仕組みは、履歴情報が介在神経細胞に入力し、介在神経細胞の活動を変化させるというものである。介在神経細胞の活動の変化が側方抑制の程度を変化させ、次に活動する運動神経細胞を変化させるのかもしれない。2 つ目の仕組みは、履歴情報が直

接運動神経細胞に作用するというものである。履歴情報の入力を受けた細胞集団の活動度合いが変化することにより、要素の出現確率が変化するのかもしれない。

歌文法の緩やかな変動から、次に出現する要素の確率が個体の状態によって変化することが示唆された。脳の温度によって歌文法が変化するという研究がある。過去の研究から、脳の温度が時間帯によってほぼ規則的に変化すると考えられるので(Aronov & Fee, 2012)、本研究で得られた日ごとの変化を脳の温度によって完全に説明することはできない。

個体の状態を歌神経核に伝えているのは、神経修飾物質である可能性がある。ノルエピネフリンやコリン性の入力が HVC の聴覚情報への反応を変化させることがわかっている(Prather & Mooney, 2004)。よって、個体の状態によって神経修飾物質の濃度が変化し、それにより聴覚フィードバックへの感受性が変化するのかもしれない。本研究の結果から、歌神経核の聴覚フィードバックへの感受性が変化すると、要素の出現確率も変化すると考えられる。

速い変動の機能

ジュウシマツは複雑な歌文法に従った歌をうたう。しかし、複雑な系列を全て記憶するのは記憶容量の効率が悪い。そこで、本研究のモデルと同様に、ジュウシマツも履歴情報を用いて複雑な歌文法をコンパクトに保持しているのかもしれない。履歴情報によって要素系列が変動するということは、実際の生体内でも履歴情報を用いることで複雑な歌文法をコンパクトに表現している可能性を示唆する。

鳴禽の歌にかぎらず、一般の複雑な運動系列においても、履歴情報を用いて系列の規則をコンパクトに保持することが可能かもしれない。このモデルでは、動作の規則を全て記憶するのではなく、直前の履歴と次の動作の対応を保持しておけば良い。本研究で確立した、大量の系列データを効率良く解析する手法を用いると、他の種における系列行動についての研究が進展しやすくなる。また、この考え方は、自動で動作を行うロボット等の効率良い学習や記憶の表現に応用できるかもしれない。

ジュウシマツの歌では歌文法が複雑であるほうが求愛の効果が大きいと考えられている(Okanoya, 2004b; Soma, Takahasi, Hasegawa, & Okanoya, 2006)。よって、フィードバックを利用した系列の速い変動には、毎回同一の系列をうたうのではなく、変化に富む刺激的な歌による効果的な求愛を促進する機能があるのかもしれない。

遅い変動の機能

鳴禽は、歌の音響構造を維持するために能動的に音響構造をばらつかせている。一般に運動学習を効率良く行うためには、適切な量の探索が重要である。ヒトを対象とした運動学習課題の研究でも同様の結果が示されている(Wu, Miyamoto, Castro, Olveczky, & Smith, 2014)。歌文法の遅い変動は、歌文法を効率良く維持するために進化してきたのかもしれない。学習が完成に近づくと、探索の強度は小さいほうが効率良く効果を維持できる。よって、遅い変動の量が小さかったのは、それが最適な量であるからかもしれない。ジュウシマツは、歌文法の遅い変動による探索を用いて、複雑な歌文法を失わないように維持することで、効果的な求愛を可能にしているのかもしれない。

本研究で対象とした行動は歌の維持のみであるので、新たな動作の学習原理については本研究の結果から直接証明することはできない。しかし、新たな動作の学習と学習された行動の維持のメカニズムには類似する点が多いため、新たな動作の学習にも同様の議論を適用することができると予想できる。

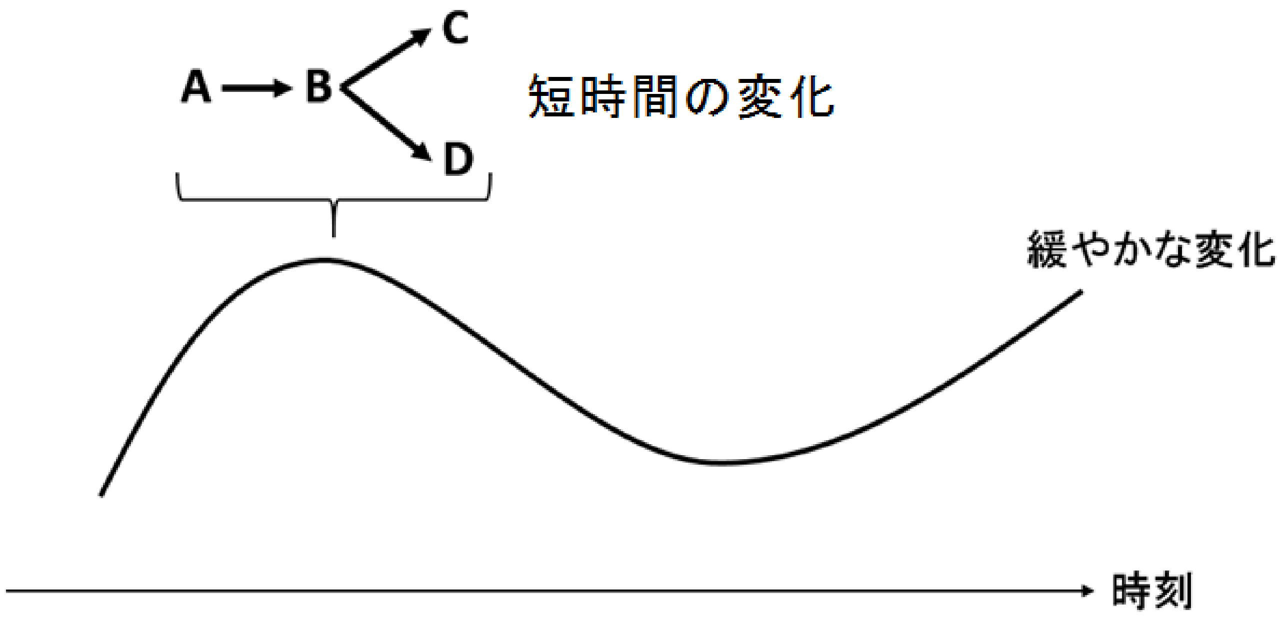


図 5.1｜歌文法の変動

本研究の結果からわかった歌文法の変動を表す模式図。

引用文献

Ali, F., Otchy, T. M., Pehlevan, C., Fantana, A. L., Burak, Y., & Ölveczky, B. P. (2013). The basal ganglia is necessary for learning spectral, but not temporal, features of birdsong. *Neuron, 80*(2), 494-506.

Amador, A., Perl, Y. S., Mindlin, G. B., & Margoliash, D. (2013). Elemental gesture dynamics are encoded by song premotor cortical neurons. *Nature, 495*(7439), 59-64. doi: 10.1038/nature11967

Anderson, S. E., Dave, A. S., & Margoliash, D. (1996). Template-based automatic recognition of birdsong syllables from continuous recordings. *J Acoust Soc Am, 100*(2 Pt 1), 1209-1219.

Aronov, D., & Fee, M. S. (2012). Natural changes in brain temperature underlie variations in song tempo during a mating behavior. *PLoS ONE, 7*(10), e47856. doi: 10.1371/journal.pone.0047856

Bahl, L. R., Brown, P. F., De Souza, P. V., & Mercer, R. L. (1986). *Maximum mutual information estimation of hidden Markov model parameters for speech recognition.* Paper presented at the proc. icassp.

Bejerano, G., & Yona, G. (2001). Variations on probabilistic suffix trees: statistical modeling and prediction of protein families. *Bioinformatics, 17*(1), 23-43.

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation Learning: A Review and New Perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on, 35*(8), 1798-1828. doi: 10.1109/tpami.2013.50

Bengio, Y., Thibodeau-Laufer, E., Alain, G., & Yosinski, J. (2013). Deep generative stochastic networks trainable by backprop. *arXiv preprint arXiv:1306.1091*.

Benzeghiba, M., De Mori, R., Deroo, O., Dupont, S., Erbes, T., Jouvet, D., . . . Wellekens, C. (2007). Automatic speech recognition and speech variability: A review. *Speech Communication, 49*(10–11), 763-786. doi:

10.1016/j.specom.2007.02.006

Berwick, R. C., Okanoya, K., Beckers, G. J., & Bolhuis, J. J. (2011). Songs to syntax: the linguistics of birdsong. [Review]. *Trends Cogn Sci, 15*(3), 113-121. doi: 10.1016/j.tics.2011.01.002

Bolhuis, J. J., Okanoya, K., & Scharff, C. (2010). Twitter evolution: converging mechanisms in birdsong and human speech. *Nat Rev Neurosci, 11*(11), 747-759. doi: 10.1038/nrn2931

Bourlard, H. A., & Morgan, N. (2012). *Connectionist speech recognition: a hybrid approach* (Vol. 247): Springer Science & Business Media.

Brainard, M. S., & Doupe, A. J. (2002). What songbirds teach us about learning. *Nature, 417*(6886), 351-358. doi: 10.1038/417351a

Charlesworth, J. D., Warren, T. L., & Brainard, M. S. (2012). Covert skill learning in a cortical-basal ganglia circuit. *Nature, 486*(7402), 251-255. doi: 10.1038/nature11078

Chi, Z., & Margoliash, D. (2001). Temporal precision and temporal drift in brain and behavior of zebra finch song. *Neuron, 32*(5), 899-910.

Ciresan, D., Meier, U., & Schmidhuber, J. (2012, 16-21 June 2012). *Multi-column deep neural networks for image classification.* Paper presented at the Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on.

Doupe, A. J. (1997). Song- and order-selective neurons in the songbird anterior forebrain and their emergence during vocal development. *J Neurosci, 17*(3), 1147-1167.

Doupe, A. J., & Konishi, M. (1991). Song-selective auditory circuits in the vocal control system of the zebra finch. *Proc Natl Acad Sci U S A, 88*(24), 11339-11343.

Doya, K., & Sejnowski, T. J. (1995). A novel reinforcement model of birdsong vocalization learning. *Advances in neural information processing systems*, 101-108.

Fantana, A. L., & Kozhevnikov, A. (2014). Finding motifs in birdsong data in the presence of acoustic noise and temporal jitter. *Behav Neurosci*, *128*(2), 228-236. doi: 10.1037/a0035985

Fee, M. S., & Goldberg, J. H. (2011). A hypothesis for basal ganglia-dependent reinforcement learning in the songbird. *Neuroscience, 198*, 152-170. doi: 10.1016/j.neuroscience.2011.09.069

Fee, M. S., Kozhevnikov, A. A., & Hahnloser, R. H. (2004). Neural mechanisms of vocal sequence generation in the songbird. *Ann N Y Acad Sci*, *1016*, 153-170. doi: 10.1196/annals.1298.022

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics, 36*(4), 193-202. doi: 10.1007/bf00344251

Glaze, C. M., & Troyer, T. W. (2006). Temporal structure in zebra finch song: implications for motor coding. *J Neurosci*, *26*(3), 991-1005. doi: 10.1523/JNEUROSCI.3387-05.2006

Glaze, C. M., & Troyer, T. W. (2007). Behavioral measurements of a temporally precise motor code for birdsong. *J Neurosci, 27*(29), 7631-7639. doi: 10.1523/JNEUROSCI.1065-07.2007

Goldberg, J. H., Adler, A., Bergman, H., & Fee, M. S. (2010). Singing-related neural activity distinguishes two putative pallidal cell types in the songbird basal ganglia: comparison to the primate internal and external pallidal segments. *J Neurosci*, *30*(20), 7088-7098. doi: 10.1523/JNEUROSCI.0168-10.2010

Green, D. M., & Swets, J. A. (1966). *Signal detection theory and psychophysics* (Vol. 1): Wiley New York.

Hahnloser, R. H., Kozhevnikov, A. A., & Fee, M. S. (2002). An ultra-sparse code underlies the generation of neural sequences in a songbird. *Nature, 419*(6902), 65-70. doi: 10.1038/nature00974

Hanuschkin, A., Diesmann, M., & Morrison, A. (2011). A reafferent and feed-forward model of song syntax generation in the Bengalese finch. *Journal of computational neuroscience, 31*(3), 509-532.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*.

Hinton, G., Li, D., Dong, Y., Dahl, G. E., Mohamed, A., Jaitly, N., . . . Kingsbury, B. (2012). Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *Signal Processing Magazine, IEEE, 29*(6), 82-97. doi: 10.1109/msp.2012.2205597

Hinton, G., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hosino, T., & Okanoya, K. (2000). Lesion of a higher-order song nucleus disrupts phrase level complexity in Bengalese finches. *Neuroreport, 11*(10), 2091-2095.

Igel, C., & Hüsken, M. (2000). *Improving the Rprop learning algorithm.* Paper presented at the Proceedings of the second international ICSC symposium on neural computation (NC 2000).

James, L. S., & Sakata, J. T. (2014). Vocal motor changes beyond the sensitive period for song plasticity. *J Neurophysiol, 112*(9), 2040-2052. doi: 10.1152/jn.00217.2014

Kao, M. H., Doupe, A. J., & Brainard, M. S. (2005). Contributions of an avian basal ganglia-forebrain circuit to real-time modulation of song. *Nature, 433*(7026), 638-643. doi: 10.1038/nature03127

Katahira, K., Suzuki, K., Okanoya, K., & Okada, M. (2011). Complex sequencing rules of birdsong can be explained by simple hidden Markov processes. *PLoS ONE, 6*(9), e24516.

Kelley, D. B., & Nottebohm, F. (1979). Projections of a telencephalic auditory nucleus-field L-in the canary. *J Comp*

Neurol, 183(3), 455-469. doi: 10.1002/cne.901830302

Kershenbaum, A., Bowles, A. E., Freeberg, T. M., Jin, D. Z., Lameira, A. R., & Bohn, K. (2014). Animal vocal sequences: not the Markov chains we thought they were. *Proc Biol Sci, 281*(1792). doi: 10.1098/rspb.2014.1370

Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kogan, J. A., & Margoliash, D. (1998). Automated recognition of bird song elements from continuous recordings using dynamic time warping and hidden Markov models: A comparative study. *J Acoust Soc Am, 103*(4), 2185-2196. doi: 10.1121/1.421364

Koumura, T., Seki, Y., & Okanoya, K. (2014). Local structure sensitivity in auditory information processing in avian song nuclei. *Neuroreport, 25*(8), 562-568. doi: 10.1097/WNR.0000000000000136

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE, 86*(11), 2278-2324. doi: 10.1109/5.726791

Lewicki, M. S., & Arthur, B. J. (1996). Hierarchical organization of auditory temporal context sensitivity. *J Neurosci, 16*(21), 6987-6998.

Lin, M., Chen, Q., & Yan, S. (2014). Network In Network. *arXiv preprint arXiv:1312.4400*.

Lipkind, D., Marcus, G. F., Bemis, D. K., Sasahara, K., Jacoby, N., Takahasi, M., . . . Tchernichovski, O. (2013). Stepwise acquisition of vocal combinatorial capacity in songbirds and human infants. *Nature, 498*(7452), 104-108. doi: 10.1038/nature12173

Markowitz, J. E., Ivie, E., Kligler, L., & Gardner, T. J. (2013). Long-range order in canary song. *PLoS Comput Biol, 9*(5), e1003052. doi: 10.1371/journal.pcbi.1003052

Menyhart, O., Kolodny, O., Goldstein, M. H., DeVoogd, T. J., & Edelman, S. (2015). Juvenile zebra finches learn the underlying structural regularities of their fathers' song. *Front Psychol, 6*, 571. doi: 10.3389/fpsyg.2015.00571

Nakamura, K. Z., & Okanoya, K. (2004). Neural correlates of song complexity in Bengalese finch high vocal center. *Neuroreport, 15*(8), 1359-1363.

Nottebohm, F., Stokes, T. M., & Leonard, C. M. (1976). Central control of song in the canary, Serinus canarius. *J Comp Neurol, 165*(4), 457-486. doi: 10.1002/cne.901650405

Okanoya, K. (2004a). The Bengalese finch: a window on the behavioral neurobiology of birdsong syntax. *Ann N Y Acad Sci, 1016*, 724-735. doi: 10.1196/annals.1298.026

Okanoya, K. (2004b). Song syntax in Bengalese finches: proximate and ultimate analyses. *Advances in the Study of Behavior, 34*, 297-346.

Okanoya, K., Ikebuchi, M., Uno, H., & Watanabe, S. (2001). Left-side dominance for song discrimination in Bengalese finches (Lonchura striata var. domestica). *Animal Cognition, 4*(3-4), 241-245. doi: 10.1007/s10071-001-0120-9

Okanoya, K., & Yamaguchi, A. (1997). Adult Bengalese finches (Lonchura striata var. domestica) require real-time auditory feedback to produce normal song syntax. *J Neurobiol, 33*(4), 343-356.

Olveczky, B. P., Andalman, A. S., & Fee, M. S. (2005). Vocal experimentation in the juvenile songbird requires a basal ganglia circuit. *PLoS Biol, 3*(5), e153. doi: 10.1371/journal.pbio.0030153

Percival, D. B., & Walden, A. T. (1993). *Spectral analysis for physical applications*: Cambridge University Press.

Prather, J. F., & Mooney, R. (2004). Neural correlates of learned song in the avian forebrain: simultaneous representation of self and others. *Curr Opin Neurobiol, 14*(4), 496-502.

Rajan, R., & Doupe, A. J. (2013). Behavioral and neural signatures of readiness to initiate a learned motor sequence. *Curr Biol, 23*(1), 87-93. doi: 10.1016/j.cub.2012.11.040

Riedmiller, M., & Braun, H. (1993, 1993). *A direct adaptive method for faster backpropagation learning: the RPROP algorithm.* Paper presented at the Neural Networks, 1993., IEEE International Conference on.

Rohrmeier, M., Zuidema, W., Wiggins, G. A., & Scharff, C. (2015). Principles of structure building in music, language and animal song. *Philos Trans R Soc Lond B Biol Sci, 370*(1664), 20140097. doi: 10.1098/rstb.2014.0097

Ron, D., Singer, Y., & Tishby, N. (1996). The power of amnesia: Learning probabilistic automata with variable memory length. *Machine learning, 25*(2-3), 117-149.

Sakata, J. T., & Brainard, M. S. (2008). Online contributions of auditory feedback to neural activity in avian song control circuitry. *J Neurosci, 28*(44), 11378-11390. doi: 10.1523/JNEUROSCI.3254-08.2008

Sasahara, K., Kakishita, Y., Nishino, T., Takahasi, M., & Okanoya, K. (2006). *A reversible automata approach to modeling birdsongs.* Paper presented at the Computing, 2006. CIC'06. 15th International Conference on.

Scharff, C., & Adam, I. (2013). Neurogenetics of birdsong. *Curr Opin Neurobiol, 23*(1), 29-36. doi: 10.1016/j.conb.2012.10.001

Seki, Y., Suzuki, K., Takahasi, M., & Okanoya, K. (2008). Song motor control organizes acoustic patterns on two levels in Bengalese finches (Lonchura striata var. domestica). *J Comp Physiol A Neuroethol Sens Neural Behav Physiol, 194*(6), 533-543. doi: 10.1007/s00359-008-0328-0

Sober, S. J., & Brainard, M. S. (2012). Vocal learning is constrained by the statistics of sensorimotor experience. *Proceedings of the National Academy of Sciences, 109*(51), 21099-21103.

Sober, S. J., Wohlgemuth, M. J., & Brainard, M. S. (2008). Central contributions to acoustic variation in birdsong. *J Neurosci, 28*(41), 10370-10379. doi: 10.1523/JNEUROSCI.2448-08.2008

Soma, M., Takahasi, M., Hasegawa, T., & Okanoya, K. (2006). Trade-offs and correlations among multiple song features in the Bengalese Finch. *Ornithological Science, 5*(1), 77-84.

Tachibana, R. O., Koumura, T., & Okanoya, K. (2015). Variability in the temporal parameters in the song of the Bengalese finch (Lonchura striata var. domestica). *Journal of Comparative Physiology A, 201*(12), 1157-1168.

Tachibana, R. O., Oosugi, N., & Okanoya, K. (2014). Semi-automatic classification of birdsong elements using a linear support vector machine. *PLoS One, 9*(3), e92584. doi: 10.1371/journal.pone.0092584

Tchernichovski, O., Nottebohm, F., Ho, C. E., Pesaran, B., & Mitra, P. P. (2000). A procedure for an automated measurement of song similarity. *Anim Behav, 59*(6), 1167-1176. doi: 10.1006/anbe.1999.1416

Theunissen, F. E., & Doupe, A. J. (1998). Temporal and spectral sensitivity of complex auditory neurons in the nucleus HVc of male zebra finches. *J Neurosci, 18*(10), 3786-3802.

Troyer, T. W., & Doupe, A. J. (2000a). An associational model of birdsong sensorimotor learning I. Efference copy and the learning of song syllables. *J Neurophysiol, 84*(3), 1204-1223.

Troyer, T. W., & Doupe, A. J. (2000b). An associational model of birdsong sensorimotor learning II. Temporal hierarchies and the learning of song sequence. *J Neurophysiol, 84*(3), 1224-1239.

Tumer, E. C., & Brainard, M. S. (2007). Performance variability enables adaptive plasticity of 'crystallized' adult birdsong. *Nature, 450*(7173), 1240-1244. doi: 10.1038/nature06390

Warren, T. L., Charlesworth, J. D., Tumer, E. C., & Brainard, M. S. (2012). Variable sequencing is actively maintained in a well learned motor skill. *J Neurosci, 32*(44), 15414-15425. doi: 10.1523/JNEUROSCI.1254-12.2012

Wittenbach, J. D., Bouchard, K. E., Brainard, M. S., & Jin, D. Z. (2015). An adapting auditory-motor feedback loop can contribute to generating vocal repetition. *PLoS Comput Biol, 11*(10), e1004471. doi: 10.1371/journal.pcbi.1004471

Wu, H. G., Miyamoto, Y. R., Castro, L. N. G., Olveczky, B. P., & Smith, M. A. (2014). Temporal structure of motor variability is dynamically regulated and predicts motor learning ability. *Nat Neurosci, 17*(2), 312-321.

Yamashita, Y., Okumura, T., Okanoya, K., & Tani, J. (2011). Cooperation of deterministic dynamics and random noise in production of complex syntactical avian song sequences: a neural network model. *Front Comput Neurosci, 5*, 18. doi: 10.3389/fncom.2011.00018

Yu, A. C., & Margoliash, D. (1996). Temporal hierarchical control of singing in birds. *Science, 273*(5283), 1871-1875.

Zeiler, M., & Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In D. Fleet, T. Pajdla, B. Schiele & T. Tuytelaars (Eds.), *Computer Vision – ECCV 2014* (Vol. 8689, pp. 818-833): Springer International Publishing.

謝辞

博士論文の執筆にあたって、多くの方のお力添えをいただきました。特に主査の岡ノ谷先生、副査の長谷川先生・四本先生・本吉先生・工藤先生に心より感謝いたします。また、指導教官である岡ノ谷先生をはじめ、岡ノ谷研究室の研究員・学生のみなさまにも心より感謝いたします。

付録 A　音圧と長さの閾値を用いた区間検出の手順

アルゴリズム：音圧と長さの閾値を用いた区間検出。

入力：スペクトログラムX。

出力：要素区間$i_{element} \in I_{element}$。

ハイパーパラメータ：音圧の閾値θ_{amp}、無音区間の最大長$\theta_{silence}$、要素区間の最小長$\theta_{element}$。

(1) 時刻tにおける音圧A_tを計算する。

$$A_t = \sum_{f=1\ \mathrm{kHz}}^{8\ \mathrm{kHz}} X_{ft}$$

ただしX_{ft}を時刻t、周波数fにおける対数振幅スペクトログラムの値とする。

(2) 音圧がθ_{amp}より大きい区間$i_{sound} \in I_{sound}$を抽出する。

(3) I_{sound}の隣り合う区間で、間の無音区間が$\theta_{silence}$より短い 2 区間を結合する。

(4) I_{sound}中の区間で、長さが$\theta_{element}$以上の区間を$I_{element}$に追加する。

付録 B　CDNN の訓練の手順

アルゴリズム：CDNN の訓練。

入力：$n_{training}$個の小訓練用データ$X^k_{training}$ $\left(1 \le k \le n_{training}\right)$と対応する要素区間$T^k_{training}$、$n_{validation}$個の小検証用データ$X^k_{validation}$ $(1 \le k \le n_{validation})$と対応する要素区間$T^k_{\mathrm{validation}}$。

出力：各層lのパラメータ$\theta_l = [W_l, b_l]$。

ハイパーパラメータ：学習率の初期値α_0。

(1)　学習率を$\alpha = \alpha_0$とする。

(2)　次の(3)から(10)を 3 回繰り返す。

(3)　全ての小検証用データ$X^k_{validation}$について、本文中の式(1)から(6)により、誤差C^kを計算する。

(4)　$\sum_k C^k$が減少しなくなるまで、(3)から(9)を繰り返す。

(5)　ある小訓練用データ$X^k_{training}$ $\left(1 \le k \le n_{training}\right)$をランダムに選ぶ。

(6)　本文中の式(1)から(5)により、$X^k_{training}$を入力としたときのソフトマックス層の値Yを計算する。

(7)　誤差Cの、ソフトマックス層の値についての偏微分を計算する。

$$\frac{\partial C}{\partial Y_{ij}} = Y_{ij} - \delta_{it_j}$$

ただしY_{ij}をソフトマックス層の時刻j におけるi次元目の値、t_jを時刻jにおける正解の要素、δ_{it_j}をクロネッカーのデルタとする。クロネッカーのデルタδ_{it_j}は$i = t_j$のとき 1 となり、$i \ne t_j$のとき 0 となる。

(8) ソフトマックス層より下位の各層について、偏微分$\frac{\partial C}{\partial Y_l}$と$\frac{\partial C}{\partial \theta}$を計算する。

(9) 本文中の式(7)と(8)により、各層のパラメータθ_lを更新する。

(10) 学習率を半分にする。

$$\alpha \leftarrow \alpha/2$$

付録 C　Viterbi アルゴリズムによる全体配列の手順

アルゴリズム：Viterbi アルゴリズムによる全体配列。

入力：要素分類$\sigma \in \Sigma$が与えられたときのi番目のスペクトログラムX_iの生成確率 $P(X_i|\sigma)$

出力：要素系列$\sigma_1\sigma_2 \cdots \sigma_n$

パラメータ：遷移確率$P(\sigma|\sigma_{i-2}\sigma_{i-1})$

(1) 全ての$\sigma \in \Sigma$について、$q_{1,\sigma} = \frac{1}{|\Sigma|} P(X_1|\sigma)$。

(2) $i = 2$ からnについて、

(a) $q_{i,\sigma'\sigma} = \max_{\sigma''\sigma'}\left(q_{i-1,\sigma''\sigma'} P(\sigma|\sigma''\sigma')\right) P(X_i|\sigma)$

(b) $\psi_{i,\sigma'\sigma} = \operatorname*{argmax}_{\sigma''\sigma'}\left(q_{i-1,\sigma''\sigma'} P(\sigma|\sigma''\sigma')\right)$

(3) $i = 2$ からnについて、$\sigma_{i-1}\sigma_i = \psi_{i,\sigma'\sigma}$

付録 D　歌自動認識のプログラムのソースコード

歌自動認識のプログラムは、次の 47 ファイルのソースコードよって構成した。各ファイルの説明を表 D.1 に示した。

表 D.1 | ソースコードの説明

ファイル名	内容
BatchData.java	DNN のデータを扱う。
DnnComputation.java	CDNN の主な計算を行う。
HmmComputation.java	HMM の主な計算を行う。
Sequence.java	要素系列を扱う。
STFTParam.java	短時間離散フーリエ変換のパラメータを扱う。
Thresholding.java	音圧と長さの閾値による区間検出を行う
ViterbiSequencer.java	Viterbi アルゴリズムの計算を行う。
ConvLayer.java	畳み込み層を扱う。
DataLayer.java	入力層を扱う。
Layer.java	DNN の層を扱う。
NonDataLayer.java	DNN の入力層以外の層を扱う。
OutputLayer.java	DNN の出力層を扱う。

表 D.1｜ソースコードの説明（続き）

ファイル名	内容
ParamLayer.java	DNN の、パラメータを持つ層を扱う。
PoolLayer.java	Max-pooling 層を扱う。
SeqSoftmaxConvLayer.java	ソフトマックス層を扱う。
Adam.java	パラメータの更新を計算する。
GradientDescent.java	パラメータの更新を計算する。
Model.java	DNN のネットワークを扱う。
SeqNetwork.java	CDNN のネットワークを扱う。
ActivationMode.java	活性化関数の種類を扱う。
Cuda.java	グラフィックプロセッサによる計算を扱う。
CudaDriver.java	グラフィックプロセッサによる計算を扱う。
CudaException.java	グラフィックプロセッサによる計算中に発生した例外を扱う。
Cudnn.java	CDNN の計算を扱う。
CudnnException.java	CDNN の計算中に発生した例外を扱う。
CudnnLibrary.java	CDNN の計算を扱う。
FloatType.java	実数値の表現方法を扱う。
IntType.java	整数値の表現方法を扱う。
Pointer.java	グラフィックプロセッサによるデータ表現を扱う。

表 D.1｜ソースコードの説明（続き）

ファイル名	内容
PoolingMode.java	Pooling 層の種類を扱う。
SoftmaxMode.java	ソフトマックス層の種類を扱う。
ErrorSaving.java	誤差の保存を行う。
Levenshtein.java	Levenshtein 誤差を計算する。
Matching.java	マッチング誤差を計算する。
BdLcGs.java	区間検出→局所分類→全体配列の主な計算を行う。
LcBdGs.java	局所分類→区間検出＆全体配列の主な計算を行う。
LcGsBdGs.java	局所分類＆全体配列→区間検出＆全体配列の主な計算を行う。
ArrayUtils.java	配列を扱う。
CollectionUtils.java	データ集合を扱う。
DnnUtils.java	DNN のパラメータの保存などを行う。
Executor.java	並列計算を扱う。
MathUtils.java	簡単な数学計算を扱う。
Pair.java	データ集合を扱う。
RandomUtils.java	乱数を扱う。
SoundUtils.java	音の処理を行う。
XmlUtils.java	データの保存を行う。

表 D.1｜ソースコードの説明（続き）

ファイル名	内容
kernel.cu	グラフィックプロセッサ上で計算を行う。

BatchData.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package computation;

import java.io.IOException;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.function.IntBinaryOperator;
import java.util.stream.Collectors;

import javax.sound.sampled.UnsupportedAudioFileException;

import org.apache.commons.math3.random.MersenneTwister;

import computation.Sequence.LabelList;
import computation.Sequence.Note;
import computation.Sequence.WavePosition;
import cudnn.network.SeqNetwork;
import no.uib.cipr.matrix.NotConvergedException;
import utils.ArrayUtils;
import utils.DnnUtils;
import utils.MathUtils;
import utils.Pair;
import utils.RandomUtils;
import utils.SoundUtils;

/**
 * This class handles data for DNN computation.
 *
 * @author koumura
 *
 */
public class BatchData
{
  private ArrayList<ArrayList<float[]>> batchData;
  private ArrayList<byte[]> batchLabel;
  private int dataLayerHeight, maxBatchSize, numBatch, labelHeight, sumValidLabelSize, packedHeight;
  private ArrayList<int[]> batchValidLabelSize;
  private ArrayList<float[]> spectrogram;
  private ArrayList<ArrayList<Integer>> packedSpectrogram;
  private List<Sequence> sequence;

  private ArrayList<float[]> inputData;
  private ArrayList<byte[]> inputLabel;

  private ArrayList<int[]> specLabelPackBegin, packBatchIndex;

  private int numLowerLabel, finalInputHeight;
  private IntBinaryOperator silentLabelFunc;
  private byte silentLabel;
  private int specMarginBegin, specMarginEnd;

  private STFTParam stftParam;

  public int getSumValidLabelSize() {
    return sumValidLabelSize;
  }

  public int specMarginSum(){return specMarginBegin+specMarginEnd;}

  public ArrayList<byte[]> getBatchLabel() {
    return batchLabel;
  }

  public int getDataLayerHeight() {
    return dataLayerHeight;
  }

  public int getMaxBatchSize() {
    return maxBatchSize;
  }

  public int getPackedHeight() {
    return packedHeight;
  }

  public int getNumBatch() {
    return numBatch;
  }

  public ArrayList<ArrayList<float[]>> getBatchData() {
    return batchData;
  }

  public ArrayList<int[]> getSpecLabelPackBegin() {
    return specLabelPackBegin;
  }

  public ArrayList<int[]> getPackBatchIndex() {
    return packBatchIndex;
  }

  public ArrayList<float[]> getSpectrogram() {
    return spectrogram;
  }

  private BatchData(ArrayList<float[]> spectrogram, int packedHeight, ArrayList<ArrayList<Integer>>
packedSpectrogram, List<Sequence> sequence, int numLowerLabel, IntBinaryOperator silentLabelFunc, int
specMarginBegin, int specMarginEnd, int finalInputHeight, STFTParam stftParam)
  {
    this.spectrogram = spectrogram;
    this.packedHeight = packedHeight;
    this.packedSpectrogram=packedSpectrogram;
    this.sequence=sequence;
    this.numLowerLabel=numLowerLabel;
    this.finalInputHeight=finalInputHeight;
    this.silentLabelFunc=silentLabelFunc;
    this.specMarginBegin=specMarginBegin;
    this.specMarginEnd=specMarginEnd;
    this.stftParam=stftParam;
  }

  private static ArrayList<ArrayList<Integer>> packedSpectrogramIndex(int heightUpper, List<float[]> spectrogram,
MersenneTwister random, int inputWidth)
  {
    int[] shuffleSpectrogram;
    if(random!=null) shuffleSpectrogram=RandomUtils.permutation(spectrogram.size(), random);
    else shuffleSpectrogram=ArrayUtils.createSequence(0, spectrogram.size());
    ArrayList<int[]> dataHeight=new ArrayList<int[]>();
    HashMap<Integer, LinkedList<Integer>> inputSpec=new HashMap<>();
    for(int s: shuffleSpectrogram)
    {
      float[] spec=spectrogram.get(s);
      int specHeight=spec.length/inputWidth;
```

```java
      int[] minHeight=null;
      if(dataHeight.size()>0) minHeight=Collections.min(dataHeight, (o1, o2)->o1[1]-o2[1]);

      if(dataHeight.size()>0 && heightUpper-minHeight[1]>=specHeight)
      {
        minHeight[1]+=specHeight;
        inputSpec.putIfAbsent(minHeight[0], new LinkedList<>());
        inputSpec.get(minHeight[0]).add(s);
      }
      else
      {
        inputSpec.putIfAbsent(dataHeight.size(), new LinkedList<>());
        inputSpec.get(dataHeight.size()).add(s);
        dataHeight.add(new int[]{dataHeight.size(), specHeight});
      }
    }
    ArrayList<ArrayList<Integer>> packedIndex=new ArrayList<>(inputSpec.size());
    for(int packed=0; packed<inputSpec.size(); ++packed) packedIndex.add(new
ArrayList<Integer>(inputSpec.get(packed)));
    return packedIndex;
  }

  private static ArrayList<float[]> packSpectrogram(List<ArrayList<Integer>> packedIndex, ArrayList<float[]>
spectrogram, int packedHeight, int inputWidth)
  {
    ArrayList<float[]> inputData=new ArrayList<>();
    for(ArrayList<Integer> index: packedIndex)
    {
      float[] d=new float[packedHeight*inputWidth];
      inputData.add(d);
      int cumLength=0;
      for(int s: index)
      {
        float[] spec=spectrogram.get(s);
        System.arraycopy(spec, 0, d, cumLength, spec.length);
        cumLength+=spec.length;
      }
    }
    return inputData;
  }

  /**
   * Packs input spectrograms for efficient computation.
   * This method must be called before {@link #batch}.
   *
   * @param packedHeight
   * @param inputWidth
   * @param batchSizeUpper
   * @param labelList
   * @param layerFilterSizeList
   * @throws IOException
   */
  public void pack(int packedHeight, int inputWidth, int batchSizeUpper, LabelList labelList,
ArrayList<Pair<DnnUtils.LayerType, Integer>> layerFilterSizeList) throws IOException
  {
    this.packedHeight=packedHeight;
    dataLayerHeight=DnnUtils.nextSmallestInputSize(layerFilterSizeList, packedHeight);
    this.packedHeight=dataLayerHeight+DnnUtils.sumStride(layerFilterSizeList)-1;
    labelHeight=this.packedHeight-finalInputHeight+1;

    inputData=packSpectrogram(packedSpectrogram, spectrogram, this.packedHeight, inputWidth);

    silentLabel=(byte)silentLabelFunc.applyAsInt(labelList.size(), numLowerLabel);

    inputLabel=new ArrayList<>();
    specLabelPackBegin=new ArrayList<int[]>(spectrogram.size());
    for(int i=0; i<spectrogram.size(); ++i) specLabelPackBegin.add(null);
    for(int in=0; in<inputData.size(); ++in)
    {
      byte[] la=new byte[labelHeight];
      Arrays.fill(la, silentLabel);
      inputLabel.add(la);
      int y=0;
      for(int sp: packedSpectrogram.get(in))
      {
        specLabelPackBegin.set(sp, new int[]{in, y});
        for(Note li: sequence.get(sp).getNote())
        {
          int specBegin=stftParam.spectrogramPosition(li.getPosition())+y;
          int specLen=stftParam.spectrogramPosition(li.getPosition()+li.getLength())-specBegin+y;
          for(int sub=0; sub<numLowerLabel; ++sub)
          {
            int begin=MathUtils.subIntervalBegin(specLen, numLowerLabel, sub)+specBegin;
            int end=MathUtils.subIntervalBegin(specLen, numLowerLabel, sub+1)+specBegin;
            Arrays.fill(la, begin, end, (byte)(labelList.indexOf(li.getLabel())*numLowerLabel+sub));
          }
        }
        y+=spectrogram.get(sp).length/inputWidth-(specMarginBegin+specMarginEnd);
        if(sp==packedSpectrogram.get(in).get(packedSpectrogram.get(in).size()-1)) break;
        Arrays.fill(la, y, y+specMarginBegin+specMarginEnd, (byte)-1);
        y+=specMarginBegin+specMarginEnd;
      }
      Arrays.fill(la, y, la.length, (byte)-1);
    }

    numBatch=MathUtils.ceil(inputData.size(), batchSizeUpper);
    maxBatchSize=MathUtils.ceil(inputData.size(), numBatch);
  }

  /**
   * Makes batch inputs.
   * This method must be called before training or recognition with a DNN.
   * @param maxBatchSize
   * @param randomShuffle
   * @param inputWidth
   * @throws IOException
   */
  public void batch(int maxBatchSize, MersenneTwister randomShuffle, int inputWidth) throws IOException
  {
    this.maxBatchSize=maxBatchSize;
    numBatch=MathUtils.ceil(inputData.size(), maxBatchSize);
    batchData=new ArrayList<>(numBatch);
    batchLabel=new ArrayList<>(numBatch);
    int[] shuffleInput;
    if(randomShuffle!=null) shuffleInput=RandomUtils.permutation(inputData.size(), randomShuffle);
    else shuffleInput=ArrayUtils.createSequence(0, inputData.size());
    packBatchIndex=new ArrayList<int[]>();
    for(int p=0; p<inputData.size(); ++p) packBatchIndex.add(null);
    for(int b=0; b<numBatch; ++b)
    {
      batchData.add(new ArrayList<float[]>(maxBatchSize));
      batchLabel.add(new byte[labelHeight*maxBatchSize]);
      for(int i=0; i<maxBatchSize; ++i)
      {
        int in=i*numBatch+b;
        if(in<inputData.size())
        {
          int si=shuffleInput[in];
          batchData.get(b).add(inputData.get(si));
          System.arraycopy(inputLabel.get(si), 0, batchLabel.get(b), i*labelHeight, labelHeight);
          packBatchIndex.set(si, new int[]{b, i});
        }
        else
        {
          for(int i1=i; i1<maxBatchSize; ++i1) batchData.get(b).add(new float[packedHeight*inputWidth]);
          Arrays.fill(batchLabel.get(b), i*packedHeight, batchLabel.get(b).length, (byte)-1);
          break;
        }
      }
    }
  }

  /**
   * Counts the number of valid labels in the batch data.
   * @param network
   */
  public void setValidLabelSize(SeqNetwork network)
  {
```

```java
    batchValidLabelSize=new ArrayList<>(numBatch);
    for(int b=0; b<numBatch; ++b) batchValidLabelSize.add(network.countValidLabelSize(batchLabel.get(b)));

    sumValidLabelSize=batchValidLabelSize.stream().mapToInt(size->Arrays.stream(size).sum()).sum();
  }

  /**
   * Creates {@link BatchData} by loading wave files and converting them into spectrogram.
   *
   * @param numLowerLabel Number of sub-divisions in a single element.
   * @param silentLabelFunc A converter from the number of labels and sub-divisions to the index for the silent
label.
   * @param finalInputHeight Input height of the spectrogram, seen from the final output layer.
   * @param waveFileDir Directory of wave files.
   * @param stftParam Parameters for short time Fourier transformation.
   * @param dpssParam Parameter for discrete prolate spheroidal sequences.
   * @param freqOffset Beginning of the frequency band.
   * @param freqLength Length of the frequency band.
   * @param spectrogramMeanSd Mean and SD of the spectrograms in training data. Used for input scaling.
   * @param inputHeightUpper Upper value of the combined input spectrogram.
   * @return
   * @throws IOException
   * @throws UnsupportedAudioFileException
   * @throws NotConvergedException
   */
  public static BatchData create(List<Sequence> sequence, MersenneTwister random, int numLowerLabel,
IntBinaryOperator silentLabelFunc, int finalInputHeight, Path waveFileDir, STFTParam stftParam, int dpssParam, int
freqOffset, int freqLength, double[] spectrogramMeanSd, int inputHeightUpper) throws IOException,
UnsupportedAudioFileException, NotConvergedException
  {
    int marginBegin=finalInputHeight/2;
    int marginEnd=finalInputHeight/2-1;
    HashMap<Sequence, WavePosition> wavePosition=Sequence.wavePositionMap(sequence, waveFileDir);
    for(WavePosition wp: wavePosition.values())
    {
      int waveBegin=wp.getPosition()-marginBegin*stftParam.getShiftLength();
      int waveEnd=wp.getEnd()+marginEnd*stftParam.getShiftLength();
      wp.setPosition(waveBegin);
      wp.setLength(waveEnd-waveBegin);
    }
    HashMap<Sequence, float[]> spectrogram=SoundUtils.spectrogram(wavePosition, stftParam, dpssParam, freqOffset,
freqLength);
    SoundUtils.whiteSpectrogram(spectrogram.values(), spectrogramMeanSd[0], spectrogramMeanSd[1]);

    ArrayList<float[]>
spectrogramList=sequence.stream().map(s->spectrogram.get(s)).collect(Collectors.toCollection(ArrayList::new));
    ArrayList<ArrayList<Integer>> packedSpectrogram=packedSpectrogramIndex(inputHeightUpper, spectrogramList,
random, freqLength);

    int packedHeight=packedSpectrogram.stream()
        .mapToInt(spec->spec.stream().mapToInt(s->spectrogram.get(sequence.get(s)).length/freqLength).sum())
        .max().getAsInt();

    return new BatchData(spectrogramList, packedHeight, packedSpectrogram, sequence, numLowerLabel,
silentLabelFunc, marginBegin, marginEnd, finalInputHeight, stftParam);
  }
}
```

DnnComputation.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package computation;

import java.io.IOException;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.function.IntBinaryOperator;

import javax.sound.sampled.UnsupportedAudioFileException;

import org.apache.commons.math3.random.MersenneTwister;

import computation.Sequence.LabelList;
import cudnn.Cuda;
import cudnn.CudaDriver;
import cudnn.CudaException;
import cudnn.Cudnn;
import cudnn.CudnnException;
import cudnn.FloatType;
import cudnn.layer.ConvLayer;
import cudnn.layer.SeqSoftmaxConvLayer;
import cudnn.learner.Adam;
import cudnn.network.SeqNetwork;
import jcuda.jcublas.JCublas;
import no.uib.cipr.matrix.NotConvergedException;
import utils.DnnUtils;

/**
 * This class performe training and recognitino of a DNN.
 * Hyper parameters and other configurations must be set in {@link HyperParam} and {@link Config}, respectively.
 * Trained parameters are handled in {@link Param}.
 *
 * @author koumura
 *
 */
public class DnnComputation
{
  /**
   * Parameter training.
   */
  public static Param training(ArrayList<Sequence> trainingSequence, LabelList labelList, MersenneTwister random,
HyperParam hyperParam, Config config) throws IOException, CudnnException, CudaException,
UnsupportedAudioFileException, NotConvergedException
  {
    BatchData data=BatchData.create(trainingSequence, random, hyperParam.numSubLabel, config.silentLabelFunc,
hyperParam.finalInputHeight, config.dirWaveFile, hyperParam.stftParam, hyperParam.dpssParam,
hyperParam.freqOffset, hyperParam.freqLength, config.spectrogramMeanSd, hyperParam.inputHeightUpper);
    data.pack(data.getPackedHeight(), hyperParam.freqLength, hyperParam.batchSizeUpper, labelList,
DnnUtils.filterSizeList525242());
    data.batch(data.getMaxBatchSize(), random, hyperParam.freqLength);

    Cudnn cudnn=new Cudnn(config.fileCudnnLibrary);
    CudaDriver driver=new CudaDriver();
    driver.load(config.fileCudaKernel.toAbsolutePath().toString());
    SeqNetwork network=DnnUtils.createNetwork(data.getDataLayerHeight(), hyperParam.freqLength,
config.softmaxSizeFunc.applyAsInt(labelList.size(), hyperParam.numSubLabel), data.getMaxBatchSize(), cudnn,
driver, hyperParam.localInputHeight, hyperParam.finalInputHeight, config.backwardAlogorithm,
hyperParam.numConvChannel, hyperParam.fullConnectionSize);

    network.cudaMalloc();
    if(config.verbose)
    {
      network.initError();
      data.setValidLabelSize(network);
    }

    DnnUtils.initParam(network.getLayer(), random);
    network.copyParamToDev();
```

```
    Adam learner=new Adam(network, FloatType.SINGLE, cudnn, driver);
    learner.init();
    for(int iter=0; iter<hyperParam.numIter; ++iter)
    {
      {
        int batch=random.nextInt(data.getNumBatch());
        int shift=random.nextInt(network.getLabelShiftUpperH());
        network.setDataF(data.getBatchData().get(batch));
        network.copyLabelToDevice(data.getBatchLabel().get(batch));
        network.setLabelShift(shift, 0);
        network.copyDataToDevice();
//      Cuda.deviceSynchronize();
        learner.iteration();
      }
      if(config.verbose)
      {
        double error=0;
        for(int batch=0; batch<data.getNumBatch(); ++batch)
        {
          network.setDataF(data.getBatchData().get(batch));
          network.copyLabelToDevice(data.getBatchLabel().get(batch));
          for(int shift=0; shift<network.getLabelShiftUpperH(); ++shift)
          {
            network.setLabelShift(shift, 0);
            network.copyDataToDevice();
//          Cuda.deviceSynchronize();
            network.compForward(driver, cudnn);
            error+=network.compError(driver);
          }
        }
        error/=data.getSumValidLabelSize();
        System.out.println("iter:"+iter+" Er:"+error);
      }
    }
    network.copyParamFromDev();
    network.destroy(cudnn);
    learner.destroy();
    cudnn.destroty();
    JCublas.cublasShutdown();  //memory leak will happen if cublasShutdown is called after driver.destroy()
    driver.destroy();

    return new Param(DnnUtils.copyParamFromLayer(network.getLayer()));
  }

  /**
   * Recognition, using trained parameter.
   * @param param trained paramter.
   */
 public static HashMap<Sequence, float[]> recognition(ArrayList<Sequence> validationSequence, LabelList labelList,
Param param, HyperParam hyperParam, Config config) throws IOException, UnsupportedAudioFileException,
NotConvergedException, CudaException, CudnnException
  {
    BatchData data=BatchData.create(validationSequence, null, hyperParam.numSubLabel, config.silentLabelFunc,
hyperParam.finalInputHeight, config.dirWaveFile, hyperParam.stftParam, hyperParam.dpssParam,
hyperParam.freqOffset, hyperParam.freqLength, config.spectrogramMeanSd, hyperParam.inputHeightUpper);
    data.pack(data.getPackedHeight(), hyperParam.freqLength, hyperParam.batchSizeUpper, labelList,
DnnUtils.filterSizeList525242());
    data.batch(data.getMaxBatchSize(), null, hyperParam.freqLength);

    Cudnn cudnn=new Cudnn(config.fileCudnnLibrary);
    CudaDriver driver=new CudaDriver();
    driver.load(config.fileCudaKernel.toAbsolutePath().toString());
    SeqNetwork network=DnnUtils.createNetwork(data.getDataLayerHeight(), hyperParam.freqLength,
config.softmaxSizeFunc.applyAsInt(labelList.size(), hyperParam.numSubLabel), data.getMaxBatchSize(), cudnn,
driver, hyperParam.localInputHeight, hyperParam.finalInputHeight, config.backwardAlogorithm,
hyperParam.numConvChannel, hyperParam.fullConnectionSize);
    SeqSoftmaxConvLayer outputLayer=(SeqSoftmaxConvLayer)network.getLayer().get(network.getLayer().size()-1);

    network.cudaMalloc();
    ArrayList<ArrayList<float[]>> networkOutput=new ArrayList<>(data.getNumBatch());
    DnnUtils.copyParamToLayer(param.getLayerParam(), network.getLayer());
    network.copyParamToDev();
    for(int batch=0; batch<data.getNumBatch(); ++batch)
    {
      networkOutput.add(new ArrayList<>(network.getLabelShiftUpperH()));
      network.setDataF(data.getBatchData().get(batch));
      for(int shift=0; shift<network.getLabelShiftUpperH(); ++shift)
      {
        network.setLabelShift(shift, 0);
        network.copyDataToDevice();
//      Cuda.deviceSynchronize();
        network.compForward(driver, cudnn);
        float[] o=Cuda.memcpyDeviceToFloatArray(outputLayer.getValueDev(),
outputLayer.sizeCHW()*data.getMaxBatchSize());
        networkOutput.get(batch).add(o);
      }
    }
    network.destroy(cudnn);
    cudnn.destroty();
    JCublas.cublasShutdown();  //memory leak will happen if cublasShutdown is called after driver.destroy()
    driver.destroy();

    HashMap<Sequence, float[]> sequenceOutput=new HashMap<>(validationSequence.size()*4/3);
    for(int s=0; s<data.getSpecLabelPackBegin().size(); ++s)
    {
      int pack=data.getSpecLabelPackBegin().get(s)[0];
      int batch=data.getPackBatchIndex().get(pack)[0];
      int batchIndex=data.getPackBatchIndex().get(pack)[1];
      int packBegin=data.getSpecLabelPackBegin().get(s)[1];

      int outputHeight=data.getSpectrogram().get(s).length/hyperParam.freqLength-data.specMarginSum();
      int outputWidth=outputLayer.getSingleSize();
      float[] array=new float[outputHeight*outputWidth];
      sequenceOutput.put(validationSequence.get(s), array);
      int arrayIndex=0;
      for(int sy=0; sy<outputHeight; ++sy)
      {
        int py=packBegin+sy;
        int shift=py%network.getLabelShiftUpperH();
        int oy=py/network.getLabelShiftUpperH();
        for(int oc=0; oc<outputLayer.getSingleSize(); ++oc)
        {
          int index=oy+(oc+batchIndex*outputLayer.getSingleSize())*outputLayer.getHeight();
          float value=networkOutput.get(batch).get(shift)[index];
          array[arrayIndex++]=value;
        }
      }
    }
    return sequenceOutput;
  }

  public static class Config
  {
    private boolean verbose;
    private IntBinaryOperator silentLabelFunc, softmaxSizeFunc;
    private double[] spectrogramMeanSd;
    private Path fileCudaKernel, fileCudnnLibrary, dirWaveFile;
    private ConvLayer.BackwardAlgorithm backwardAlogorithm;

    public Config(boolean verbose, IntBinaryOperator silentLabelFunc, IntBinaryOperator softmaxSizeFunc,
        double[] spectrogramMeanSd, Path fileCudaKernel, Path fileCudnnLibrary, Path dirWaveFile,
ConvLayer.BackwardAlgorithm backwardAlogorithm) {
      this.verbose = verbose;
      this.silentLabelFunc = silentLabelFunc;
      this.softmaxSizeFunc = softmaxSizeFunc;
      this.spectrogramMeanSd = spectrogramMeanSd;
      this.fileCudaKernel = fileCudaKernel;
      this.fileCudnnLibrary = fileCudnnLibrary;
      this.dirWaveFile = dirWaveFile;
      this.backwardAlogorithm=backwardAlogorithm;
    }
  }

  public static class HyperParam
  {
    private STFTParam stftParam;
    private int dpssParam, localInputHeight, finalInputHeight, numSubLabel, freqOffset, freqLength,
inputHeightUpper, batchSizeUpper, numIter, numConvChannel, fullConnectionSize;
```

```
    public HyperParam(STFTParam stftParam, int dpssParam, int localInputHeight, int finalInputHeight, int
numSubLabel,
        int freqOffset, int freqLength, int inputHeightUpper, int batchSizeUpper, int numIter, int numConvChannel,
int fullConnectionSize) {
      this.stftParam = stftParam;
      this.dpssParam = dpssParam;
      this.localInputHeight = localInputHeight;
      this.finalInputHeight = finalInputHeight;
      this.numSubLabel = numSubLabel;
      this.freqOffset = freqOffset;
      this.freqLength = freqLength;
      this.inputHeightUpper = inputHeightUpper;
      this.batchSizeUpper = batchSizeUpper;
      this.numIter = numIter;
      this.numConvChannel=numConvChannel;
      this.fullConnectionSize=fullConnectionSize;
    }
  }

  public static class Param
  {
    private ArrayList<float[]> layerParam;

    public Param(ArrayList<float[]> layerParam)
    {
      this.layerParam = layerParam;
    }

    public ArrayList<float[]> getLayerParam() {
      return layerParam;
    }

    public void save(Path file) throws IOException
    {
      DnnUtils.saveParam(layerParam, file);
    }

    public static Param load(Path file) throws IOException
    {
      return new Param(DnnUtils.loadParam(file));
    }
  }
}
```

HmmComputation.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package computation;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;
import java.util.stream.Collectors;

import computation.Sequence.LabelList;
import computation.Sequence.Note;
import computation.ViterbiSequencer.SecondOrderLowerTransition;
import computation.ViterbiSequencer.SecondOrderTransition;
import utils.ArrayUtils;
import utils.CollectionUtils;
import utils.Executor;
import utils.MathUtils;

/**
 * This class performs computation of an HMM.
 * Hyper parameters and other configurations must be set in {@link HyperParam} and {@link Config}, respectively.
 *
 * @author koumura
 *
 */
public class HmmComputation
{
  private static double[][][] transitionProbability(List<Sequence> sequence, LabelList labelList)
  {
    int numLabel=labelList.size();
    int[][][] count=new int[numLabel][numLabel][numLabel];
    for(Sequence seq: sequence)
    {
      ArrayList<Note> noteList=seq.getNote();
      for(int i=2; i<noteList.size(); ++i)
++count[labelList.indexOf(noteList.get(i-2).getLabel())][labelList.indexOf(noteList.get(i-1).getLabel())][labe
lList.indexOf(noteList.get(i).getLabel())];
    }
    double[][][] prob=new double[numLabel][numLabel][numLabel];
    for(int l0=0; l0<numLabel; ++l0) for(int l1=0; l1<numLabel; ++l1)
    {
      int sum=MathUtils.sum(count[l0][l1]);
      if(sum>0) for(int l2=0; l2<numLabel; ++l2) prob[l0][l1][l2]=(double)count[l0][l1][l2]/sum;
    }
    return prob;
  }

  private static void smoothTransitionProbability(double[][][] prob, double smoothing)
  {
    for(int l0=0; l0<prob.length; ++l0) for(int l1=0; l1<prob[l0].length; ++l1)
    {
      for(int l2=0; l2<prob[l0][l1].length; ++l2) prob[l0][l1][l2]+=smoothing;
      MathUtils.divideBySum(prob[l0][l1]);
    }
  }

  private static double[] fullSpecLabelFrequency(List<Sequence> sequence, int numLowerLabel, LabelList labelList,
STFTParam stftParam)
  {
    int numUpperLabel=labelList.size();
    int[] count=new int[numUpperLabel*numLowerLabel+1];
    for(Sequence seq: sequence)
    {
      ArrayList<int[]> specLabelInterval=seq.getNote().stream()
          .map(n->{
            int begin=stftParam.spectrogramPosition(n.getPosition());
            int end=stftParam.spectrogramPosition(n.end());
            return new int[]{begin, end-begin, labelList.indexOf(n.getLabel())};
          }).collect(CollectionUtils.arrayListCollector());

      for(int[] li: specLabelInterval) for(int lower=0; lower<numLowerLabel; ++lower)
      {
        int begin=MathUtils.subIntervalBegin(li[1], numLowerLabel, lower);
        int end=MathUtils.subIntervalBegin(li[1], numLowerLabel, lower+1);
        int fullLabel=li[2]*numLowerLabel+lower;
        count[fullLabel]+=end-begin;
      }
      for(int i=0; i<specLabelInterval.size()-1; ++i)
      {
        count[count.length-1]+=specLabelInterval.get(i+1)[0]-ArrayUtils.sum01(specLabelInterval.get(i));
```

```
      }
      count[count.length-1]+=specLabelInterval.get(0)[0];
      int spectrogramLength=stftParam.spectrogramLength(seq.getLength());

  count[count.length-1]+=spectrogramLength-ArrayUtils.sum01(specLabelInterval.get(specLabelInterval.size()-1))
;
    }
    return MathUtils.divideBySum(count);
  }

  private static double[] upperLabelFrequency(List<Sequence> specLabelInterval, LabelList labelList)
  {
    int numUpperLabel=labelList.size();
    int[] count=new int[numUpperLabel];
    for(Sequence seq: specLabelInterval)
    {
      for(Note li: seq.getNote())
      {
        ++count[labelList.indexOf(li.getLabel())];
      }
    }
    return MathUtils.divideBySum(count);
  }

  private static void posteriorToObservationProb(ArrayList<double[]> posterior, double[] fullLabelFrequency)
  {
    for(double[] p: posterior) for(int la=0; la<p.length; ++la) p[la]/=fullLabelFrequency[la];
  }

  /**
   * Computes trainsition probability from training data.
   * @return transition probability
   */
  public static double[][][] transitionProbability(List<Sequence> trainingSequence, LabelList labelList,
HyperParam hyperParam)
  {
    double[][][] transitionProbability=transitionProbability(trainingSequence, labelList);
    if(hyperParam.smoothingConstant>0) smoothTransitionProbability(transitionProbability,
hyperParam.smoothingConstant);
    return transitionProbability;
  }

  /**
   * Converts continuous outputs of a DNN to segmented observation probabilities for HMM. Used in the BD -&gt; LC
-&gt; GS arrangement.
   * @param posteriorProb Outputs of a DNN
   * @return Observation probability
   */
  public static HashMap<Sequence, ArrayList<double[]>> segmentedPosteriorToObservationProb(HashMap<Sequence,
ArrayList<double[]>> posteriorProb, List<Sequence> trainingSequence, LabelList labelList, HyperParam hyperParam)
  {
    HashMap<Sequence, ArrayList<double[]>> observationProb=posteriorProb;
    if(hyperParam.posteriorToObservationProb)
    {
      double[] fullLabelFreq=upperLabelFrequency(trainingSequence, labelList);
      for(ArrayList<double[]> op: observationProb.values()) posteriorToObservationProb(op, fullLabelFreq);
    }
    return observationProb;
  }

  /**
   * Performs global sequencing with a HMM. Used in the BD -&gt; LC -&gt; GS arrangement.
   * @return Recognized elements in each input sequence.
   */
  public static HashMap<Sequence, ArrayList<Note>> globalSequencing(HashMap<Sequence, ArrayList<double[]>>
observationProb, double[][][] transitionProb, LabelList labelList, HashMap<Sequence, ArrayList<int[]>>
soundInterval, HyperParam hyperParam, Config config) throws InterruptedException, ExecutionException
  {
    ArrayList<Sequence>
sequence=observationProb.keySet().stream().collect(CollectionUtils.arrayListCollector());
    SecondOrderTransition transition=new SecondOrderTransition(transitionProb);
    ArrayList<UpperSequencerJob> job=new ArrayList<>(config.executor.getNumThread());
    for(int th=0; th<config.executor.getNumThread(); ++th)
    {
      int subIntervalBegin=MathUtils.subIntervalBegin(observationProb.size(), config.executor.getNumThread(),
th);
      int subIntervalEnd=MathUtils.subIntervalBegin(observationProb.size(), config.executor.getNumThread(),
th+1);
      HashMap<Sequence, ArrayList<double[]>> op=new HashMap<>((subIntervalEnd-subIntervalBegin)*4/3);
      for(int i=subIntervalBegin; i<subIntervalEnd; ++i) op.put(sequence.get(i),
observationProb.get(sequence.get(i)));
      job.add(new UpperSequencerJob(transition, op));
    }

    List<Future<HashMap<Sequence, int[]>>> future=config.executor.get().invokeAll(job);

    HashMap<Sequence, ArrayList<Note>> viterbiLabelInterval=new HashMap<>(observationProb.size()*4/3);
    for(Future<HashMap<Sequence, int[]>> f: future)
    {
      HashMap<Sequence, int[]> answerMap=f.get();
      for(Sequence seq: answerMap.keySet())
      {
        int[] answer=answerMap.get(seq);
        ArrayList<Note> answerInterval=new ArrayList<Note>(answer.length);
        viterbiLabelInterval.put(seq, answerInterval);
        for(int i=0; i<soundInterval.get(seq).size(); ++i)
        {
          int[] si=soundInterval.get(seq).get(i);
          int start=hyperParam.stftParam.wavePosition(si[0]);
          int end=hyperParam.stftParam.wavePosition(si[0]+si[1]);
          String label=labelList.get(answer[i]);
          answerInterval.add(new Note(start, end-start, label));
        }
      }
    }
    return viterbiLabelInterval;
  }

  /**
   * Converts continuous outputs of a DNN into continuous observation probabilities for HMM. Used in the LC -&gt;
BD & GS and LC & GS -&gt; BD & GS arrangements.
   * @param posteriorProb Outputs of a DNN
   * @return Observation probability
   */
  public static HashMap<Sequence, ArrayList<double[]>> continuousPosteriorToObservationProb(HashMap<Sequence,
float[]> posteriorProb, List<Sequence> trainingSequence, LabelList labelList, HyperParam hyperParam, Config config)
  {
    int posteriorWidth=config.posteriorWidth;
    HashMap<Sequence, ArrayList<double[]>> observationProb=new HashMap<>(posteriorProb.size()*4/3);
    for(Sequence seq: posteriorProb.keySet())
    {
      int height=posteriorProb.get(seq).length/posteriorWidth;
      ArrayList<double[]> prob=new ArrayList<>(height);
      observationProb.put(seq, prob);
      int i=0;
      for(int h=0; h<height; ++h)
      {
        double[] array=new double[posteriorWidth];
        prob.add(array);
        for(int w=0; w<posteriorWidth; ++w) array[w]=posteriorProb.get(seq)[i++];
      }
    }
    if(hyperParam.posteriorToObservationProb)
    {
      double[] fullLabelFreq=fullSpecLabelFrequency(trainingSequence, hyperParam.numSubLabel, labelList,
hyperParam.stftParam);
      for(ArrayList<double[]> op: observationProb.values()) posteriorToObservationProb(op, fullLabelFreq);
    }
    return observationProb;
  }

  /**
   * Performs global sequencing with a HMM. Used in the LC -&gt; BD & GS and LC & GS -&gt; BD & GS arrangements.
   * @return Recognized elements in each input sequence.
   */
  public static HashMap<Sequence, ArrayList<Note>> globalSequencingWithBoundaryDetection(HashMap<Sequence,
ArrayList<double[]>> observationProb, double[][][] transitionProb, LabelList labelList, HyperParam hyperParam,
Config config) throws InterruptedException, ExecutionException
```

```
  {
    ArrayList<Sequence>
sequence=observationProb.keySet().stream().collect(CollectionUtils.arrayListCollector());
    SecondOrderLowerTransition transition=new SecondOrderLowerTransition(transitionProb, labelList.size(),
hyperParam.numSubLabel);
    ArrayList<SequencerJob> job=new ArrayList<>(config.executor.getNumThread());
    for(int th=0; th<config.executor.getNumThread(); ++th)
    {
      int subIntervalBegin=MathUtils.subIntervalBegin(observationProb.size(), config.executor.getNumThread(),
th);
      int subIntervalEnd=MathUtils.subIntervalBegin(observationProb.size(), config.executor.getNumThread(),
th+1);
      HashMap<Sequence, ArrayList<double[]>> op=new HashMap<>((subIntervalEnd-subIntervalBegin)*4/3);
      for(int i=subIntervalBegin; i<subIntervalEnd; ++i) op.put(sequence.get(i),
observationProb.get(sequence.get(i)));
      job.add(new SequencerJob(transition, op));
    }

    List<Future<Object>> future=config.executor.get().invokeAll(job);
    for(Future<Object> f: future) f.get();

    HashMap<Sequence, ArrayList<Note>> viterbiLabelInterval=new HashMap<>(observationProb.size()*4/3);
    for(SequencerJob j: job) for(Sequence seq: j.observationProb.keySet())
    {
      ArrayList<Note> noteList=viterbiLabelInterval(j.answer.get(seq), labelList.size(),
hyperParam.numSubLabel).stream()
          .map(li->{
            int start=hyperParam.stftParam.wavePosition(li[0]);
            int end=hyperParam.stftParam.wavePosition(li[0]+li[1]);
            String label=labelList.get(li[2]);
            return new Note(start, end-start, label);
          }).collect(Collectors.toCollection(ArrayList::new));
      viterbiLabelInterval.put(seq, noteList);
    }
    return viterbiLabelInterval;
  }

  public static class Config
  {
    private Executor executor;
    private int posteriorWidth;

    public Config(Executor executor, int posteriorWidth) {
      this.executor = executor;
      this.posteriorWidth=posteriorWidth;
    }
  }

  public static class HyperParam
  {
    private double smoothingConstant;
    private boolean posteriorToObservationProb;
    private int numSubLabel;
    private STFTParam stftParam;

    public HyperParam(double smoothingConstant, boolean posteriorToObservationProb, int numSubLabel,
        STFTParam stftParam) {
      this.smoothingConstant = smoothingConstant;
      this.posteriorToObservationProb = posteriorToObservationProb;
      this.numSubLabel = numSubLabel;
      this.stftParam = stftParam;
    }
  }

  private static class SequencerJob implements Callable<Object>
  {
    private SecondOrderLowerTransition transition;
    private ViterbiSequencer sequencer;
    private HashMap<Sequence, ArrayList<double[]>> observationProb;
    private HashMap<Sequence, ArrayList<int[]>> answer;

    private SequencerJob(SecondOrderLowerTransition transition, HashMap<Sequence, ArrayList<double[]>>
observationProb)
    {
      this.observationProb=observationProb;
      this.transition=transition;
      sequencer=new ViterbiSequencer(transition);
    }

    @Override
    public Object call()
    {
      answer=new HashMap<>(observationProb.size()*4/3);
      for(Sequence seq: observationProb.keySet())
      {
        ArrayList<double[]> op=observationProb.get(seq);
        ArrayList<Integer> a=sequencer.compLabelSequence(op);
        ArrayList<int[]> answerArray=new ArrayList<>(a.size());
        answer.put(seq, answerArray);
        for(int t=0; t<a.size(); ++t)
        {
          answerArray.add(new int[]{transition.upperLabel(a.get(t)), transition.lowerLabel(a.get(t))});
        }
      }
      return null;
    }
  }

  private static class UpperSequencerJob implements Callable<HashMap<Sequence, int[]>>
  {
    private ViterbiSequencer sequencer;
    private HashMap<Sequence, ArrayList<double[]>> observationProb;

    private UpperSequencerJob(SecondOrderTransition transition, HashMap<Sequence, ArrayList<double[]>>
observationProb)
    {
      this.observationProb=observationProb;
      sequencer=new ViterbiSequencer(transition);
    }

    @Override
    public HashMap<Sequence, int[]> call()
    {
      HashMap<Sequence, int[]> answer=new HashMap<Sequence, int[]>(observationProb.size()*4/3);
      for(Sequence seq: observationProb.keySet())
      {
        ArrayList<Integer> a=sequencer.compLabelSequence(observationProb.get(seq));
        int[] array=a.stream().mapToInt(x->x).toArray();
        answer.put(seq, array);
      }
      return answer;
    }
  }

  private static ArrayList<int[]> viterbiLabelInterval(ArrayList<int[]> viterbiSequence, int numUpperLabel, int
numLowerLabel)
  {
    ArrayList<int[]> classified=new ArrayList<int[]>();
    for(int x=0; x<viterbiSequence.size(); ++x)
    {
      int[] vp=viterbiSequence.get(x);
      if(vp[0]<numUpperLabel&&vp[1]==0)
      {
        if(x>0 && viterbiSequence.get(x-1)[1]==numLowerLabel-1) classified.get(classified.size()-1)[1]=(x);
        if(classified.size()==0 || classified.get(classified.size()-1)[1]!=-1) classified.add(new int[]{x, -1,
vp[0]});
      }
      else if((vp[0]==numUpperLabel || vp[1]==numLowerLabel) &&
classified.size()>0&&classified.get(classified.size()-1)[1]==-1) classified.get(classified.size()-1)[1]=(x);
    }
    if(classified.size()>0&&classified.get(classified.size()-1)[1]==-1)
classified.get(classified.size()-1)[1]=(viterbiSequence.size());
    for(int[] li: classified) li[1]=(li[1]-li[0]);

    return classified;
  }
}
```

Sequence.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package computation;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.stream.Collectors;

import org.apache.commons.math3.random.MersenneTwister;
import org.w3c.dom.Element;
import org.w3c.dom.Node;

import utils.CollectionUtils;
import utils.Pair;
import utils.RandomUtils;
import utils.XmlUtils;

/**
 * This class handles annotations of sound data.
 * @author koumura
 *
 */
public class Sequence
{
  private String waveFileName;
  private int position, length;
  private ArrayList<Note> note;

  private Sequence(String waveFileName, int position, int length, ArrayList<Note> note)
  {
    this.waveFileName = waveFileName;
    this.position = position;
    this.length = length;
    this.note = note;
  }

  public String getWaveFileName() {
    return waveFileName;
  }

  public int getPosition() {
    return position;
  }

  public int getLength() {
    return length;
  }

  public ArrayList<Note> getNote() {
    return note;
  }

  public static ArrayList<Sequence> readXml(Path file) throws IOException
  {
    Node rootChild=XmlUtils.parse(file).getFirstChild().getFirstChild();
    int numSequence=XmlUtils.nodeInt(rootChild);
    ArrayList<Sequence> sequence=new ArrayList<Sequence>(numSequence);
    for(int s=0; s<numSequence; ++s)
    {
      rootChild=rootChild.getNextSibling();
      Node seqChild=rootChild.getFirstChild();
      String waveFileName=XmlUtils.nodeText(seqChild);
      seqChild=seqChild.getNextSibling();
      int position=XmlUtils.nodeInt(seqChild);
      seqChild=seqChild.getNextSibling();
      int length=XmlUtils.nodeInt(seqChild);
      seqChild=seqChild.getNextSibling();
      int numNote=XmlUtils.nodeInt(seqChild);
      ArrayList<Note> note=new ArrayList<>(numNote);
      sequence.add(new Sequence(waveFileName, position, length, note));
      for(int n=0; n<numNote; ++n)
      {
        seqChild=seqChild.getNextSibling();
        Node noteChild=seqChild.getFirstChild();
        int pos=XmlUtils.nodeInt(noteChild);
        noteChild=noteChild.getNextSibling();
        int len=XmlUtils.nodeInt(noteChild);
        noteChild=noteChild.getNextSibling();
        String label=XmlUtils.nodeText(noteChild);
        note.add(new Note(pos, len, label));
      }
    }
    return sequence;
  }

  @Override
  public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + length;
    result = prime * result + position;
    result = prime * result + ((waveFileName == null) ? 0 : waveFileName.hashCode());
    return result;
  }

  @Override
  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if (obj == null)
      return false;
    if (getClass() != obj.getClass())
      return false;
    Sequence other = (Sequence) obj;
    if (length != other.length)
      return false;
    if (position != other.position)
      return false;
    if (waveFileName == null) {
      if (other.waveFileName != null)
        return false;
    } else if (!waveFileName.equals(other.waveFileName))
      return false;
    return true;
  }

  /**
   * Writes an XML file.
   */
  public static void writeOutputSequence(HashMap<Sequence, ArrayList<Note>> outputSequence, Path file) throws
IOException
  {
```

```java
  Element rootEl=XmlUtils.rootElement("SequenceNoteInterval");
  XmlUtils.addChild(rootEl, "NumSequence", outputSequence.size());
  List<Sequence> sequenceList=outputSequence.keySet().stream().collect(Collectors.toList());
  sequenceList.sort((o1, o2)->{
    int c=o1.waveFileName.compareTo(o2.waveFileName);
    if(c!=0) return c;
    c=o1.position-o2.position;
    if(c!=0) return c;
    c=o1.length-o2.length;
    return c;
  });
  for(Sequence seq: outputSequence.keySet())
  {
    Element seqEl=XmlUtils.addChild(rootEl, "Sequence", null);
    XmlUtils.addChild(seqEl, "WaveFileName", seq.getWaveFileName());
    XmlUtils.addChild(seqEl, "Position", seq.getPosition());
    XmlUtils.addChild(seqEl, "Length", seq.getLength());
    XmlUtils.addChild(seqEl, "NumNote", outputSequence.get(seq).size());
    for(Note note: outputSequence.get(seq))
    {
      Element noteEl=XmlUtils.addChild(seqEl, "Note", null);
      XmlUtils.addChild(noteEl, "Position", note.getPosition());
      XmlUtils.addChild(noteEl, "Length", note.getLength());
      XmlUtils.addChild(noteEl, "Label", note.getLabel());
    }
  }
  XmlUtils.write(rootEl.getOwnerDocument(), file);
}

public int byteSize()
{
  return Integer.BYTES*4+waveFileName.getBytes().length;
}

/**
 * Serializes itself into a byte array.
 */
public void serialize(ByteBuffer buf)
{
  byte[] b=waveFileName.getBytes();
  buf.putInt(b.length);
  buf.put(b);
  buf.putInt(position);
  buf.putInt(length);
}

/**
 * Deserializes from a byte array.
 */
public static Sequence deserialize(ByteBuffer buf, HashMap<Sequence, Sequence> sequenceMap)
{
  int byteLen=buf.getInt();
  byte[] b=new byte[byteLen];
  buf.get(b);
  int position=buf.getInt();
  int length=buf.getInt();
  Sequence seq=sequenceMap.get(new Sequence(new String(b), position, length, null));
  if(seq==null) System.err.println("Sequence not contained in given set.");
  return seq;
}

/**
 * Sound elements in a song.
 * @author koumura
 *
 */
public static class Note
{
  private int position, length;
  private String label;

  public Note(int position, int length, String label)
  {
    this.position = position;
```

```java
    this.length = length;
    this.label = label;
  }

  public int getPosition() {
    return position;
  }

  public int getLength() {
    return length;
  }

  public String getLabel() {
    return label;
  }

  public int end(){return position+length;}
}

public static class WavePosition
{
  private Path waveFile;
  private int position, length;

  public WavePosition(Path waveFile, int position, int length)
  {
    this.waveFile = waveFile;
    this.position = position;
    this.length = length;
  }

  public Path getWaveFile() {
    return waveFile;
  }

  public void setWaveFile(Path waveFile) {
    this.waveFile = waveFile;
  }

  public int getPosition() {
    return position;
  }

  public void setPosition(int position) {
    this.position = position;
  }

  public int getLength() {
    return length;
  }

  public void setLength(int length) {
    this.length = length;
  }

  public int getEnd()
  {
    return position+length;
  }
}

public static HashMap<Sequence, WavePosition> wavePositionMap(Collection<Sequence> sequence, Path waveFileDir)
{
  HashMap<Sequence, WavePosition> wavePositionMap=new HashMap<>(sequence.size()*4/3);
  for(Sequence seq: sequence)
  {
    WavePosition wp=new WavePosition(waveFileDir.resolve(seq.waveFileName), seq.getPosition(),
seq.getLength());
    wavePositionMap.put(seq, wp);
  }
  return wavePositionMap;
}

/**
```

```
   * Extracts sequences with limited total length.
   * @param sequenceLength0Upper
   * @param random
   * @param sequence
   * @return
   */
  public static Pair<ArrayList<Sequence>, ArrayList<Sequence>> extract(int sequenceLength0Upper, MersenneTwister
random, List<Sequence> sequence)
  {
    ArrayList<Sequence> all=sequence.stream().collect(CollectionUtils.arrayListCollector());
    all=RandomUtils.permutation(all, random);
    int length=0;
    int num0=0;
    while(num0<all.size())
    {
      int len=all.get(num0).getLength();
      if(length+len>=sequenceLength0Upper) break;
      ++num0;
      length+=len;
    }
    ArrayList<Sequence> sequence0=all.subList(0, num0).stream().collect(CollectionUtils.arrayListCollector());
    ArrayList<Sequence> sequence1=all.subList(num0,
all.size()).stream().collect(CollectionUtils.arrayListCollector());
    return new Pair<>(sequence0, sequence1);
  }

  /**
   * This class handles conversion of a label string and the corresponding label index.
   * @author koumura
   *
   */
  public static class LabelList
  {
    private ArrayList<String> list;
    private HashMap<String, Integer> index;

    private LabelList(ArrayList<String> list)
    {
      this.list = list;
      index=new HashMap<>(list.size()*4/3);
      for(int li=0; li<list.size(); ++li) index.put(list.get(li), li);
    }

    public String get(int index){return list.get(index);}

    public int indexOf(String label){return index.get(label);}

    public int size(){return list.size();}

    public static LabelList create(Collection<Sequence> sequence)
    {
      ArrayList<String> labelList=sequence.stream()
          .flatMap(s->s.getNote().stream())
          .map(n->n.getLabel())
          .collect(Collectors.toSet())
          .stream()
          .collect(Collectors.toCollection(ArrayList::new));
      Collections.sort(labelList);
      return new LabelList(labelList);
    }
  }
}
```

STFTParam.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package computation;

import utils.MathUtils;

/**
 * This class handles parameters for short time Fourier transformation.
 * @author koumura
 *
 */
public class STFTParam
{
  private int fftLength, shiftLength;

  public STFTParam(int fftLength, int shiftLength)
  {
    this.fftLength = fftLength;
    this.shiftLength = shiftLength;
  }

  public int getFftLength() {
    return fftLength;
  }

  public int getShiftLength() {
    return shiftLength;
  }

  /**
   * Converts a length of a wave data into length of the spectrogram.
   * @param waveLength Length of a wave data.
   * @return Length of the spectrogram.
   */
  public int spectrogramLength(int waveLength)
  {
    return MathUtils.ceil(waveLength-(fftLength-shiftLength), shiftLength);
  }

  /**
   * Converts a position in a spectrogram into the position in the wave data.
   * @param spectrogramPosition Position in a spectrogram.
   * @return Position in the wave data.
   */
  public int wavePosition(int spectrogramPosition)
  {
    return spectrogramPosition*shiftLength+fftLength/2;
  }

  /**
   * Converts a position in a wave data into the position in the spectrogram.
   * @param wavePosition Position in a wave data.
   * @return Position in the spectrogram.
   */
  public int spectrogramPosition(int wavePosition)
  {
    return (wavePosition-fftLength/2)/shiftLength;
  }

  /**
   * Computes frequency width (Hz) in a single cell.
   * @param samplingRate
   * @return frequency width (Hz).
   */
  public double unitFrequency(int samplingRate)
```

```
  {
    return unitFrequency(samplingRate, fftLength);
  }

  /**
   * Computes frequency width (Hz) in a single cell.
   * @param samplingRate
   * @param fftLength
   * @return frequency width (Hz).
   */
  public static double unitFrequency(int samplingRate, int fftLength)
  {
    return (double)samplingRate/fftLength;
  }
}
```

Thresholding.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package computation;

import java.io.IOException;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;
import java.util.stream.Collectors;

import javax.sound.sampled.UnsupportedAudioFileException;

import org.w3c.dom.Element;
import org.w3c.dom.Node;

import computation.Sequence.Note;
import errorcomputation.Matching;
import no.uib.cipr.matrix.NotConvergedException;
import utils.CollectionUtils;
import utils.Executor;
import utils.Pair;
import utils.SoundUtils;
import utils.XmlUtils;

/**
 * This class performs boundary detection by thresholding of sound amplitude.
 * Used in the BD -&gt; LC -&gt; GS arrangement.
 * Hyper parameters and other configurations must be set in {@link HyperParameter} and {@link Config}, respectively.
 * Thresholds are stored in {@link Parameter}.
 *
 * @author koumura
 *
 */
public class Thresholding
{
  private static LinkedList<int[]> soundInterval(double[] amplitude, double threshold)
  {
    LinkedList<int[]> soundInterval=new LinkedList<int[]>();
    for(int t=0; t<amplitude.length; ++t)
    {
      if(amplitude[t]>=threshold && (soundInterval.size()==0 || soundInterval.getLast()[1]!=-1))
soundInterval.add(new int[]{t, -1, 0});
      else if(amplitude[t]<threshold && soundInterval.size()>0 && soundInterval.getLast()[1]==-1)
soundInterval.getLast()[1]=(t);
    }
    if(soundInterval.size()>0&&soundInterval.getLast()[1]==-1) soundInterval.getLast()[1]=(amplitude.length);

    for(int[] si: soundInterval) si[1]-=si[0];

    return soundInterval;
  }

  private static void removeShortGap(ArrayList<int[]> soundInterval, int gapLengthLower)
  {
    if(gapLengthLower>0)
    {
      int remmainingTail=0;
      LinkedList<int[]> remove=new LinkedList<>();
      for(int i=0; i<soundInterval.size()-1; ++i)
      {
        if(soundInterval.get(i+1)[0]-(soundInterval.get(i)[0]+soundInterval.get(i)[1])<=gapLengthLower)
        {
          remove.add(soundInterval.get(i+1));

  soundInterval.get(remmainingTail)[1]=(soundInterval.get(i+1)[0]+soundInterval.get(i+1)[1]-soundInterval.get(
remmainingTail)[0]);
        }
        else remmainingTail=i+1;
      }
      soundInterval.removeAll(remove);
    }
  }

  private static void removeShortNote(ArrayList<int[]> soundInterval, int noteLengthLower)
  {
    if(noteLengthLower>0)
    {
      soundInterval.removeIf(si -> si[1]<=noteLengthLower);
    }
  }

  private static double[] spectrogramAmplitude(float[] spec, int freqLength) throws IOException
  {
    double[] amp=new double[spec.length/freqLength];
    for(int t=0; t<amp.length; ++t)
    {
      for(int f=0; f<freqLength; ++f) amp[t]+=spec[t*freqLength+f];
      amp[t]/=freqLength;
    }
    return amp;
  }
  private static HashMap<Sequence, double[]> spectrogramAmplitude(Collection<Sequence> sequence, HyperParameter
hyperParameter, Config config) throws IOException, UnsupportedAudioFileException, NotConvergedException
  {
    HashMap<Sequence, float[]> spectrogram=SoundUtils.spectrogram(Sequence.wavePositionMap(sequence,
config.waveFileDir), hyperParameter.stftParam, hyperParameter.dpssParam, hyperParameter.freqOffset,
hyperParameter.freqLength);
    HashMap<Sequence, double[]> spectrogramAmplitude=new HashMap<>(sequence.size()*4/3);
    for(Sequence seq: sequence) spectrogramAmplitude.put(seq, spectrogramAmplitude(spectrogram.get(seq),
hyperParameter.freqLength));
    return spectrogramAmplitude;
  }

  private static LinkedList<Pair<Sequence, Double>> specIndexAmplitude(HashMap<Sequence, double[]>
spectrogramAmplitude)
```

```
  {
    LinkedList<Pair<Sequence, Double>> ampAll=new LinkedList<>();
    for(Sequence seq: spectrogramAmplitude.keySet()) for(double a: spectrogramAmplitude.get(seq)) ampAll.add(new
Pair<>(seq, a));
    return ampAll;
  }

  /**
   * Compute the best thresholds in the training data.
   */
  public static Parameter train(Collection<Sequence> sequence, HyperParameter hyperParameter, Config config) throws
InterruptedException, ExecutionException, UnsupportedAudioFileException, IOException, NotConvergedException
  {
    HashMap<Sequence, double[]> spectrogramAmplitude=spectrogramAmplitude(sequence, hyperParameter, config);
    ArrayList<Pair<Sequence, Double>>
specIndexAmp=specIndexAmplitude(spectrogramAmplitude).stream().collect(CollectionUtils.arrayListCollector());
    specIndexAmp.sort((o1, o2)->Double.compare(o1.get1(), o2.get1()));

    HashMap<Sequence, ArrayList<int[]>> soundInterval=new HashMap<>(sequence.size()*4/3);
    for(Sequence seq: sequence) soundInterval.put(seq, soundInterval(spectrogramAmplitude.get(seq),
specIndexAmp.get(0).get1()).stream().collect(CollectionUtils.arrayListCollector()));

    HashMap<Sequence, ArrayList<int[]>> correctSoundSpecInterval=new HashMap<>(sequence.size()*4/3);
    for(Sequence seq: sequence)
    {
      ArrayList<int[]> correct=new ArrayList<>(seq.getNote().size());
      correctSoundSpecInterval.put(seq, correct);
      for(Note note: seq.getNote())
      {
        int begin=hyperParameter.stftParam.spectrogramPosition(note.getPosition());
        int end=hyperParameter.stftParam.spectrogramPosition(note.getPosition()+note.getLength());
        correct.add(new int[]{begin, end-begin, 0});
      }
    }

    HashMap<Sequence, Integer> sequenceSpecLength=new HashMap<>(sequence.size()*4/3);
    for(Sequence seq: sequence) sequenceSpecLength.put(seq,
hyperParameter.stftParam.spectrogramLength(seq.getLength()));

    HashMap<Sequence, int[][]> currentScore=new HashMap<>(sequence.size()*4/3);
    for(Sequence seq: sequence)
    {
      int[][] score=new int[hyperParameter.gapLowerUpper][hyperParameter.noteLowerUpper];
      currentScore.put(seq, score);

      ArrayList<Integer> gapLengthList=new ArrayList<Integer>(soundInterval.get(seq).size()-1);
      for(int i=0; i<soundInterval.get(seq).size()-1; ++i)
gapLengthList.add(soundInterval.get(seq).get(i+1)[0]-(soundInterval.get(seq).get(i)[0]+soundInterval.get(seq).
get(i)[1]));

  gapLengthList=gapLengthList.stream().collect(Collectors.toSet()).stream().filter(gl->gl<hyperParameter.gapLo
werUpper).collect(CollectionUtils.arrayListCollector());
      gapLengthList.add(hyperParameter.gapLowerUpper);
      gapLengthList.add(0);
      Collections.sort(gapLengthList);
      for(int gi=0; gi<gapLengthList.size()-1; ++gi)
      {
        ArrayList<int[]> shortGapRemoved=CollectionUtils.deepCopy(soundInterval.get(seq));
        removeShortGap(shortGapRemoved, gapLengthList.get(gi));

        ArrayList<Integer> noteLengthList=shortGapRemoved.stream()
          .map(si->si[1])
          .filter(nl->nl<hyperParameter.noteLowerUpper)
          .collect(Collectors.toSet()).stream()
          .collect(CollectionUtils.arrayListCollector());
        noteLengthList.add(0);
        noteLengthList.add(hyperParameter.noteLowerUpper);
        Collections.sort(noteLengthList);
        for(int ni=0; ni<noteLengthList.size()-1; ++ni)
        {
          ArrayList<int[]> shortNoteRemoved=CollectionUtils.deepCopy(shortGapRemoved);
          removeShortNote(shortNoteRemoved, noteLengthList.get(ni));
          int ms=Matching.computeMatchedLength(correctSoundSpecInterval.get(seq), shortNoteRemoved,
sequenceSpecLength.get(seq));
          for(int i=noteLengthList.get(ni); i<noteLengthList.get(ni+1); ++i) score[gapLengthList.get(gi)][i]=ms;
        }

        for(int i=gapLengthList.get(gi)+1; i<gapLengthList.get(gi+1); ++i)
System.arraycopy(score[gapLengthList.get(gi)], 0, score[i], 0, score[i].length);
      }
    }

    int bestScore=0;
    LinkedList<Parameter> bestParam=new LinkedList<>();
    for(int index=0; index<specIndexAmp.size(); ++index)
    {
      if(config.verbose && index%(specIndexAmp.size()/20)==0) System.out.println("Thresholding training
"+(int)((double)index/specIndexAmp.size()*100)+"%");

      Sequence changedSpec=specIndexAmp.get(index).get0();
      double threshold=specIndexAmp.get(index).get1();
      ArrayList<int[]> changedSoundInterval=soundInterval(spectrogramAmplitude.get(changedSpec),
threshold).stream().collect(CollectionUtils.arrayListCollector());
      int[][] changedScore=currentScore.get(changedSpec);

      ArrayList<Integer> gapLengthList=new ArrayList<Integer>(changedSoundInterval.size()-1);
      for(int i=0; i<changedSoundInterval.size()-1; ++i)
gapLengthList.add(changedSoundInterval.get(i+1)[0]-(changedSoundInterval.get(i)[0]+changedSoundInterval.get(i)
[1]));

  gapLengthList=gapLengthList.stream().collect(Collectors.toSet()).stream().filter(gl->gl<hyperParameter.gapLo
werUpper).collect(CollectionUtils.arrayListCollector());
      gapLengthList.add(0);

      ArrayList<Callable<Pair<Integer, int[]>>> job=new ArrayList<>(config.executor.getNumThread());
      for(int gapLength: gapLengthList)
      {
        job.add(new Callable<Pair<Integer, int[]>>(){
          @Override
          public Pair<Integer, int[]> call()
          {
            ArrayList<int[]> shortGapRemoved=CollectionUtils.deepCopy(changedSoundInterval);
            removeShortGap(shortGapRemoved, gapLength);

            ArrayList<Integer> noteLengthList=new ArrayList<Integer>(shortGapRemoved.stream()
              .map(si->si[1])
              .filter(nl->nl<hyperParameter.noteLowerUpper).collect(Collectors.toSet()));
            noteLengthList.add(0);
            noteLengthList.add(hyperParameter.noteLowerUpper);
            Collections.sort(noteLengthList);
            int[] score=new int[hyperParameter.noteLowerUpper];
            for(int ni=0; ni<noteLengthList.size()-1; ++ni)
            {
              int noteLength=noteLengthList.get(ni);
              ArrayList<int[]> shortNoteRemoved=CollectionUtils.deepCopy(shortGapRemoved);
              removeShortNote(shortNoteRemoved, noteLength);
              int ms=Matching.computeMatchedLength(correctSoundSpecInterval.get(changedSpec), shortNoteRemoved,
sequenceSpecLength.get(changedSpec));
              for(int i=noteLengthList.get(ni); i<noteLengthList.get(ni+1); ++i) score[i]=ms;
            }
            return new Pair<>(gapLength, score);
          }
        });
      }
      List<Future<Pair<Integer, int[]>>> future=config.executor.get().invokeAll(job);
      HashMap<Integer, int[]> parallel=new HashMap<>(future.size()*4/3);
      for(Future<Pair<Integer, int[]>> f: future)
      {
        Pair<Integer, int[]> p=f.get();
        parallel.put(p.get0(), p.get1());
      }
      gapLengthList.add(hyperParameter.gapLowerUpper);
      Collections.sort(gapLengthList);

      for(int gi=0; gi<gapLengthList.size()-1; ++gi)
      {
```

```java
        for(int i=gapLengthList.get(gi); i<gapLengthList.get(gi+1); ++i)
System.arraycopy(parallel.get(gapLengthList.get(gi)), 0, changedScore[i], 0, changedScore[i].length);
      }

      for(int gapLengthLower=0; gapLengthLower<hyperParameter.gapLowerUpper; ++gapLengthLower)
        for(int noteLengthLower=0; noteLengthLower<hyperParameter.noteLowerUpper; ++noteLengthLower)
      {
        int gl=gapLengthLower, nl=noteLengthLower;
        int sumScore=currentScore.values().stream().mapToInt(score->score[gl][nl]).sum();
        if(sumScore>bestScore)
        {
          bestScore=sumScore;
          bestParam.clear();
        }
        if(sumScore==bestScore)
        {
          bestParam.add(new Parameter(threshold, noteLengthLower, gapLengthLower));
        }
      }
    }

    Parameter parameter=bestParam.getFirst();
    for(Parameter th: bestParam)
    {
      if(th.amplitude>parameter.amplitude ||
        th.amplitude==parameter.amplitude && (th.noteLengthLower>parameter.noteLengthLower ||
        th.noteLengthLower==parameter.noteLengthLower && th.gapLengthLower>parameter.gapLengthLower))
        parameter=th;
    }
    return parameter;
  }

  /**
   * Performs boundary detection using trained thresholds.
   * @param parameter Trained thresholds.
   */
  public static HashMap<Sequence, ArrayList<int[]>> boundaryDetection(Collection<Sequence> sequence, Parameter
parameter, HyperParameter hyperParameter, Config config) throws IOException, UnsupportedAudioFileException,
NotConvergedException
  {
    HashMap<Sequence, double[]> spectrogramAmplitude=spectrogramAmplitude(sequence, hyperParameter, config);
    HashMap<Sequence, ArrayList<int[]>> noteList=new HashMap<>(sequence.size()*4/3);
    for(Sequence seq: sequence)
    {
      ArrayList<int[]> soundInterval=soundInterval(spectrogramAmplitude.get(seq),
parameter.amplitude).stream().collect(Collectors.toCollection(ArrayList::new));
      removeShortGap(soundInterval, parameter.gapLengthLower);
      removeShortNote(soundInterval, parameter.noteLengthLower);
      ArrayList<int[]> note=soundInterval.stream()
          .map(si->new int[]{si[0], si[1]})
          .collect(Collectors.toCollection(ArrayList::new));
      noteList.put(seq, note);
    }
    return noteList;
  }

  /**
   * Converts continuous outputs of a DNN into discrete values by averaging within the detected intervals.
   * @param soundInterval Detected sound intervals.
   * @param output Continuous outputs of a DNN.
   * @param stftParam
   * @return Averaged outputs.
   */
  public static HashMap<Sequence, ArrayList<double[]>> averageOutput(HashMap<Sequence, ArrayList<int[]>>
soundInterval, HashMap<Sequence, float[]> output, STFTParam stftParam)
  {
    HashMap<Sequence, ArrayList<double[]>> average=new HashMap<>(output.size()*4/3);
    for(Sequence seq: output.keySet())
    {
      ArrayList<double[]> averageList=new ArrayList<>(soundInterval.get(seq).size());
      average.put(seq, averageList);
      int height=stftParam.spectrogramLength(seq.getLength());
      int width=output.get(seq).length/height;
      for(int[] si: soundInterval.get(seq))
      {
        double[] ave=new double[width];
        averageList.add(ave);
        for(int y=si[0]; y<si[0]+si[1]; ++y) for(int x=0; x<width; ++x) ave[x]+=output.get(seq)[y*width+x];
        for(int x=0; x<width; ++x) ave[x]/=si[1];
      }
    }
    return average;
  }

  public static class HyperParameter
  {
    private STFTParam stftParam;
    private int dpssParam, freqOffset, freqLength, gapLowerUpper, noteLowerUpper;
    public HyperParameter(STFTParam stftParam, int dpssParam, int freqOffset, int freqLength, int gapLowerUpper,
        int noteLowerUpper) {
      this.stftParam = stftParam;
      this.dpssParam = dpssParam;
      this.freqOffset = freqOffset;
      this.freqLength = freqLength;
      this.gapLowerUpper = gapLowerUpper;
      this.noteLowerUpper = noteLowerUpper;
    }
  }

  public static class Parameter
  {
    private double amplitude;
    private int noteLengthLower, gapLengthLower;

    public Parameter(double amplitude, int noteLengthLower, int gapLengthLower)
    {
      this.amplitude = amplitude;
      this.noteLengthLower = noteLengthLower;
      this.gapLengthLower = gapLengthLower;
    }

    public double getAmplitude() {
      return amplitude;
    }

    public int getNoteLengthLower() {
      return noteLengthLower;
    }

    public int getGapLengthLower() {
      return gapLengthLower;
    }

    public void writeXml(Path file) throws IOException
    {
      Element rootEl=XmlUtils.rootElement("ThresholdingParameter");
      XmlUtils.addChild(rootEl, "Amplitude", amplitude);
      XmlUtils.addChild(rootEl, "NoteLengthLower", noteLengthLower);
      XmlUtils.addChild(rootEl, "GapLengthLower", gapLengthLower);
      XmlUtils.write(rootEl.getOwnerDocument(), file);
    }

    public static Parameter parseXml(Path file) throws IOException
    {
      Node rootChild=XmlUtils.parse(file).getFirstChild().getFirstChild();
      double amplitude=XmlUtils.nodeDouble(rootChild);
      rootChild=rootChild.getNextSibling();
      int noteLengthLower=XmlUtils.nodeInt(rootChild);
      rootChild=rootChild.getNextSibling();
      int gapLengthLower=XmlUtils.nodeInt(rootChild);
      return new Parameter(amplitude, noteLengthLower, gapLengthLower);
    }
  }

  public static class Config
  {
    private Executor executor;
    private Path waveFileDir;
```

```
    private boolean verbose;

    public Config(Executor executor, Path waveFileDir, boolean verbose) {
      this.executor = executor;
      this.waveFileDir = waveFileDir;
      this.verbose = verbose;
    }
  }
}
```

ViterbiSequencer.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package computation;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.LinkedList;

import utils.ArrayUtils;

/**
 * This class performs sequencing by the Viterbi algorithm, used in {@link HmmComputation}.
 * @author koumura
 *
 */
public class ViterbiSequencer
{
  private StateTransition transition;
  private double[] currentScore, nextScore;
  private ArrayList<int[]> srcState;

  public ViterbiSequencer(StateTransition transition)
  {
    this.transition = transition;
    this.currentScore = new double[transition.numState()];
    this.nextScore = new double[transition.numState()];
    this.srcState = new ArrayList<>();
  }

  /**
   * Computes label sequences that would generate the given observation probability.
   */
  public ArrayList<Integer> compLabelSequence(ArrayList<double[]> observationProbability)
  {
    for(int t=srcState.size(); t<observationProbability.size()-1; ++t) srcState.add(new
int[transition.numState()]);

    Arrays.fill(currentScore, Double.NEGATIVE_INFINITY);
    for(int d=0; d<transition.dstLabel(transition.headState()).length; ++d)
    {
      int dstState=transition.dstState(transition.headState())[d];
      int dstLabel=transition.dstLabel(transition.headState())[d];
      double transitionProb=transition.transitionProbability(transition.headState())[d];
      double observationProb=observationProbability.get(0)[dstLabel];
      currentScore[dstState]=Math.log(transitionProb)+Math.log(observationProb);
//    currentScore[dstState]=transitionProb+observationProb;
    }

    for(int t=0; t<observationProbability.size()-1; ++t)
    {
      Arrays.fill(nextScore, Double.NEGATIVE_INFINITY);
      for(int s=0; s<transition.numState(); ++s)
      {
        for(int d=0; d<transition.dstLabel(s).length; ++d)
        {
          int dstState=transition.dstState(s)[d];
          int dstLabel=transition.dstLabel(s)[d];
          double transitionProb=transition.transitionProbability(s)[d];
          double observationProb=observationProbability.get(t+1)[dstLabel];
          double ns=currentScore[s]+Math.log(transitionProb)+Math.log(observationProb);
//        double ns=currentScore[s]+transitionProb+observationProb;
          if(ns>=nextScore[dstState])
          {
            nextScore[dstState]=ns;
            srcState.get(t)[dstState]=s;
          }
        }
      }
      {
        double[] tmp=currentScore;
        currentScore=nextScore;
        nextScore=tmp;
      }
    }

    int currentState=-1;
    for(int s: transition.tailState()) if(currentState==-1||currentScore[s]>currentScore[currentState])
currentState=s;
    ArrayList<Integer> labelSequence=new ArrayList<Integer>(observationProbability.size());
    for(int t=observationProbability.size()-2; t>=0; --t)
    {
      int prevState=srcState.get(t)[currentState];
      int srcLabel=-1;
      for(int d=0; d<transition.dstState(prevState).length; ++d)
if(transition.dstState(prevState)[d]==currentState)
      {
        srcLabel=transition.dstLabel(prevState)[d];
        break;
      }
      labelSequence.add(srcLabel);
      currentState=prevState;
    }
    {
      int prevState=transition.headState();
      int srcLabel=-1;
      for(int d=0; d<transition.dstState(prevState).length; ++d)
if(transition.dstState(prevState)[d]==currentState)
      {
        srcLabel=transition.dstLabel(prevState)[d];
        break;
      }
      labelSequence.add(srcLabel);

    }
    Collections.reverse(labelSequence);

    return labelSequence;
  }

  /**
   * This class handles constraints of state transitions.
   * @author koumura
   *
   */
  public static interface StateTransition
  {
    int[] dstState(int srcState);
```

```
    int[] dstLabel(int srcState);
    double[] transitionProbability(int srcState);
    int headState();
    int numState();
    int numLabel();
    int[] tailState();
  }

  /**
   * Constraints for a first order Markov process.
   * label=int[]{0, 1, ..., numLabel-1, headState}.
   */
  public static class FirstOrderTransition implements StateTransition
  {
    private int[] dstLabel;
    private ArrayList<double[]> transitionProbability;

    /**
     * @param transitionProbability List(state0)[state1] = transition probability from state0 to state1.
     */
    public FirstOrderTransition(ArrayList<double[]> transitionProbability)
    {
      this.dstLabel = ArrayUtils.createSequence(0, transitionProbability.size()-1);
      this.transitionProbability = transitionProbability;
    }

    @Override
    public int[] dstState(int srcState) {return dstLabel;}

    @Override
    public int[] dstLabel(int srcState) {return dstLabel;}

    @Override
    public double[] transitionProbability(int srcState) {return transitionProbability.get(srcState);}

    @Override
    public int headState() {return numLabel();}

    @Override
    public int numState() {return numLabel();}

    @Override
    public int numLabel() {return dstLabel.length;}

    @Override
    public int[] tailState(){return dstLabel;}
  }

  /**
   * Constraints for a second order Markov process without sub-divisions in a state.
   * @author koumura
   *
   */
  public static class SecondOrderTransition implements StateTransition
  {
    private int[] dstLabel, tailState;
    private double[][][] transitionProbability;
    private int numState;
    private double[] evenTransitionProbability;
    private ArrayList<int[]> dstState;

    /**
     * @param transitionProbability double[state0][state1][state2] = transition probability from consecutive state0
& state1 to state2
     */
    public SecondOrderTransition(double[][][] transitionProbability)
    {
      this.dstLabel = ArrayUtils.createSequence(0, transitionProbability.length);
      this.transitionProbability = transitionProbability;

      numState=(numLabel()+1)*numLabel()+1;
      evenTransitionProbability=ArrayUtils.createFilled(numLabel(), 1.0/numLabel());

      dstState=new ArrayList<>(numState());
      for(int label0=0; label0<numLabel()+1; ++label0) for(int label1=0; label1<numLabel(); ++label1)
      {
        int dstLabel0=label1;
        int[] ds=new int[numLabel()];
        dstState.add(ds);
        for(int dstLabel1=0; dstLabel1<numLabel(); ++dstLabel1)
        {
          ds[dstLabel1]=state(dstLabel0, dstLabel1);
        }
      }
      {
        int label1=numLabel();
        int dstLabel0=label1;
        int[] ds=new int[numLabel()];
        dstState.add(ds);
        for(int dstLabel1=0; dstLabel1<numLabel(); ++dstLabel1)
        {
          ds[dstLabel1]=state(dstLabel0, dstLabel1);
        }
      }

      tailState=ArrayUtils.createSequence(0, numState);
    }

    @Override
    public int[] dstState(int srcState) {return dstState.get(srcState);}

    @Override
    public int[] dstLabel(int srcState) {return dstLabel;}

    @Override
    public double[] transitionProbability(int srcState)
    {
      int label0=label0(srcState);
      if(label0==numLabel()) return evenTransitionProbability;  //label0==head
      int label1=label1(srcState);
      return transitionProbability[label0][label1];
    }

    @Override
    public int headState() {return numState-1;}

    @Override
    public int numState() {return numState;}

    @Override
    public int numLabel() {return dstLabel.length;}

    @Override
    public int[] tailState(){return tailState;}

    private int label0(int state)
    {
      if(state==headState()) return numLabel();
      return state/numLabel();
    }

    private int label1(int state)
    {
      if(state==headState()) return numLabel();
      return state%numLabel();
    }

    private int state(int label0, int label1)
    {
      return label0*numLabel()+label1;
    }
  }

  /**
   * Constraints for a second order Markov process with sub-divisions in a state.
   * @author koumura
   *
   */
```

```
  public static class SecondOrderLowerTransition implements StateTransition
  {
    private ArrayList<int[]> dstState, dstLabel;
    private ArrayList<double[]> transitionProbability;
    private int numUpperSoundLabel, numLowerSoundLabel, numFullLabel, numState;
    private ArrayList<Integer> stateLabel0, stateLabel1, stateLowerLabel;
    private int[] tailState;

    /**
     * @param upperTransitionProbability double[state0][state1][state2] = transition probability from consecutive
state0 & state1 to state2
     * @param numUpperSoundLabel Number of labels (excluding the silent label).
     * @param numLowerSoundLabel Number of sub-divisions (excluding the silent label).
     */
    public SecondOrderLowerTransition(double[][][] upperTransitionProbability, int numUpperSoundLabel, int
numLowerSoundLabel)
    {
      this.numUpperSoundLabel=numUpperSoundLabel;
      this.numLowerSoundLabel=numLowerSoundLabel;
      this.numState=(numUpperSoundLabel+1)*numUpperSoundLabel*(numLowerSoundLabel+1)+1;
      numFullLabel=numUpperSoundLabel*numLowerSoundLabel+1;
      stateLabel0=new ArrayList<Integer>(numState);
      stateLabel1=new ArrayList<Integer>(numState);
      stateLowerLabel=new ArrayList<Integer>(numState);
      dstState=new ArrayList<>(numState);
      dstLabel=new ArrayList<>(numState);
      transitionProbability=new ArrayList<>(numState);

      int state=0;
      for(int ul0=0; ul0<numUpperSoundLabel+1; ++ul0)
        for(int ul1=0; ul1<numUpperSoundLabel; ++ul1)
          for(int ll=0; ll<numLowerSoundLabel+1; ++ll)
      {
        if(state!=state(ul0, ul1, ll)) System.err.println(state+" != "+state(ul0, ul1, ll));

        stateLabel0.add(ul0);
        stateLabel1.add(ul1);
        stateLowerLabel.add(ll);
        int[] ds=null, dl=null;
        double[] tp=null;
        if(ll<numLowerSoundLabel-1)
        {
          ds=new int[]{state, state+1};
          dl=new int[]{fullLabel(ul1, ll), fullLabel(ul1, ll+1)};
          tp=new double[]{0.5, 0.5};
        }
        else if(ll==numLowerSoundLabel-1)  //last lower
        {
          //allow no silence
          int numDst=numUpperSoundLabel+2;
          ds=new int[numDst];
          dl=new int[numDst];
          tp=new double[numDst];
          for(int d=0; d<numUpperSoundLabel; ++d)
          {
            ds[d]=state(ul1, d, 0);
            dl[d]=fullLabel(d, 0);
            if(ul0<numUpperSoundLabel) tp[d]=upperTransitionProbability[ul0][ul1][d]*(numDst-2)/numDst;
            else tp[d]=1.0/numDst;
          }
          {
            int d=numUpperSoundLabel;
            ds[d]=state;
            dl[d]=fullLabel(ul1, ll);
            tp[d]=1.0/numDst;
          }
          {
            int d=numUpperSoundLabel+1;
            ds[d]=state+1;
            dl[d]=silenceFullLabel();
            tp[d]=1.0/numDst;
          }

          //forbid no silence
          /*ds=new int[]{state, state+1};
          dl=new int[]{fullLabel(ul1, ll), silenceFullLabel()};
          tp=new double[]{0.5, 0.5};*/
        }
        else if(ll==numLowerSoundLabel)  //silence
        {
          int numDst=numUpperSoundLabel+1;
          ds=new int[numDst];
          dl=new int[numDst];
          tp=new double[numDst];
          for(int d=0; d<numUpperSoundLabel; ++d)
          {
            ds[d]=state(ul1, d, 0);
            dl[d]=fullLabel(d, 0);
            if(ul0<numUpperSoundLabel) tp[d]=upperTransitionProbability[ul0][ul1][d]*(numDst-1)/numDst;
            else tp[d]=1.0/numDst;
          }
          {
            int d=numUpperSoundLabel;
            ds[d]=state;
            dl[d]=silenceFullLabel();
            tp[d]=1.0/numDst;
          }
        }
        dstState.add(ds);
        dstLabel.add(dl);
        transitionProbability.add(tp);
        ++state;
      }
      //head state
      {
        int numDst=numUpperSoundLabel+1;
        int[] ds=new int[numDst];
        int[] dl=new int[numDst];
        double[] tp=ArrayUtils.createFilled(numDst, 1.0/numDst);
        for(int d=0; d<numUpperSoundLabel; ++d)
        {
          ds[d]=state(numUpperSoundLabel, d, 0);
          dl[d]=fullLabel(d, 0);
        }
        {
          int d=numUpperSoundLabel;
          ds[d]=state;
          dl[d]=silenceFullLabel();
        }
        dstState.add(ds);
        dstLabel.add(dl);
        transitionProbability.add(tp);

        stateLabel0.add(numUpperSoundLabel);
        stateLabel1.add(numUpperSoundLabel);
        stateLowerLabel.add(numLowerSoundLabel);
      }

      LinkedList<Integer> tailState=new LinkedList<>();
      for(int ul0=0; ul0<numUpperSoundLabel+1; ++ul0) for(int ul1=0; ul1<numUpperSoundLabel; ++ul1)
      {
        tailState.add(state(ul0, ul1, numLowerSoundLabel-1));
        tailState.add(state(ul0, ul1, numLowerSoundLabel));
      }
      tailState.add(headState());
      this.tailState=tailState.stream().mapToInt(x->x).toArray();
    }

    @Override
    public int[] dstState(int srcState){return dstState.get(srcState);}

    @Override
    public int[] dstLabel(int srcState) {return dstLabel.get(srcState);}

    @Override
    public double[] transitionProbability(int srcState)
    {
      return transitionProbability.get(srcState);
```

```java
//    return ArrayUtils.create(dstState.get(srcState).length, 1d);
    }

    @Override
    public int headState() {return numState()-1;}

    @Override
    public int numLabel() {return numFullLabel;}

    @Override
    public int numState() {return numState;}

    @Override
    public int[] tailState(){return tailState;}

    private int silenceFullLabel(){return numLabel()-1;}

    private int fullLabel(int upperLabel, int lowerLabel)
    {
      if(upperLabel==numUpperSoundLabel||lowerLabel==numLowerSoundLabel) return silenceFullLabel();
      return upperLabel*numLowerSoundLabel+lowerLabel;
    }

    private int state(int label0, int label1, int lowerLabel)
    {
      if(label1==numUpperSoundLabel) return headState();
      return (label0*numUpperSoundLabel+label1)*(numLowerSoundLabel+1)+lowerLabel;
    }

    public int upperLabel(int fullLabel)
    {
      if(fullLabel==silenceFullLabel()) return numUpperSoundLabel;
      return fullLabel/numLowerSoundLabel;
    }

    public int lowerLabel(int fullLabel)
    {
      if(fullLabel==silenceFullLabel()) return numLowerSoundLabel;
      return fullLabel%numLowerSoundLabel;
    }
  }
}
```

ConvLayer.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn.layer;

import com.sun.jna.ptr.IntByReference;

import cudnn.ActivationMode;
import cudnn.Cuda;
import cudnn.CudaDriver;
import cudnn.CudaException;
import cudnn.Cudnn;
import cudnn.CudnnException;
import cudnn.CudnnLibrary;
import cudnn.CudnnLibrary.cudnnConvolutionFwdPreference_t;
import cudnn.FloatType;
import cudnn.Pointer;

/**
 * A class for a convolutional layer.
 * {@link #init(FloatType, Cudnn, int)} and {@link #cudaMalloc(FloatType, int)} must be called before computation.
 * {@link #destroy(Cudnn)} must be called before termination of the program.
 * **Dev means a pointer in a GPU.
 * @author koumura
 *
 */
public class ConvLayer implements ParamLayer, NonDataLayer
{
  private Layer lower;
  private int numChannel, filterWidth, filterHeight, width, height;

  private float[] weightF, biasF;
  private double[] weightD, biasD;
  private int weightSize, biasSize;

  private CudnnLibrary.cudnnTensorDescriptor_t tensorDesc;
  private CudnnLibrary.cudnnTensorDescriptor_t biasTensorDesc;
  private CudnnLibrary.cudnnFilterDescriptor_t filterDesc;
  private CudnnLibrary.cudnnConvolutionDescriptor_t convDesc;
  private IntByReference forwardAlgo;
  private int backwardFilterAlgo, backwardDataAlgo;

  private Pointer weightDev, biasDev, actDev, gradWeightDev, gradBiasDev, derActDev;

  private ActivationMode activationMode;

  private Pointer workspaceDev;
  private int workspaceSize;

  private BackwardAlgorithm backwardAlgo;

  public ConvLayer(int numChannel, int filterHeight, int filterWidth, ActivationMode activationMode, Layer lower,
BackwardAlgorithm backwardAlgo)
  {
    this.lower=lower;

    weightSize=lower.getNumChannel()* filterWidth* filterHeight* numChannel;
    biasSize=numChannel;
    this.numChannel=numChannel;
    this.width=lower.getWidth()-filterWidth+1;
    this.height=lower.getHeight()-filterHeight+1;
    this.activationMode=activationMode;
    this.filterWidth=filterWidth;
    this.filterHeight=filterHeight;

    this.backwardAlgo=backwardAlgo;
  }

  public void destroy(Cudnn cudnn) throws CudnnException, CudaException
  {
    getTensorDesc().destroy(cudnn);
    biasTensorDesc.destroy(cudnn);
    filterDesc.destroy(cudnn);
    convDesc.destroy(cudnn);

    if(getWeightDev()!=null) Cuda.free(getWeightDev());
    if(getBiasDev()!=null) Cuda.free(getBiasDev());
    if(actDev!=null) Cuda.free(actDev);
//  /*act!=value*/if(getValueDev()!=null) Cuda.free(getValueDev());
    if(getGradWeightDev()!=null) Cuda.free(getGradWeightDev());
    if(getGradBiasDev()!=null) Cuda.free(getGradBiasDev());
    if(derActDev!=null) Cuda.free(derActDev);
//  /*act!=value*/if(derValueDev!=null) Cuda.free(derValueDev);
  }

  public void cudaMalloc(FloatType floatType, int batchSize) throws CudaException
```

```
  {
    actDev=Cuda.malloc(floatType.getBytes()* batchSize * getNumChannel() * getHeight()                *
getWidth());
//  /*act!=value*/valueDev=Cuda.malloc(floatType.getBytes() * batchSize * getNumChannel() * getHeight()
* getWidth());
//  derActDev=Cuda.malloc(floatType.getBytes()*lower.sizeBCHW());
//  /*act!=value*/derValueDev=Cuda.malloc(  floatType.getBytes() * lower.sizeBCHW());
    derActDev=Cuda.malloc(floatType.getBytes()*sizeCHW()*batchSize);
//  /*act!=value*/derValueDev=Cuda.malloc(  floatType.getBytes() * sizeBCHW());
    weightDev=Cuda.malloc(    floatType.getBytes() * weightSize);
    biasDev=Cuda.malloc(    floatType.getBytes() * biasSize);
    gradWeightDev=Cuda.malloc(    floatType.getBytes() * weightSize);
    gradBiasDev=Cuda.malloc(    floatType.getBytes() * biasSize);
  }
  public void setWodkspace(Pointer workspaceDev, int workspaceSize)
  {
    this.workspaceDev=workspaceDev;
    this.workspaceSize=workspaceSize;
  }

  protected void compForwardAct(FloatType floatType, Cudnn cudnn) throws CudnnException, CudaException
  {
    cudnn.convolutionForward(Cudnn.const1(floatType), lower.getTensorDesc(),
lower.getValueDev().getJnaPointer(), filterDesc, weightDev.getJnaPointer(), convDesc, forwardAlgo.getValue(),
workspaceDev.getJnaPointer(), workspaceSize, Cudnn.const0(floatType), getTensorDesc(), actDev.getJnaPointer());
    cudnn.addTensor(CudnnLibrary.cudnnAddMode_t.CUDNN_ADD_SAME_C, Cudnn.const1(floatType), biasTensorDesc,
getBiasDev().getJnaPointer(), Cudnn.const1(floatType), getTensorDesc(), actDev.getJnaPointer());
  }

  public void compForward(FloatType floatType, Cudnn cudnn, int batchSize, CudaDriver driver) throws CudnnException,
CudaException
  {
    compForwardAct(floatType, cudnn);
    if(activationMode!=ActivationMode.IDENT)
      cudnn.activationForward(activationMode.getValue(), Cudnn.const1(floatType), getTensorDesc(),
actDev.getJnaPointer(), Cudnn.const0(floatType), getTensorDesc(), actDev.getJnaPointer());
  }

 public void compBackward(FloatType floatType, Cudnn cudnn, int batchSize, CudaDriver driver) throws CudnnException,
CudaException
  {
    if(activationMode!=ActivationMode.IDENT)
    {
      cudnn.activationBackward(activationMode.getValue(), Cudnn.const1(floatType),
          getTensorDesc(), getValueDev().getJnaPointer(), getTensorDesc(), derActDev.getJnaPointer(),
          getTensorDesc(), actDev.getJnaPointer(), Cudnn.const0(floatType), getTensorDesc(),
derActDev.getJnaPointer());
    }

    // valueDev layer
    cudnn.convolutionBackwardBias(Cudnn.const1(floatType), getTensorDesc(),
        derActDev.getJnaPointer(), Cudnn.const0(floatType), biasTensorDesc, getGradBiasDev().getJnaPointer());

    cudnn.convolutionBackwardFilter(Cudnn.const1(floatType), lower.getTensorDesc(),
        lower.getValueDev().getJnaPointer(), getTensorDesc(), derActDev.getJnaPointer(), convDesc,
        backwardFilterAlgo, workspaceDev.getJnaPointer(), workspaceSize,
        Cudnn.const0(floatType), filterDesc, gradWeightDev.getJnaPointer());

    if(lower instanceof NonDataLayer)
    {
      cudnn.convolutionBackwardData(Cudnn.const1(floatType), filterDesc,
          weightDev.getJnaPointer(), getTensorDesc(), derActDev.getJnaPointer(), convDesc,
          backwardDataAlgo, workspaceDev.getJnaPointer(), workspaceSize,
          Cudnn.const0(floatType), lower.getTensorDesc(), ((NonDataLayer)
lower).getDerValueDev().getJnaPointer());
    }
  }

  public void init(FloatType floatType, Cudnn cudnn, int batchSize) throws CudnnException
  {
    tensorDesc=cudnn.createTensorDescriptor();
    cudnn.initTensorDesc(floatType, this, batchSize);
    biasTensorDesc=cudnn.createTensorDescriptor();
```

```
    convDesc=cudnn.createConvolutionDescriptor();
    biasTensorDesc.setTensor4dDescriptor(cudnn, CudnnLibrary.cudnnTensorFormat_t.CUDNN_TENSOR_NCHW,
        floatType.getDataTypeValue(),
        1, getNumChannel(),
        1, 1);

    filterDesc=cudnn.createFilterDescriptor();
    filterDesc.setFilter4dDescriptor(cudnn, floatType.getDataTypeValue(),
        getNumChannel(),
        lower.getNumChannel(),
        filterHeight,
        filterWidth);

    convDesc.setConvolution2dDescriptor(cudnn, 0, 0,
        1, 1,
        1, 1,
        CudnnLibrary.cudnnConvolutionMode_t.CUDNN_CROSS_CORRELATION);

    int algoValue=cudnn.getConvolutionForwardAlgorithm(lower.getTensorDesc(), filterDesc, convDesc, tensorDesc,
cudnnConvolutionFwdPreference_t.CUDNN_CONVOLUTION_FWD_PREFER_FASTEST, 0);
    forwardAlgo=new IntByReference(algoValue);

    if(backwardAlgo==BackwardAlgorithm.FAST_NON_DETERMINISTIC)
    {
      backwardFilterAlgo=CudnnLibrary.cudnnConvolutionBwdFilterAlgo_t.CUDNN_CONVOLUTION_BWD_FILTER_ALGO_0;
      backwardDataAlgo=CudnnLibrary.cudnnConvolutionBwdDataAlgo_t.CUDNN_CONVOLUTION_BWD_DATA_ALGO_0;
    }
    else
    {
      backwardFilterAlgo=CudnnLibrary.cudnnConvolutionBwdFilterAlgo_t.CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1;
      backwardDataAlgo=CudnnLibrary.cudnnConvolutionBwdDataAlgo_t.CUDNN_CONVOLUTION_BWD_DATA_ALGO_1;
    }
  }

  public float[] getWeightF() {
    return weightF;
  }

  public float[] getBiasF() {
    return biasF;
  }

  public int getNumChannel() {
    return numChannel;
  }

  public int getWidth() {
    return width;
  }

  public int getHeight() {
    return height;
  }

  public Pointer getWeightDev() {
    return weightDev;
  }

  public Pointer getBiasDev() {
    return biasDev;
  }

  public CudnnLibrary.cudnnTensorDescriptor_t getTensorDesc() {
    return tensorDesc;
  }

  public int getWeightSize() {
    return weightSize;
  }

  public void setWeightSize(int weightSize) {
    this.weightSize = weightSize;
  }
```

```
  public int getBiasSize() {
    return biasSize;
  }

  public void setBiasSize(int biasSize) {
    this.biasSize = biasSize;
  }

  public void setWeightF(float[] weightF) {
    this.weightF = weightF;
  }

  public void setBiasF(float[] biasF) {
    this.biasF = biasF;
  }

  public void setWeightD(double[] weightD) {
    this.weightD = weightD;
  }

  public void setBiasD(double[] biasD) {
    this.biasD = biasD;
  }

  @Override
  public Layer getLower() {
    return lower;
  }

  @Override
  public Pointer getValueDev() {
//  /*act!=value*/return valueDev;
    return actDev;
  }

  public Pointer getGradWeightDev() {
    return gradWeightDev;
  }

  public Pointer getGradBiasDev() {
    return gradBiasDev;
  }

  @Override
  public Pointer getDerValueDev()
  {
//  /*act!=value*/return derValueDev;
    return derActDev;
  }

  public double[] getWeightD(){return weightD;}

  public double[] getBiasD(){return biasD;}

  protected Pointer getActDev() {
    return actDev;
  }

  public Pointer getDerActDev() {
    return derActDev;
  }

  public int getFilterHeight(){return filterHeight;}

  public int getFilterWidth(){return filterWidth;}

  public int compWorkspaceSize(Cudnn cudnn) throws CudnnException
  {
    int sizeInBytes=Integer.max(Integer.max(
      cudnn.getConvolutionForwardWorkspaceSize(lower.getTensorDesc(), filterDesc, convDesc, tensorDesc,
forwardAlgo.getValue()),
      cudnn.convolutionBackwardDataWorkspaceSize(filterDesc, tensorDesc, convDesc, lower.getTensorDesc(),
backwardDataAlgo)),
      cudnn.convolutionBackwardFilterWorkspaceSize(lower.getTensorDesc(), tensorDesc, convDesc, filterDesc,
backwardFilterAlgo));
    return sizeInBytes;
  }

  public static enum BackwardAlgorithm
  {
    FAST_NON_DETERMINISTIC, SLOW_DETERMINISTIC;
  }
}
```

DataLayer.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn.layer;

import cudnn.CudaException;
import cudnn.Cudnn;
import cudnn.CudnnException;
import cudnn.CudnnLibrary;
import cudnn.FloatType;
import cudnn.Pointer;

/**
 * A class for a data layer.
 * {@link #init(FloatType, Cudnn, int)} and {@link #cudaMalloc(FloatType, int)} must be called before computation.
 * {@link #destroy(Cudnn)} must be called before termination of the program.
 * **Dev means a pointer in a GPU.
 * @author koumura
 *
 */
public class DataLayer implements Layer
{
  private int numChannel, height, width;
  private CudnnLibrary.cudnnTensorDescriptor_t tensorDesc;
  private Pointer dataDev;

  public DataLayer(int numChannel, int height, int width)
  {
    this.numChannel = numChannel;
    this.height = height;
    this.width = width;
  }

  public void setDataDev(Pointer dataDev)
  {
    this.dataDev=dataDev;
  }

  public void destroy(Cudnn cudnn) throws CudnnException, CudaException
  {
    tensorDesc.destroy(cudnn);
  }

  public void init(FloatType floatType, Cudnn cudnn, int batchSize) throws CudnnException
```

```
  {
    tensorDesc=cudnn.createTensorDescriptor();
    cudnn.initTensorDesc(floatType, this, batchSize);
  }

  public int getNumChannel() {
    return numChannel;
  }

  public int getHeight() {
    return height;
  }

  public int getWidth() {
    return width;
  }

  public CudnnLibrary.cudnnTensorDescriptor_t getTensorDesc() {
    return tensorDesc;
  }

  @Override
  public void cudaMalloc(FloatType floatType, int batchSize) throws CudaException{}

  public Pointer getValueDev() {
    return dataDev;
  }
}
```

Layer.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn.layer;

import cudnn.CudaException;
import cudnn.Cudnn;
import cudnn.CudnnException;
import cudnn.CudnnLibrary;
import cudnn.FloatType;
import cudnn.Pointer;

/**
 * An interface for a general layer.
 * {@link #init(FloatType, Cudnn, int)} and {@link #cudaMalloc(FloatType, int)} must be called before computation.
 * {@link #destroy(Cudnn)} must be called before termination of the program.
 * **Dev means a pointer in a GPU.
 * @author koumura
 *
 */
public interface Layer
{
  int getNumChannel();
  int getHeight();
  int getWidth();
  /**
   * @return numChannel * height * width
   */
  default int sizeCHW(){return getNumChannel()*getHeight()*getWidth();}
  /**
   * @return height * width
   */
  default int sizeHW(){return getHeight()*getWidth();}

  void cudaMalloc(FloatType floatType, int batchSize) throws CudaException;
  CudnnLibrary.cudnnTensorDescriptor_t getTensorDesc();
  void init(FloatType floatType, Cudnn cudnn, int batchSize) throws CudnnException;
  void destroy(Cudnn cudnn) throws CudnnException, CudaException;

  Pointer getValueDev();
}
```

NonDataLayer.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn.layer;

import cudnn.CudaDriver;
import cudnn.CudaException;
import cudnn.Cudnn;
import cudnn.CudnnException;
import cudnn.FloatType;
import cudnn.Pointer;

/**
 * An interface for a non-data layer.
 * **Dev means a pointer in a GPU.
 * @author koumura
 *
 */
public interface NonDataLayer extends Layer
{
  void compForward(FloatType floatType, Cudnn cudnn, int batchSize, CudaDriver driver) throws CudnnException,
CudaException;
  void compBackward(FloatType floatType, Cudnn cudnn, int batchSize, CudaDriver driver) throws CudnnException,
CudaException;
  Pointer getDerValueDev();
  /**
   * @return Lower layer, ie. input layer of this layer.
   */
  Layer getLower();
}
```

OutputLayer.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
```

```
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn.layer;

import cudnn.CudaDriver;
import cudnn.CudaException;
import cudnn.FloatType;
import cudnn.IntType;
import cudnn.Pointer;

/**
 * An interface for an output layer.
 * {@link #initError(FloatType)} must be called before computing the output error.
 * **Dev means a pointer in a GPU.
 * @author koumura
 *
 */
public interface OutputLayer extends Layer
{
  double compError(CudaDriver driver, Pointer labelDev, IntType labelType, FloatType floatType) throws
CudaException;
  void initError(FloatType floatType) throws CudaException;
  Pointer getDerActDev();
  void backwardCost(CudaDriver driver, FloatType floatType, Pointer labelDev, int batchSize, int blockWidth, IntType
labelType) throws CudaException;
}
```


ParamLayer.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn.layer;

import cudnn.Cuda;
import cudnn.CudaException;
import cudnn.FloatType;
import cudnn.Pointer;

/**
 * An interface for a layer with parameters (weights & biases).
 * **Dev means a pointer in a GPU.
 * **F means float vlaues. **D means double values.
 * @author koumura
 *
 */
public interface ParamLayer extends Layer
{
  float[] getWeightF();
  float[] getBiasF();
  double[] getWeightD();
  double[] getBiasD();
  void setWeightF(float[] weight);
  void setBiasF(float[] bias);
  void setWeightD(double[] weight);
  void setBiasD(double[] bias);
  Pointer getWeightDev();
  Pointer getBiasDev();
  Pointer getGradWeightDev();
  Pointer getGradBiasDev();
  int getWeightSize();
  int getBiasSize();

  default void copyParamToDev(FloatType floatType) throws CudaException
  {
    if(floatType==FloatType.SINGLE)
    {
      Cuda.memcpyAsyncHostToDevice(getWeightDev(), getWeightF());
      Cuda.memcpyAsyncHostToDevice(getBiasDev(), getBiasF());
    }
    else
    {
      Cuda.memcpyAsyncHostToDevice(getWeightDev(), getWeightD());
      Cuda.memcpyAsyncHostToDevice(getBiasDev(), getBiasD());
    }
  }

  default void copyParamFromDev(FloatType floatType) throws CudaException
  {
    if(floatType==FloatType.SINGLE)
    {
      Cuda.memcpyAsyncDeviceToHost(Pointer.to(getWeightF()), getWeightDev(),
floatType.getBytes()*getWeightSize());
      Cuda.memcpyAsyncDeviceToHost(Pointer.to(getBiasF()), getBiasDev(), floatType.getBytes()*getBiasSize());
    }
    else
    {
      Cuda.memcpyAsyncDeviceToHost(Pointer.to(getWeightD()), getWeightDev(),
floatType.getBytes()*getWeightSize());
      Cuda.memcpyAsyncDeviceToHost(Pointer.to(getBiasD()), getBiasDev(), floatType.getBytes()*getBiasSize());
    }
  }

  default void newWeightBias(FloatType floatType)
  {
    if(floatType==FloatType.SINGLE)
    {
      setWeightF(new float[getWeightSize()]);
      setBiasF(new float[getBiasSize()]);
    }
    else
    {
      setWeightD(new double[getWeightSize()]);
      setBiasD(new double[getBiasSize()]);
    }

  }
}
```

PoolLayer.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
```

```java
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn.layer;

import cudnn.Cuda;
import cudnn.CudaDriver;
import cudnn.CudaException;
import cudnn.Cudnn;
import cudnn.CudnnException;
import cudnn.CudnnLibrary;
import cudnn.FloatType;
import cudnn.Pointer;
import cudnn.PoolingMode;

/**
 * A class for a pooling layer.
 * {@link #init(FloatType, Cudnn, int)} and {@link #cudaMalloc(FloatType, int)} must be called before computation.
 * {@link #destroy(Cudnn)} must be called before termination of the program.
 * **Dev means a pointer in a GPU.
 * @author koumura
 *
 */
public class PoolLayer implements NonDataLayer
{
  private int numChannel, height, width;
  private int filterHeight, filterWidth;
  private int stride;
  private Layer lower;

  private CudnnLibrary.cudnnTensorDescriptor_t tensorDesc;
  private CudnnLibrary.cudnnPoolingDescriptor_t poolDesc;

  private Pointer valueDev;
  private Pointer derValueDev;

  private PoolingMode poolingMode;

  public PoolLayer(int filterHeight, int filterWidth, int stride, PoolingMode poolingMode, Layer lower)
  {
    this.lower=lower;
    this.numChannel=lower.getNumChannel();
    this.height=(lower.getHeight()-(filterHeight-stride))/stride;
    this.width=(lower.getWidth()-(filterWidth-stride))/stride;
    this.filterHeight=filterHeight;
    this.filterWidth=filterWidth;
    this.stride=stride;
    this.lower=lower;
    this.poolingMode=poolingMode;
  }
  public PoolLayer(int filterSize, int stride, PoolingMode poolingMode, Layer lower)
  {
    this(filterSize, filterSize, stride, poolingMode, lower);
  }

  public void destroy(Cudnn cudnn) throws CudnnException, CudaException
  {
    getTensorDesc().destroy(cudnn);
    poolDesc.destroy(cudnn);

//  if(getValueDev()!=null) Cuda.free(getValueDev());
//  if(derValueDev!=null) Cuda.free(derValueDev);
    if(valueDev!=null) valueDev.free();
    if(derValueDev!=null) derValueDev.free();
  }

  public void cudaMalloc(FloatType floatType, int batchSize) throws CudaException
  {
    valueDev=Cuda.malloc(floatType.getBytes() * batchSize * sizeCHW());
    derValueDev=Cuda.malloc(  floatType.getBytes() * batchSize * sizeCHW());
  }

  public void cudaMallocCombLower(FloatType floatType, int batchSize, Pointer valueDev, Pointer derValueDev) throws
CudaException
  {
    this.valueDev=valueDev;
    this.derValueDev=derValueDev;
  }

  public void init(FloatType floatType, Cudnn cudnn, int batchSize) throws CudnnException
  {
    tensorDesc=cudnn.createTensorDescriptor();
    cudnn.initTensorDesc(floatType, this, batchSize);

    poolDesc=cudnn.createPoolingDescriptor();
    poolDesc.setPooling2dDescriptor(cudnn, poolingMode.getValue(),filterHeight, filterWidth,
        0, 0,
        getStride(), getStride());
  }

  public void compForward(FloatType floatType, Cudnn cudnn, int batchSize, CudaDriver driver) throws CudnnException,
CudaException
  {
    cudnn.poolingForward(poolDesc, Cudnn.const1(floatType), lower.getTensorDesc(),
lower.getValueDev().getJnaPointer(), Cudnn.const0(floatType), getTensorDesc(), getValueDev().getJnaPointer());

  }

  public void compBackward(FloatType floatType, Cudnn cudnn, int batchSize, CudaDriver driver) throws CudnnException
  {
    if(lower instanceof NonDataLayer)
    {
      cudnn.poolingBackward(poolDesc, Cudnn.const1(floatType),
          getTensorDesc(), valueDev.getJnaPointer(), getTensorDesc(), derValueDev.getJnaPointer(),
          lower.getTensorDesc(), lower.getValueDev().getJnaPointer(), Cudnn.const0(floatType),
lower.getTensorDesc(), ((NonDataLayer) lower).getDerValueDev().getJnaPointer());
    }
  }

  public Pointer derValueDev(){return derValueDev;}

  public int getStride() {
    return stride;
  }

  public int getNumChannel() {
    return numChannel;
  }

  public int getHeight() {
    return height;
  }

  public int getWidth() {
    return width;
  }

  public CudnnLibrary.cudnnTensorDescriptor_t getTensorDesc() {
    return tensorDesc;
  }

  public void setTensorDesc(CudnnLibrary.cudnnTensorDescriptor_t tensorDesc) {
    this.tensorDesc = tensorDesc;
  }

  public Pointer getValueDev() {
    return valueDev;
  }
```

```java
  @Override
  public Pointer getDerValueDev() {return derValueDev;}

  @Override
  public Layer getLower() {return lower;}
}
```

SeqSoftmaxConvLayer.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn.layer;

import cudnn.ActivationMode;
import cudnn.Cuda;
import cudnn.CudaDriver;
import cudnn.CudaException;
import cudnn.Cudnn;
import cudnn.CudnnException;
import cudnn.CudnnLibrary;
import cudnn.FloatType;
import cudnn.IntType;
import cudnn.Pointer;

/**
 * A class for a convolutional layer with the softmax activation function.
 * {@link #init(FloatType, Cudnn, int)} and {@link #cudaMalloc(FloatType, int)} must be called before computation.
 * {@link #destroy(Cudnn)} must be called before termination of the program.
 * **Dev means a pointer in a GPU.
 *
 * @author koumura
 *
 */
public class SeqSoftmaxConvLayer extends ConvLayer implements OutputLayer
{
  private Pointer sizeDev, errorDev;
  private int labelShiftUpperH, labelShiftUpperW, labelShiftY, labelShiftX, labelWidth, labelHeight, batchSize;
  private float outputLowerForErrorF;
  private double outputLowerForErrorD;
  private float[] errorF;
  private double[] errorD;

  private int singleSize, singleHeight, singleWidth;
  private CudnnLibrary.cudnnTensorDescriptor_t softmaxTensorDesc;

  public SeqSoftmaxConvLayer(int singleSize, int singleHeight, int singleWidth, int filterHeight, int filterWidth,
Layer lower, ConvLayer.BackwardAlgorithm backwardAlgo)
  {
    super(singleSize*singleHeight*singleWidth, filterHeight, filterWidth, ActivationMode.IDENT, lower,
backwardAlgo);
    this.singleSize=singleSize;
    this.singleHeight=singleHeight;
    this.singleWidth=singleWidth;
  }

  @Override
  public void init(FloatType floatType, Cudnn cudnn, int batchSize) throws CudnnException
  {
    super.init(floatType, cudnn, batchSize);
    softmaxTensorDesc=cudnn.createTensorDescriptor();
    softmaxTensorDesc.setTensor4dDescriptor(cudnn, CudnnLibrary.cudnnTensorFormat_t.CUDNN_TENSOR_NCHW,
floatType.getDataTypeValue(), batchSize, singleSize, singleHeight*getHeight(), singleWidth*getWidth());
  }

  @Override
  public void compForward(FloatType floatType, Cudnn cudnn, int batchSize, CudaDriver driver) throws CudnnException,
CudaException
  {
    compForwardAct(floatType, cudnn);

    cudnn.softmaxForward(CudnnLibrary.cudnnSoftmaxAlgorithm_t.CUDNN_SOFTMAX_ACCURATE,
CudnnLibrary.cudnnSoftmaxMode_t.CUDNN_SOFTMAX_MODE_CHANNEL,
        Cudnn.const1(floatType), softmaxTensorDesc, getActDev().getJnaPointer(), Cudnn.const0(floatType),
softmaxTensorDesc, getValueDev().getJnaPointer());
  }

  public int finalSizeHW(){return sizeHW()*singleHeight*singleWidth;}

  @Override
  public void cudaMalloc(FloatType floatType, int batchSize) throws CudaException
  {
    super.cudaMalloc(floatType, batchSize);
    sizeDev=Cuda.malloc(11*Integer.BYTES);
    this.batchSize=batchSize;
  }

  public void initError(FloatType floatType) throws CudaException
  {
    errorDev=Cuda.malloc(batchSize*finalSizeHW()*floatType.getBytes());
    if(floatType==FloatType.SINGLE)
    {
      errorF=new float[batchSize*finalSizeHW()];
    }
    else
    {
      errorD=new double[batchSize*finalSizeHW()];
    }
  }

  @Override
  public void destroy(Cudnn cudnn) throws CudaException, CudnnException
  {
    super.destroy(cudnn);
    if(errorDev!=null) errorDev.free();
    if(sizeDev!=null) sizeDev.free();
  }

  @Override
  public void copyParamToDev(FloatType floatType) throws CudaException
  {
    super.copyParamToDev(floatType);
    if(sizeDev!=null)
    {
      int[] size=new int[11];
      size[0]=batchSize;
      size[1]=getNumChannel();
      size[2]=getHeight();
      size[3]=getWidth();
      size[4]=getSingleSize();
      size[5]=singleHeight;
      size[6]=singleWidth;
      size[7]=labelHeight;
      size[8]=labelWidth;
      size[9]=labelShiftUpperH;
      size[10]=labelShiftUpperW;
      Cuda.memcpyAsyncHostToDevice(sizeDev, size);
    }
  }
```

```java
  /**
   * @return sum, not average
   */
  public double compError(CudaDriver driver, Pointer labelDev, IntType labelType, FloatType floatType) throws
CudaException
  {
    jcuda.Pointer pLower;
    if(floatType==FloatType.SINGLE) pLower=jcuda.Pointer.to(new float[]{outputLowerForErrorF});
    else pLower=jcuda.Pointer.to(new double[]{outputLowerForErrorD});

    int blockWidth=128;

  driver.call("SeqSoftmaxConvError"+(floatType==FloatType.SINGLE?"Float":"Double")+(labelType==IntType.BYTE?"C
har":""), (int)Math.ceil((double)batchSize*finalSizeHW()/blockWidth), 1, 1, blockWidth, 1, 1, 0, jcuda.Pointer.to(
        jcuda.Pointer.to(errorDev),
        jcuda.Pointer.to(getValueDev()),
        jcuda.Pointer.to(labelDev),
        jcuda.Pointer.to(sizeDev),
        jcuda.Pointer.to(new int[]{labelShiftY}),
        jcuda.Pointer.to(new int[]{labelShiftX}),
        pLower
        ));

    double sum=0;
    if(floatType==FloatType.SINGLE)
    {
      Cuda.memcpyDeviceToHost(Pointer.to(errorF), errorDev, errorF.length*floatType.getBytes());
      for(float e: errorF) sum+=e;
    }
    else
    {
      Cuda.memcpyDeviceToHost(Pointer.to(errorD), errorDev, errorD.length*floatType.getBytes());
      for(double e: errorD) sum+=e;
    }
    return sum;
  }

  public void backwardCost(CudaDriver driver, FloatType floatType, Pointer labelDev, int batchSize, int blockWidth,
IntType labelType) throws CudaException
  {

  driver.call("SeqSoftmaxConvBackward"+(floatType==FloatType.SINGLE?"Float":"Double")+(labelType==IntType.BYTE
?"Char":""), (int)Math.ceil((double)batchSize*finalSizeHW()/blockWidth), 1, 1, blockWidth, 1, 1, 0,
jcuda.Pointer.to(
        jcuda.Pointer.to(labelDev),
        jcuda.Pointer.to(getDerActDev()),
        jcuda.Pointer.to(sizeDev),
        jcuda.Pointer.to(new int[]{labelShiftY}),
        jcuda.Pointer.to(new int[]{labelShiftX})
        ));
  }

  public int getLabelHeight(){return labelHeight;}

  public int getLabelWidth(){return labelWidth;}

  public void setLabelShiftUpper(int labelShiftUpperH, int labelShiftUpperW) {
    this.labelShiftUpperH = labelShiftUpperH;
    this.labelHeight=getHeight()*labelShiftUpperH-1+singleHeight;
    this.labelShiftUpperW = labelShiftUpperW;
    this.labelWidth=getWidth()*labelShiftUpperW-1+singleWidth;
  }

  public void setLabelShift(int labelShiftY, int labelShiftX) {
    this.labelShiftY = labelShiftY;
    this.labelShiftX = labelShiftX;
  }

  public int countValidLabelSize(byte[] label, int batchSize)
  {
    int size=0;
    for(int b=0; b<batchSize; ++b)
      for(int y=0; y<getHeight(); ++y) for(int x=0; x<getWidth(); ++x)
        for(int sy=0; sy<singleHeight; ++sy) for(int sx=0; sx<singleWidth; ++sx)
    {
      byte la=label[labelIndex(b, y, x, sy, sx)];
      if(la>=0) ++size;
    }
    return size;
  }
  public int countValidLabelSize(int[] label, int batchSize)
  {
    int size=0;
    for(int b=0; b<batchSize; ++b)
      for(int y=0; y<getHeight(); ++y) for(int x=0; x<getWidth(); ++x)
        for(int sy=0; sy<singleHeight; ++sy) for(int sx=0; sx<singleWidth; ++sx)
    {
      int la=label[labelIndex(b, y, x, sy, sx)];
      if(la>=0) ++size;
    }
    return size;
  }

  public int labelIndex(int batch, int y, int x, int singleY, int singleX)
  {
    return
batch*labelWidth*labelHeight+(y*labelShiftUpperH+labelShiftY+singleY)*getLabelWidth()+x*labelShiftUpperW+label
ShiftX+singleX;
  }

  public void setOutputLowerForError(double outputLowerForError, FloatType floatType) {
    if(floatType==FloatType.SINGLE)
    {
      this.outputLowerForErrorF = (float)outputLowerForError;
    }
    else
    {
      this.outputLowerForErrorD = outputLowerForError;
    }
  }

  public int getSingleHeight() {
    return singleHeight;
  }

  public int getSingleWidth() {
    return singleWidth;
  }

  public int getSingleSize() {
    return singleSize;
  }
}
```

Adam.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn.learner;
```

```java
import java.util.ArrayList;

import cudnn.Cuda;
import cudnn.CudaDriver;
import cudnn.CudaException;
import cudnn.Cudnn;
import cudnn.CudnnException;
import cudnn.FloatType;
import cudnn.Pointer;

/**
 * An implementation of weight updating by Adam.
 * {@link #init()} and {@link #destroy()} must be called before and after the computation.
 * @see http://arxiv.org/abs/1412.6980
 * @author koumura
 *
 */
public class Adam
{
  private Model model;
  private double alpha, beta1, beta2, epsilon, beta1T, beta2T;
  private ArrayList<Pointer> moment, moment2;
  private Pointer hyperParam;
  private CudaDriver driver;
  private Cudnn cudnn;
  private FloatType floatType;

  public Adam(Model model, FloatType floatType, double alpha, double beta1, double beta2, double epsilon, Cudnn
cudnn, CudaDriver driver)
  {
    this.model = model;
    this.alpha = alpha;
    this.beta1 = beta1;
    this.beta2 = beta2;
    this.epsilon=epsilon;
    this.cudnn=cudnn;
    this.driver=driver;
    this.floatType=floatType;
  }
  public Adam(Model model, FloatType floatType, Cudnn cudnn, CudaDriver driver)
  {
    this(model, floatType, 0.001, 0.9, 0.999, 1e-8, cudnn, driver);
  }

  public void init() throws CudaException
  {
    int blockWidth=128;
    jcuda.Pointer p0;
    if(floatType==FloatType.SINGLE) p0=jcuda.Pointer.to(new float[]{0});
    else p0=jcuda.Pointer.to(new double[]{0});

    moment=new ArrayList<Pointer>(model.getParamSize().size());
    moment2=new ArrayList<Pointer>(model.getParamSize().size());
    for(int p: model.getParamSize())
    {
      moment.add(Cuda.malloc(p*floatType.getBytes()));
      moment2.add(Cuda.malloc(p*floatType.getBytes()));


      driver.call("Fill"+(floatType==FloatType.SINGLE?"Float":"Double"), (int)Math.ceil((double)p/blockWidth),
1, 1, blockWidth, 1, 1, 0, jcuda.Pointer.to(
          jcuda.Pointer.to(moment.get(moment.size()-1)),
          p0,
          jcuda.Pointer.to(new int[]{p})
          ));
      driver.call("Fill"+(floatType==FloatType.SINGLE?"Float":"Double"), (int)Math.ceil((double)p/blockWidth),
1, 1, blockWidth, 1, 1, 0, jcuda.Pointer.to(
          jcuda.Pointer.to(moment2.get(moment2.size()-1)),
          p0,
          jcuda.Pointer.to(new int[]{p})
          ));
    }

    hyperParam=Cuda.malloc(6*floatType.getBytes());
    if(floatType==FloatType.SINGLE)
    {
      Cuda.memcpyAsyncHostToDevice(hyperParam, new float[]{(float)alpha, (float)beta1, (float)beta2,
(float)epsilon, 1, 1});
    }
    else
    {
      Cuda.memcpyAsyncHostToDevice(hyperParam, new double[]{alpha, beta1, beta2, epsilon, 1, 1});
    }
    beta1T=1;
    beta2T=1;
  }

  public void destroy() throws CudaException
  {
    for(Pointer p: moment) if(p!=null) p.free();
    for(Pointer p: moment2) if(p!=null) p.free();
    if(hyperParam!=null) hyperParam.free();
  }

  public void iteration() throws CudaException, CudnnException
  {
    model.compGradient(driver, cudnn);
    if(beta1T>0||beta2T>0)
    {
      beta1T*=beta1;
      beta2T*=beta2;
      if(floatType==FloatType.SINGLE)
      {
        Cuda.memcpyAsyncHostToDevice(hyperParam.withByteOffset(floatType.getBytes()*4),
          new float[]{(float)beta1T, (float)beta2T});
      }
      else
      {
        Cuda.memcpyAsyncHostToDevice(hyperParam.withByteOffset(floatType.getBytes()*4),
          new double[]{beta1T, beta2T});
      }
    }

    int blockWidth=128;
    for(int la=0; la<model.getParamDev().size(); ++la)
    {
      int size=model.getParamSize().get(la);
      driver.call("Adam"+(floatType==FloatType.SINGLE?"Float":"Double"),
(int)Math.ceil((double)size/blockWidth), 1, 1, blockWidth, 1, 1, 0, jcuda.Pointer.to(
          jcuda.Pointer.to(model.getParamDev().get(la)),
          jcuda.Pointer.to(model.getGradDev().get(la)),
          jcuda.Pointer.to(moment.get(la)),
          jcuda.Pointer.to(moment2.get(la)),
          jcuda.Pointer.to(hyperParam),
          jcuda.Pointer.to(new int[]{size})
          ));
    }
  }
}
```

GradientDescent.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
```

```
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn.learner;

import cudnn.CudaDriver;
import cudnn.CudaException;
import cudnn.Cudnn;
import cudnn.CudnnException;
import jcuda.jcublas.JCublas;

/**
 * An implementation of weight updating by gradient decent.
 * {@link #iteration(float, CudaDriver, Cudnn)} is for single precision floating point computation, and {@link
#iteration(double, CudaDriver, Cudnn)} is for double precision floating point computation.
 * @author koumura
 *
 */
public class GradientDescent
{
  private Model model;

  public GradientDescent(Model model)
  {
    this.model = model;
  }

  public void iteration(double learningRate, CudaDriver driver, Cudnn cudnn) throws CudaException, CudnnException
  {
    model.compGradient(driver, cudnn);
    updateParam(learningRate);
  }
  public void iteration(float learningRate, CudaDriver driver, Cudnn cudnn) throws CudaException, CudnnException
  {
    model.compGradient(driver, cudnn);
    updateParam(learningRate);
  }

  private void updateParam(float learning_rate)
  {
    float alpha = -learning_rate;
    for(int i=0; i<model.getParamSize().size(); ++i)
    {
      JCublas.cublasSaxpy(model.getParamSize().get(i), alpha, model.getGradDev().get(i), 1,
model.getParamDev().get(i), 1);
    }
  }
  private void updateParam(double learning_rate)
  {
    double alpha = -learning_rate;
    for(int i=0; i<model.getParamSize().size(); ++i)
    {
      JCublas.cublasDaxpy(model.getParamSize().get(i), alpha, model.getGradDev().get(i), 1,
model.getParamDev().get(i), 1);
    }
  }


}
```

Model.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn.learner;

import java.util.ArrayList;

import cudnn.CudaDriver;
import cudnn.CudaException;
import cudnn.Cudnn;
import cudnn.CudnnException;
import cudnn.Pointer;

/**
 * A general model with trainable parameters.
 * @author koumura
 *
 */
public interface Model
{
  ArrayList<Pointer> getParamDev();
  ArrayList<Pointer> getGradDev();
  ArrayList<Integer> getParamSize();
  void compGradient(CudaDriver driver, Cudnn cudnn) throws CudaException, CudnnException;
}
```

SeqNetwork.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn.network;

import java.util.ArrayList;

import cudnn.Cuda;
import cudnn.CudaDriver;
import cudnn.CudaException;
import cudnn.Cudnn;
import cudnn.CudnnException;
import cudnn.FloatType;
import cudnn.IntType;
import cudnn.Pointer;
import cudnn.layer.ConvLayer;
import cudnn.layer.DataLayer;
import cudnn.layer.Layer;
import cudnn.layer.NonDataLayer;
import cudnn.layer.ParamLayer;
import cudnn.layer.PoolLayer;
import cudnn.layer.SeqSoftmaxConvLayer;
import cudnn.learner.Model;
import jcuda.jcublas.JCublas;

/**
 * A class for a convolutional neural network.
 * {@link #init(CudaDriver, Cudnn)} and {@link #cudaMalloc()} must be called before computation.
 * {@link #initError()} must be called before computing output errors.
 * {@link #destroy(Cudnn)} must be called before termination.
 * The type of the floating points (single or double) must be consistent throughout the computation. Eg. if {@link
FloatType#SINGLE} is given to the {@link #SeqNetwork(FloatType, ArrayList, int, Cudnn, IntType, int, int)}, {@link
#setDataF(ArrayList)} must be used to set data, but not {@link #setDataD(ArrayList)}.
 * **Dev means a pointer in a GPU.
 * **F means float vlaues. **D means double values.
 * @author koumura
 *
 */
public class SeqNetwork implements Model
{
  private FloatType floatType;
  private ArrayList<Layer> layer;
  private DataLayer dataLayer;
  private SeqSoftmaxConvLayer softMaxLayer;
  private int batchSize, workspaceSize;
  private Pointer workspaceDev;
  private ArrayList<Pointer> paramDev, gradDev;
  private ArrayList<Integer> paramSize;
  private Pointer dataDev, labelDev;
  private int labelShiftUpperH, labelShiftUpperW, labelShiftY, labelShiftX;
  private IntType labelType;
  private byte[] labelB;
  private int[] labelI;
  private ArrayList<float[]> dataF;
  private ArrayList<double[]> dataD;

  public SeqNetwork(FloatType floatType, ArrayList<Layer> layer, int batchSize, Cudnn cudnn, IntType labelType,
int labelShiftUpperH, int labelShiftUpperW) throws CudnnException
  {
    this.layer=layer;
    this.floatType=floatType;
    this.batchSize=batchSize;
    this.labelType=labelType;

    for(Layer la: layer)
    {
      if(la instanceof DataLayer) dataLayer=(DataLayer)la;
      if(la instanceof SeqSoftmaxConvLayer) softMaxLayer=(SeqSoftmaxConvLayer)la;
      la.init(floatType, cudnn, batchSize);

      if(la instanceof ParamLayer) ((ParamLayer)la).newWeightBias(floatType);
    }
    this.labelShiftUpperH=labelShiftUpperH;
    this.labelShiftUpperW=labelShiftUpperW;
    if(this.labelShiftUpperH<=0)
    {
      this.labelShiftUpperH=1;
      Layer la=softMaxLayer;
      while(la instanceof NonDataLayer)
      {
        if(la instanceof PoolLayer) this.labelShiftUpperH*=((PoolLayer) la).getStride();
        la=((NonDataLayer) la).getLower();
      }
    }
    if(this.labelShiftUpperW<=0)
    {
      this.labelShiftUpperW=1;
      Layer la=softMaxLayer;
      while(la instanceof NonDataLayer)
      {
        if(la instanceof PoolLayer) this.labelShiftUpperW*=((PoolLayer) la).getStride();
        la=((NonDataLayer) la).getLower();
      }
    }
    softMaxLayer.setLabelShiftUpper(this.labelShiftUpperH, this.labelShiftUpperW);
  }

  public int getLabelShiftUpperH() {
    return labelShiftUpperH;
  }

  public int getLabelShiftUpperW() {
    return labelShiftUpperW;
  }

  /**
   * @return sum, not average
   */
  public double compError(CudaDriver driver) throws CudaException
  {
    return softMaxLayer.compError(driver, labelDev, labelType, floatType);
  }
  public void initError() throws CudaException
  {
    softMaxLayer.initError(floatType);
  }

  public void init(CudaDriver driver, Cudnn cudnn) throws CudnnException, CudaException
  {
    workspaceSize=0;
    for(Layer la: layer) if(la instanceof ConvLayer)
    {
      ConvLayer conv=(ConvLayer)la;
      int sizeInBytes=conv.compWorkspaceSize(cudnn);
      if(sizeInBytes>workspaceSize) workspaceSize=sizeInBytes;
    }

    if (workspaceSize > 0)
    {
      workspaceDev=Cuda.malloc(workspaceSize);
```

```java
    }
    else workspaceDev=Pointer.createNull();

    for(Layer la: layer)
    {
      if(la instanceof ConvLayer) ((ConvLayer)la).setWodkspace(workspaceDev, workspaceSize);
    }
  }

  public void cudaMalloc() throws CudaException
  {
    for(Layer la: layer) la.cudaMalloc(floatType, batchSize);

    dataDev=Cuda.malloc(floatType.getBytes()*batchSize*dataLayer.sizeCHW());
    dataLayer.setDataDev(dataDev);

  labelDev=Cuda.malloc(labelType.getBytes()*batchSize*softMaxLayer.getLabelWidth()*softMaxLayer.getLabelHeight
());

    paramDev=new ArrayList<>();
    gradDev=new ArrayList<>();
    paramSize=new ArrayList<>();
    for(Layer la: layer) if(la instanceof ParamLayer)
    {
      ParamLayer pl=(ParamLayer)la;
      paramDev.add(pl.getWeightDev());
      gradDev.add(pl.getGradWeightDev());
      paramSize.add(pl.getWeightSize());
      paramDev.add(pl.getBiasDev());
      gradDev.add(pl.getGradBiasDev());
      paramSize.add(pl.getBiasSize());
    }
  }

  /**
   * Copies parameters from host to devide (GPU).
   * @throws CudaException
   */
  public void copyParamToDev() throws CudaException
  {
    for(Layer la: layer) if(la instanceof ParamLayer) ((ParamLayer)la).copyParamToDev(floatType);
  }

  /**
   * Copies parameters from device (GPU) to host.
   * @throws CudaException
   */
  public void copyParamFromDev() throws CudaException
  {
    for(Layer la: layer) if(la instanceof ParamLayer) ((ParamLayer)la).copyParamFromDev(floatType);
  }

  /**
   * Copies inputs from host to device (GPU).
   * @throws CudaException
   */
  public void copyDataToDevice() throws CudaException
  {
    if(floatType==FloatType.SINGLE)
    {
      int by=floatType.getBytes();
      for(int b=0; b<batchSize; ++b)
      {
        Cuda.memcpyAsyncHostToDevice(dataDev.withByteOffset(b*dataLayer.sizeHW()*by),
Pointer.to(dataF.get(b)).withByteOffset(labelShiftY*dataLayer.getWidth()*by), dataLayer.sizeHW()*by);
      }
    }
    else
    {
      int by=floatType.getBytes();
      for(int b=0; b<batchSize; ++b)
      {
        Cuda.memcpyAsyncHostToDevice(dataDev.withByteOffset(b*dataLayer.sizeHW()*by),
Pointer.to(dataD.get(b)).withByteOffset(labelShiftY*dataLayer.getWidth()*by), dataLayer.sizeHW()*by);
        }
      }
    }

  /**
   * Copies desired labels from host to device (GPU).
   * @throws CudaException
   */
  public void copyLabelToDevice(byte[] label) throws CudaException
  {
    if(label.length!=batchSize*softMaxLayer.getLabelHeight()*softMaxLayer.getLabelWidth())
System.err.println("label.length="+label.length+" !=
expected="+batchSize*softMaxLayer.getLabelHeight()*softMaxLayer.getLabelWidth());
    Cuda.memcpyAsyncHostToDevice(labelDev, label);
    this.labelB=label;
  }
  /**
   * Copies desired labels from host to device (GPU).
   * @throws CudaException
   */
  public void copyLabelToDevice(int[] label) throws CudaException
  {
    if(label.length!=batchSize*softMaxLayer.getLabelHeight()*softMaxLayer.getLabelWidth())
System.err.println("label.length="+label.length+" !=
expected="+batchSize*softMaxLayer.getLabelHeight()*softMaxLayer.getLabelWidth());
    Cuda.memcpyAsyncHostToDevice(labelDev, label);
    this.labelI=label;
  }

  public void destroy(Cudnn cudnn) throws CudaException, CudnnException
  {
    for(Layer la: layer) la.destroy(cudnn);
    if(workspaceDev.getJnaPointer()!=null) Cuda.free(workspaceDev);

    if(dataDev!=null) Cuda.free(dataDev);
    if(labelDev!=null) Cuda.free(labelDev);
  }

  public void compForward(CudaDriver driver, Cudnn cudnn) throws CudaException, CudnnException
  {
    for(Layer la: layer) if(la instanceof NonDataLayer)
    {
      NonDataLayer ndl=(NonDataLayer)la;
      ndl.compForward(floatType, cudnn, batchSize, driver);
    }
  }

  public void compBackward(CudaDriver driver, Cudnn cudnn) throws CudaException, CudnnException
  {
    {
      Cuda.memcpyAsyncDeviceToDevice(softMaxLayer.getDerActDev(), softMaxLayer.getValueDev(),
floatType.getBytes()* softMaxLayer.sizeCHW()*batchSize);

      int blockWidth=128;
      softMaxLayer.backwardCost(driver, floatType, labelDev, batchSize, blockWidth, labelType);

      int size;
      if(labelType==IntType.BYTE) size=softMaxLayer.countValidLabelSize(labelB, batchSize);
      else size=softMaxLayer.countValidLabelSize(labelI, batchSize);

      if(floatType==FloatType.SINGLE)
      {
        float scalVal=(float)(1d/size);
        JCublas.cublasSscal(softMaxLayer.sizeCHW() * batchSize, scalVal, softMaxLayer.getDerActDev(), 1);
      }
      else
      {
        double scalVal=1d/size;
        JCublas.cublasDscal(softMaxLayer.sizeCHW() * batchSize, scalVal, softMaxLayer.getDerActDev(), 1);
      }
    }
    for(int li=layer.size()-1; li>=0; --li)
    {
      Layer la=layer.get(li);
```

```java
      if(la instanceof NonDataLayer) ((NonDataLayer)la).compBackward(floatType, cudnn, batchSize, driver);
    }
  }

  public ArrayList<Pointer> getParamDev() {
    return paramDev;
  }

  public ArrayList<Pointer> getGradDev() {
    return gradDev;
  }

  public ArrayList<Integer> getParamSize() {
    return paramSize;
  }

  @Override
  public void compGradient(CudaDriver driver, Cudnn cudnn) throws CudaException, CudnnException
  {
    compForward(driver, cudnn);
    compBackward(driver, cudnn);
  }

  public void setDataF(ArrayList<float[]> data) {
    this.dataF = data;
  }

  public void setDataD(ArrayList<double[]> data) {
    this.dataD = data;
  }

  public void setLabelShift(int labelShiftY, int labelShiftX)
  {
    this.labelShiftY = labelShiftY;
    this.labelShiftX = labelShiftX;
    softMaxLayer.setLabelShift(labelShiftY, labelShiftX);
  }

  public int labelIndex(int batch, int y, int x, int singleY, int singleX)
  {
    return softMaxLayer.labelIndex(batch, y, x, singleY, singleX);
  }

  public ArrayList<Layer> getLayer() {
    return layer;
  }

  /**
   * A label is valid if and only if label>=0
   * @return int[labelShiftUpperH*labelShiftUpperW]
   */
  public int[] countValidLabelSize(byte[] label)
  {
    int[] validSize=new int[labelShiftUpperH*labelShiftUpperW];
    for(int shiftY=0; shiftY<labelShiftUpperH; ++shiftY) for(int shiftX=0; shiftX<labelShiftUpperW; ++shiftX)
    {
      setLabelShift(shiftY, shiftX);
      for(int b=0; b<batchSize; ++b)
        for(int y=0; y<softMaxLayer.getHeight(); ++y) for(int x=0; x<softMaxLayer.getWidth(); ++x)
          for(int sy=0; sy<softMaxLayer.getSingleHeight(); ++sy) for(int sx=0; sx<softMaxLayer.getSingleWidth();
++sx)
            if(label[labelIndex(b, y, x, sy, sx)]>=0) ++validSize[shiftY*labelShiftUpperW+shiftX];
    }
    return validSize;
  }
}
```

ActivationMode.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn;

/**
 * Activation functions.
 * @author koumura
 *
 */
public enum ActivationMode
{
  SIGMOID(CudnnLibrary.cudnnActivationMode_t.CUDNN_ACTIVATION_SIGMOID),
  RELU(CudnnLibrary.cudnnActivationMode_t.CUDNN_ACTIVATION_RELU),
  TANH(CudnnLibrary.cudnnActivationMode_t.CUDNN_ACTIVATION_TANH),
  IDENT(-1);

  private int value;

  private ActivationMode(int value) {
    this.value = value;
  }

  public int getValue() {
    return value;
  }
}
```

Cuda.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn;

import jcuda.runtime.JCuda;
import jcuda.runtime.cudaError;
import jcuda.runtime.cudaMemcpyKind;
import jcuda.Pointer;

/**
 * A class for cuda functions.
 * @author koumura
 *
 */
```

```java
public class Cuda
{
  public static void setDevice(int device) throws CudaException
  {
    checkError(JCuda.cudaSetDevice(device));
  }

  public static void checkError(int status) throws CudaException
  {
    if(status!=cudaError.cudaSuccess)
    {
      throw new CudaException(cudaError.stringFor(status));
    }
  }

  /**
   * @param size in bytes
   */
  public static cudnn.Pointer malloc(long size) throws CudaException
  {
    Pointer deviceData=new Pointer();
    checkError(JCuda.cudaMalloc(deviceData, size));
    return cudnn.Pointer.fromJcuda(deviceData);
  }

  public static void memcpyAsyncHostToDevice(Pointer deviceData, Pointer hostData, long size) throws CudaException
  {
    checkError(JCuda.cudaMemcpyAsync(deviceData, hostData, size, cudaMemcpyKind.cudaMemcpyHostToDevice, null));
  }
  public static void memcpyAsyncHostToDevice(Pointer deviceData, int[] hostData) throws CudaException
  {
    memcpyAsyncHostToDevice(deviceData, Pointer.to(hostData), Integer.BYTES*hostData.length);
  }
  public static void memcpyAsyncHostToDevice(Pointer deviceData, byte[] hostData) throws CudaException
  {
    memcpyAsyncHostToDevice(deviceData, Pointer.to(hostData), hostData.length);
  }
  public static void memcpyAsyncHostToDevice(Pointer deviceData, float[] hostData) throws CudaException
  {
    memcpyAsyncHostToDevice(deviceData, Pointer.to(hostData), Float.BYTES*hostData.length);
  }
  public static void memcpyAsyncHostToDevice(Pointer deviceData, double[] hostData) throws CudaException
  {
    memcpyAsyncHostToDevice(deviceData, Pointer.to(hostData), Double.BYTES*hostData.length);
  }

  public static void memcpyAsyncDeviceToHost(Pointer hostData, Pointer deviceData, long size) throws CudaException
  {
    checkError(JCuda.cudaMemcpyAsync(hostData, deviceData, size, cudaMemcpyKind.cudaMemcpyDeviceToHost, null));
  }

  public static void memcpyAsyncDeviceToDevice(Pointer dst, Pointer src, long size) throws CudaException
  {
    checkError(JCuda.cudaMemcpyAsync(dst, src, size, cudaMemcpyKind.cudaMemcpyDeviceToDevice, null));
  }

  public static void memcpyDeviceToHost(Pointer hostData, Pointer deviceData, long size) throws CudaException
  {
    checkError(JCuda.cudaMemcpy(hostData, deviceData, size, cudaMemcpyKind.cudaMemcpyDeviceToHost));
  }

  public static int[] memcpyDeviceToIntArray(Pointer deviceData, int size) throws CudaException
  {
    int[] host=new int[size];
    memcpyDeviceToHost(Pointer.to(host), deviceData, Integer.BYTES*size);
    return host;
  }

  public static byte[] memcpyDeviceToByteArray(Pointer deviceData, int size) throws CudaException
  {
    byte[] host=new byte[size];
    memcpyDeviceToHost(Pointer.to(host), deviceData, Byte.BYTES*size);
    return host;
  }

  public static float[] memcpyDeviceToFloatArray(Pointer deviceData, int size) throws CudaException
  {
    float[] host=new float[size];
    memcpyDeviceToHost(Pointer.to(host), deviceData, Float.BYTES*size);
    return host;
  }

  public static double[] memcpyDeviceToDoubleArray(Pointer deviceData, int size) throws CudaException
  {
    double[] host=new double[size];
    memcpyDeviceToHost(Pointer.to(host), deviceData, Double.BYTES*size);
    return host;
  }

  public static void memcpyDeviceToDevice(Pointer dst, Pointer src, long size) throws CudaException
  {
    checkError(JCuda.cudaMemcpy(dst, src, size, cudaMemcpyKind.cudaMemcpyDeviceToDevice));
  }

  public static void memcpyHostToDevice(Pointer deviceData, Pointer hostData, long size) throws CudaException
  {
    checkError(JCuda.cudaMemcpy(deviceData, hostData, size, cudaMemcpyKind.cudaMemcpyHostToDevice));
  }
  public static void memcpyHostToDevice(Pointer deviceData, int[] hostData) throws CudaException
  {
    memcpyHostToDevice(deviceData, Pointer.to(hostData), Integer.BYTES*hostData.length);
  }
  public static void memcpyHostToDevice(Pointer deviceData, byte[] hostData) throws CudaException
  {
    memcpyHostToDevice(deviceData, Pointer.to(hostData), Byte.BYTES*hostData.length);
  }
  public static void memcpyHostToDevice(Pointer deviceData, float[] hostData) throws CudaException
  {
    memcpyHostToDevice(deviceData, Pointer.to(hostData), Float.BYTES*hostData.length);
  }
  public static void memcpyHostToDevice(Pointer deviceData, double[] hostData) throws CudaException
  {
    memcpyHostToDevice(deviceData, Pointer.to(hostData), Double.BYTES*hostData.length);
  }

  public static void free(Pointer deviceData) throws CudaException
  {
    checkError(JCuda.cudaFree(deviceData));
  }

  public static void deviceSynchronize() throws CudaException
  {
    checkError(JCuda.cudaDeviceSynchronize());
  }

  public static void deviceReset() throws CudaException
  {
    checkError(JCuda.cudaDeviceReset());
  }
}
```

CudaDriver.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn;

import static jcuda.driver.JCudaDriver.cuCtxSynchronize;
import static jcuda.driver.JCudaDriver.cuLaunchKernel;
import static jcuda.driver.JCudaDriver.cuModuleGetFunction;

import java.nio.file.Path;

import jcuda.Pointer;
import jcuda.driver.CUcontext;
import jcuda.driver.CUdevice;
import jcuda.driver.CUfunction;
import jcuda.driver.CUmodule;
import jcuda.driver.JCudaDriver;

/**
 * A class to call custom-written kernels.
 * @author koumura
 *
 */
public class CudaDriver
{
  private CUmodule module;
  private CUcontext context;

  public CudaDriver() throws CudaException
  {
    JCudaDriver.cuInit(0);
    CUdevice device = new CUdevice();
    checkError(JCudaDriver.cuDeviceGet(device, 0));
    context = new CUcontext();
    checkError(JCudaDriver.cuCtxCreate(context, 0, device));
    module = new CUmodule();
  }
  public CudaDriver(Path path) throws CudaException
  {
    this();
    load(path.toAbsolutePath().toString());
  }

  public void destroy() throws CudaException
  {
    checkError(JCudaDriver.cuCtxDestroy(context));
  }

  private static void checkError(int status) throws CudaException
  {
    Cuda.checkError(status);
  }

  public void load(String ptxPath) throws CudaException
  {
    checkError(JCudaDriver.cuModuleLoad(module, ptxPath));
  }

  /*public static CUdeviceptr malloc(int size) throws CudaException
  {
    CUdeviceptr pointer= new CUdeviceptr();
    checkError(JCudaDriver.cuMemAlloc(pointer, size));
    return pointer;
  }

  public static void free(CUdeviceptr pointer) throws CudaException
  {
    checkError(JCudaDriver.cuMemFree(pointer));
  }*/

  public CUfunction getFunction(String name) throws CudaException
  {
      CUfunction function = new CUfunction();
      cuModuleGetFunction(function, module, name);
      return function;
  }

  public static void call(CUfunction function, int gridDimX, int gridDimY, int gridDimZ, int blockDimX, int blockDimY,
int blockDimZ, int sharedMemBytes, Pointer kernelParams) throws CudaException
  {
    checkError(cuLaunchKernel(function, gridDimX, gridDimY, gridDimZ, blockDimX, blockDimY, blockDimZ,
sharedMemBytes, null, kernelParams, null));
  }
  public void call(String function, int gridDimX, int gridDimY, int gridDimZ, int blockDimX, int blockDimY, int
blockDimZ, int sharedMemBytes, Pointer kernelParams) throws CudaException
  {
    call(getFunction(function), gridDimX, gridDimY, gridDimZ, blockDimX, blockDimY, blockDimZ, sharedMemBytes,
kernelParams);
  }

  public static void ctxSynchronize() throws CudaException
  {
    checkError(cuCtxSynchronize());
  }

  /*public static void memcpyDeviceToHost(Pointer dstHost, CUdeviceptr srcDevice, int size) throws CudaException
  {
    checkError(JCudaDriver.cuMemcpyDtoH(dstHost, srcDevice, size));
  }

  public static void memcpyAsyncHostToDevice(CUdeviceptr dstDevice, Pointer srcHost, int size) throws CudaException
  {
    checkError(JCudaDriver.cuMemcpyHtoDAsync(dstDevice, srcHost, size, null));
  }*/
}
```

CudaException.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn;
```

```java
public class CudaException extends Exception {

  public CudaException() {
    super();
    // TODO Auto-generated constructor stub
  }

  public CudaException(String arg0, Throwable arg1, boolean arg2, boolean arg3) {
    super(arg0, arg1, arg2, arg3);
    // TODO Auto-generated constructor stub
  }

  public CudaException(String arg0, Throwable arg1) {
    super(arg0, arg1);
    // TODO Auto-generated constructor stub
  }

  public CudaException(String arg0) {
    super(arg0);
    // TODO Auto-generated constructor stub
  }

  public CudaException(Throwable arg0) {
    super(arg0);
    // TODO Auto-generated constructor stub
  }

}
```

Cudnn.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn;


import java.nio.file.Path;

import com.sun.jna.Pointer;
import com.sun.jna.ptr.ByReference;
import com.sun.jna.ptr.DoubleByReference;
import com.sun.jna.ptr.FloatByReference;
import com.sun.jna.ptr.IntByReference;
import com.sun.jna.ptr.PointerByReference;

import cudnn.CudnnLibrary.cudnnConvolutionDescriptor_t;
import cudnn.CudnnLibrary.cudnnFilterDescriptor_t;
import cudnn.CudnnLibrary.cudnnTensorDescriptor_t;
import cudnn.layer.Layer;

/**
 * A class to handle the cudnn library.
 * @author koumura
 *
 */
public class Cudnn
{
  CudnnLibrary.Instance library;
  CudnnLibrary.cudnnHandle_t handle;

  public static final FloatByReference FLOAT_1=new FloatByReference(1);
  public static final FloatByReference FLOAT_0=new FloatByReference(0);
  public static final DoubleByReference DOUBLE_1=new DoubleByReference(1);
  public static final DoubleByReference DOUBLE_0=new DoubleByReference(0);

  public static ByReference const1(FloatType floatType)
  {
    if(floatType==FloatType.SINGLE) return FLOAT_1;
    else return DOUBLE_1;
  }

  public static ByReference const0(FloatType floatType)
  {
    if(floatType==FloatType.SINGLE) return FLOAT_0;
    else return DOUBLE_0;
  }

  public Cudnn(Path libraryPath) throws CudnnException
  {
    library=new CudnnLibrary.Instance(libraryPath);
    PointerByReference handle=new PointerByReference();
    checkError(library.get().cudnnCreate(handle));
    this.handle=new CudnnLibrary.cudnnHandle_t(handle.getValue());
  }

  public void destroty() throws CudnnException
  {
    checkError(library.get().cudnnDestroy(handle));
  }

  public void initTensorDesc(FloatType floatType, Layer layer, int batchSize) throws CudnnException
  {
    layer.getTensorDesc().setTensor4dDescriptor(this, CudnnLibrary.cudnnTensorFormat_t.CUDNN_TENSOR_NCHW,
floatType.getDataTypeValue(), batchSize, layer.getNumChannel(), layer.getHeight(), layer.getWidth());
  }

  void checkError(int status) throws CudnnException
  {
    if(status!=CudnnLibrary.cudnnStatus_t.CUDNN_STATUS_SUCCESS)
    {
      throw new CudnnException(library.get().cudnnGetErrorString(status));
    }
  }

  public CudnnLibrary.cudnnTensorDescriptor_t createTensorDescriptor() throws CudnnException
  {
    PointerByReference desc=new PointerByReference();
    checkError(library.get().cudnnCreateTensorDescriptor(desc));
    return new CudnnLibrary.cudnnTensorDescriptor_t(desc.getValue());
  }

  public CudnnLibrary.cudnnFilterDescriptor_t createFilterDescriptor() throws CudnnException
  {
    PointerByReference desc=new PointerByReference();
    checkError(library.get().cudnnCreateFilterDescriptor(desc));
    return new CudnnLibrary.cudnnFilterDescriptor_t(desc.getValue());
  }

  public CudnnLibrary.cudnnConvolutionDescriptor_t createConvolutionDescriptor() throws CudnnException
  {
    PointerByReference desc=new PointerByReference();
    checkError(library.get().cudnnCreateConvolutionDescriptor(desc));
    return new CudnnLibrary.cudnnConvolutionDescriptor_t(desc.getValue());
  }

  public CudnnLibrary.cudnnPoolingDescriptor_t createPoolingDescriptor() throws CudnnException
  {
    PointerByReference desc=new PointerByReference();
    checkError(library.get().cudnnCreatePoolingDescriptor(desc));
    return new CudnnLibrary.cudnnPoolingDescriptor_t(desc.getValue());
```

```
  }

  /**
   * Helper function to return the dimensions of the output tensor given a convolution descriptor<br>
   * Original signature : <code>cudnnStatus_t cudnnGetConvolution2dForwardOutputDim(const
cudnnConvolutionDescriptor_t, const cudnnTensorDescriptor_t, const cudnnFilterDescriptor_t, int*, int*, int*,
int*)</code><br>
   * <i>native declaration : line 322</i>
   * @throws CudnnException
   */
  public void getConvolution2dForwardOutputDim(CudnnLibrary.cudnnConvolutionDescriptor_t convDesc,
CudnnLibrary.cudnnTensorDescriptor_t inputTensorDesc, CudnnLibrary.cudnnFilterDescriptor_t filterDesc,
IntByReference n, IntByReference c, IntByReference h, IntByReference w) throws CudnnException
  {
    checkError(library.get().cudnnGetConvolution2dForwardOutputDim(convDesc, inputTensorDesc, filterDesc, n, c,
h, w));
  }

  /**
   * Original signature : <code>cudnnStatus_t cudnnGetConvolutionForwardAlgorithm(cudnnHandle_t, const
cudnnTensorDescriptor_t, const cudnnFilterDescriptor_t, const cudnnConvolutionDescriptor_t, const
cudnnTensorDescriptor_t, cudnnConvolutionFwdPreference_t, size_t, cudnnConvolutionFwdAlgo_t*)</code><br>
   * <i>native declaration : line 379</i>
   * @throws CudnnException
   */
  public int getConvolutionForwardAlgorithm(CudnnLibrary.cudnnTensorDescriptor_t srcDesc,
CudnnLibrary.cudnnFilterDescriptor_t filterDesc, CudnnLibrary.cudnnConvolutionDescriptor_t convDesc,
CudnnLibrary.cudnnTensorDescriptor_t destDesc, int preference, int memoryLimitInbytes) throws CudnnException
  {
    IntByReference algo=new IntByReference();
    checkError(library.get().cudnnGetConvolutionForwardAlgorithm(this.handle, srcDesc, filterDesc, convDesc,
destDesc, preference, memoryLimitInbytes, algo));
    return algo.getValue();
  }

  /**
   * Helper function to return the minimum size of the workspace to be passed to the convolution given an algo<br>
   * Original signature : <code>cudnnStatus_t cudnnGetConvolutionForwardWorkspaceSize(cudnnHandle_t, const
cudnnTensorDescriptor_t, const cudnnFilterDescriptor_t, const cudnnConvolutionDescriptor_t, const
cudnnTensorDescriptor_t, cudnnConvolutionFwdAlgo_t, size_t*)</code><br>
   * <i>native declaration : line 394</i>
   * @throws CudnnException
   */
  public int getConvolutionForwardWorkspaceSize(CudnnLibrary.cudnnTensorDescriptor_t srcDesc,
CudnnLibrary.cudnnFilterDescriptor_t filterDesc, CudnnLibrary.cudnnConvolutionDescriptor_t convDesc,
CudnnLibrary.cudnnTensorDescriptor_t destDesc, int algo) throws CudnnException
  {
    IntByReference sizeInBytes=new IntByReference();
    checkError(library.get().cudnnGetConvolutionForwardWorkspaceSize(handle, srcDesc, filterDesc, convDesc,
destDesc, algo, sizeInBytes));
    return sizeInBytes.getValue();
  }

  /**
   * Function to perform the forward multiconvolution<br>
   * Original signature : <code>cudnnStatus_t cudnnConvolutionForward(cudnnHandle_t, const void*, const
cudnnTensorDescriptor_t, const void*, const cudnnFilterDescriptor_t, const void*, const
cudnnConvolutionDescriptor_t, cudnnConvolutionFwdAlgo_t, void*, size_t, const void*, const cudnnTensorDescriptor_t,
void*)</code><br>
   * <i>native declaration : line 407</i>
   * @throws CudnnException
   */
  public void convolutionForward(ByReference alpha, CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer srcData,
CudnnLibrary.cudnnFilterDescriptor_t filterDesc, Pointer filterData, CudnnLibrary.cudnnConvolutionDescriptor_t
convDesc, int algo, Pointer workSpace, int workSpaceSizeInBytes, ByReference beta,
CudnnLibrary.cudnnTensorDescriptor_t destDesc, Pointer destData) throws CudnnException
  {
    checkError(library.get().cudnnConvolutionForward(handle, alpha, srcDesc, srcData, filterDesc, filterData,
convDesc, algo, workSpace, workSpaceSizeInBytes, beta, destDesc, destData));
  }

  /**
   * Tensor Bias addition : srcDest = alpha * bias + beta * srcDestDesc<br>
   * Original signature : <code>cudnnStatus_t cudnnAddTensor(cudnnHandle_t, cudnnAddMode_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, cudnnTensorDescriptor_t, void*)</code><br>
   * <i>native declaration : line 229</i>
   * @throws CudnnException
   */
  public void addTensor(int mode, ByReference alpha, CudnnLibrary.cudnnTensorDescriptor_t biasDesc, Pointer
biasData, ByReference beta, CudnnLibrary.cudnnTensorDescriptor_t srcDestDesc, Pointer srcDestData) throws
CudnnException
  {
    checkError(library.get().cudnnAddTensor(handle, mode, alpha, biasDesc, biasData, beta, srcDestDesc,
srcDestData));
  }

  /**
   * Function to perform forward pooling<br>
   * Original signature : <code>cudnnStatus_t cudnnPoolingForward(cudnnHandle_t, const cudnnPoolingDescriptor_t,
const void*, const cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t,
void*)</code><br>
   * <i>native declaration : line 581</i>
   * @throws CudnnException
   */
  public void poolingForward(CudnnLibrary.cudnnPoolingDescriptor_t poolingDesc, ByReference alpha,
CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer srcData, ByReference beta,
CudnnLibrary.cudnnTensorDescriptor_t destDesc, Pointer destData) throws CudnnException
  {
    checkError(library.get().cudnnPoolingForward(handle, poolingDesc, alpha, srcDesc, srcData, beta, destDesc,
destData));
  }

  /**
   * Function to perform forward activation<br>
   * Original signature : <code>cudnnStatus_t cudnnActivationForward(cudnnHandle_t, cudnnActivationMode_t, const
void*, const cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
   * <i>native declaration : line 619</i>
   * @throws CudnnException
   */
  public void activationForward(int mode, ByReference alpha, CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer
srcData, ByReference beta, CudnnLibrary.cudnnTensorDescriptor_t destDesc, Pointer destData) throws CudnnException
  {
    checkError(library.get().cudnnActivationForward(handle, mode, alpha, srcDesc, srcData, beta, destDesc,
destData));
  }

  /**
   * Function to perform forward softmax<br>
   * Original signature : <code>cudnnStatus_t cudnnSoftmaxForward(cudnnHandle_t, cudnnSoftmaxAlgorithm_t,
cudnnSoftmaxMode_t, const void*, const cudnnTensorDescriptor_t, const void*, const void*, const
cudnnTensorDescriptor_t, void*)</code><br>
   * <i>native declaration : line 487</i>
   * @throws CudnnException
   */
  public void softmaxForward(int algorithm, int mode, ByReference alpha, CudnnLibrary.cudnnTensorDescriptor_t
srcDesc, Pointer srcData, ByReference beta, CudnnLibrary.cudnnTensorDescriptor_t destDesc, Pointer destData) throws
CudnnException
  {
    checkError(library.get().cudnnSoftmaxForward(handle, algorithm, mode, alpha, srcDesc, srcData, beta, destDesc,
destData));
  }

  /**
   * Function to perform backward activation<br>
   * Original signature : <code>cudnnStatus_t cudnnActivationBackward(cudnnHandle_t, cudnnActivationMode_t, const
void*, const cudnnTensorDescriptor_t, const void*, const cudnnTensorDescriptor_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
   * <i>native declaration : line 630</i>
   * @throws CudnnException
   */
  public void activationBackward(int mode, ByReference alpha, CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer
srcData, CudnnLibrary.cudnnTensorDescriptor_t srcDiffDesc, Pointer srcDiffData,
CudnnLibrary.cudnnTensorDescriptor_t destDesc, Pointer destData, ByReference beta,
CudnnLibrary.cudnnTensorDescriptor_t destDiffDesc, Pointer destDiffData) throws CudnnException
  {
    checkError(library.get().cudnnActivationBackward(handle, mode, alpha, srcDesc, srcData, srcDiffDesc,
srcDiffData, destDesc, destData, beta, destDiffDesc, destDiffData));
```

```java
  }

  /**
   * Function to perform backward pooling<br>
   * Original signature : <code>cudnnStatus_t cudnnPoolingBackward(cudnnHandle_t, const cudnnPoolingDescriptor_t,
const void*, const cudnnTensorDescriptor_t, const void*, const cudnnTensorDescriptor_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
   * <i>native declaration : line 592</i>
   * @throws CudnnException
   */
  public void poolingBackward(CudnnLibrary.cudnnPoolingDescriptor_t poolingDesc, ByReference alpha,
CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer srcData, CudnnLibrary.cudnnTensorDescriptor_t srcDiffDesc,
Pointer srcDiffData, CudnnLibrary.cudnnTensorDescriptor_t destDesc, Pointer destData, ByReference beta,
CudnnLibrary.cudnnTensorDescriptor_t destDiffDesc, Pointer destDiffData) throws CudnnException
  {
    checkError(library.get().cudnnPoolingBackward(handle, poolingDesc, alpha, srcDesc, srcData, srcDiffDesc,
srcDiffData, destDesc, destData, beta, destDiffDesc, destDiffData));
  }

  /**
   * Functions to perform the backward multiconvolution<br>
   * Original signature : <code>cudnnStatus_t cudnnConvolutionBackwardBias(cudnnHandle_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
   * <i>native declaration : line 423</i>
   * @throws CudnnException
   */
  public void convolutionBackwardBias(ByReference alpha, CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer
srcData, ByReference beta, CudnnLibrary.cudnnTensorDescriptor_t destDesc, Pointer destData) throws CudnnException
  {
    checkError(library.get().cudnnConvolutionBackwardBias(handle, alpha, srcDesc, srcData, beta, destDesc,
destData));
  }

  /**
   * Original signature : <code>cudnnStatus_t cudnnConvolutionBackwardFilter(cudnnHandle_t, const void*, const
cudnnTensorDescriptor_t, const void*, const cudnnTensorDescriptor_t, const void*, const
cudnnConvolutionDescriptor_t, const void*, const cudnnFilterDescriptor_t, void*)</code><br>
   * <i>native declaration : line 434</i>
   * @throws CudnnException
   */
  public void convolutionBackwardFilter(ByReference alpha, CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer
srcData, CudnnLibrary.cudnnTensorDescriptor_t diffDesc, Pointer diffData,
CudnnLibrary.cudnnConvolutionDescriptor_t convDesc, ByReference beta, CudnnLibrary.cudnnFilterDescriptor_t
gradDesc, Pointer gradData) throws CudnnException
  {
    checkError(library.get().cudnnConvolutionBackwardFilter(handle, alpha, srcDesc, srcData, diffDesc, diffData,
convDesc, beta, gradDesc, gradData));
  }

  /**
   * Original signature : <code>cudnnStatus_t cudnnConvolutionBackwardData(cudnnHandle_t, const void*, const
cudnnFilterDescriptor_t, const void*, const cudnnTensorDescriptor_t, const void*, const
cudnnConvolutionDescriptor_t, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
   * <i>native declaration : line 447</i>
   * @throws CudnnException
   */
  public void convolutionBackwardData(ByReference alpha, CudnnLibrary.cudnnFilterDescriptor_t filterDesc, Pointer
filterData, CudnnLibrary.cudnnTensorDescriptor_t diffDesc, Pointer diffData,
CudnnLibrary.cudnnConvolutionDescriptor_t convDesc, ByReference beta, CudnnLibrary.cudnnTensorDescriptor_t
gradDesc, Pointer gradData) throws CudnnException
  {
    checkError(library.get().cudnnConvolutionBackwardData(handle, alpha, filterDesc, filterData, diffDesc,
diffData, convDesc, beta, gradDesc, gradData));
  }

  public CudnnLibrary.Instance getLibrary() {
    return library;
  }


///**
// * Tensor Bias addition : srcDest = alpha * bias + beta * srcDestDesc<br>
// * Original signature : <code>cudnnStatus_t cudnnAddTensor(cudnnHandle_t, cudnnAddMode_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, cudnnTensorDescriptor_t, void*)</code><br>
// * <i>native declaration : line 229</i>
// */
//public void addTensor(CudnnLibrary.cudnnHandle_t handle, int mode, Pointer alpha,
CudnnLibrary.cudnnTensorDescriptor_t biasDesc, Pointer biasData, Pointer beta,
CudnnLibrary.cudnnTensorDescriptor_t srcDestDesc, Pointer srcDestData)
//{
//  int cudnnAddTensor(CudnnLibrary.cudnnHandle_t handle, int mode, Pointer alpha,
CudnnLibrary.cudnnTensorDescriptor_t biasDesc, Pointer biasData, Pointer beta,
CudnnLibrary.cudnnTensorDescriptor_t srcDestDesc, Pointer srcDestData);
//}

  //v3
  public int getConvolutionBackwardFilterAlgorithm(
      cudnnTensorDescriptor_t srcDesc,
      cudnnTensorDescriptor_t diffDesc,
      cudnnConvolutionDescriptor_t convDesc,
      cudnnFilterDescriptor_t gradDesc,
      int preference,
      int memoryLimitInbytes) throws CudnnException
  {
    IntByReference algo=new IntByReference();
    checkError(library.get().cudnnGetConvolutionBackwardFilterAlgorithm(
        handle,
        srcDesc,
        diffDesc,
        convDesc,
        gradDesc,
        preference,
        memoryLimitInbytes,
        algo));
    return algo.getValue();
  }

  public void convolutionBackwardFilter(
      ByReference alpha,
      cudnnTensorDescriptor_t srcDesc,
      Pointer srcData,
      cudnnTensorDescriptor_t diffDesc,
      Pointer diffData,
      cudnnConvolutionDescriptor_t convDesc,
      int algo,
      Pointer workSpace,
      int workSpaceSizeInBytes,
      ByReference beta,
      cudnnFilterDescriptor_t gradDesc,
      Pointer gradData) throws CudnnException
  {
    checkError(library.get().cudnnConvolutionBackwardFilter_v3(
        handle,
        alpha,
        srcDesc,
        srcData,
        diffDesc,
        diffData,
        convDesc,
        algo,
        workSpace,
        workSpaceSizeInBytes,
        beta,
        gradDesc,
        gradData));
  }

  public void convolutionBackwardData(
      ByReference alpha,
      cudnnFilterDescriptor_t filterDesc,
      Pointer filterData,
      cudnnTensorDescriptor_t diffDesc,
      Pointer diffData,
      cudnnConvolutionDescriptor_t convDesc,
      int algo,
      Pointer workSpace,
      int workSpaceSizeInBytes,
      ByReference beta,
```

```
      cudnnTensorDescriptor_t gradDesc,
      Pointer gradData) throws CudnnException
  {
    checkError(library.get().cudnnConvolutionBackwardData_v3(
        handle,
        alpha,
        filterDesc,
        filterData,
        diffDesc,
        diffData,
        convDesc,
        algo,
        workSpace,
        workSpaceSizeInBytes,
        beta,
        gradDesc,
        gradData));
  }

  public int convolutionBackwardFilterWorkspaceSize(
      cudnnTensorDescriptor_t srcDesc,
      cudnnTensorDescriptor_t diffDesc,
      cudnnConvolutionDescriptor_t convDesc,
      cudnnFilterDescriptor_t gradDesc,
      int algo) throws CudnnException
  {
    IntByReference size=new IntByReference();
    checkError(library.get().cudnnGetConvolutionBackwardFilterWorkspaceSize(
        handle,
        srcDesc,
        diffDesc,
        convDesc,
        gradDesc,
        algo,
        size));
    return size.getValue();
  }

  public int convolutionBackwardDataWorkspaceSize(
      cudnnFilterDescriptor_t filterDesc,
      cudnnTensorDescriptor_t diffDesc,
      cudnnConvolutionDescriptor_t convDesc,
      cudnnTensorDescriptor_t gradDesc,
      int algo) throws CudnnException
  {
    IntByReference size=new IntByReference();
    checkError(library.get().cudnnGetConvolutionBackwardDataWorkspaceSize(
        handle,
        filterDesc,
        diffDesc,
        convDesc,
        gradDesc,
        algo,
        size));
    return size.getValue();
  }
}
```

CudnnException.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn;

public class CudnnException extends Exception {

  public CudnnException() {
    super();
    // TODO Auto-generated constructor stub
  }

  public CudnnException(String arg0, Throwable arg1, boolean arg2, boolean arg3) {
    super(arg0, arg1, arg2, arg3);
    // TODO Auto-generated constructor stub
  }

  public CudnnException(String arg0, Throwable arg1) {
    super(arg0, arg1);
    // TODO Auto-generated constructor stub
  }

  public CudnnException(String arg0) {
    super(arg0);
    // TODO Auto-generated constructor stub
  }

  public CudnnException(Throwable arg0) {
    super(arg0);
    // TODO Auto-generated constructor stub
  }

}
```

CudnnLibrary.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn;
import java.nio.file.Path;

import com.sun.jna.Native;
import com.sun.jna.Pointer;
import com.sun.jna.PointerType;
import com.sun.jna.ptr.ByReference;
import com.sun.jna.ptr.IntByReference;
import com.sun.jna.ptr.PointerByReference;
import com.sun.jna.win32.StdCallLibrary;
/**
 * JNA Wrapper for library <b>cudnn</b><br>
 * This file was autogenerated by <a href="http://jnaerator.googlecode.com/">JNAerator</a>,<br>
 * a tool written by <a href="http://ochafik.com/">Olivier Chafik</a> that <a
href="http://code.google.com/p/jnaerator/wiki/CreditsAndLicense">uses a few opensource projects.</a>.<br>
```

```java
 * For help, please visit <a href="http://nativelibs4java.googlecode.com/">NativeLibs4Java</a> , <a
href="http://rococoa.dev.java.net/">Rococoa</a>, or <a href="http://jna.dev.java.net/">JNA</a>.
 */
public interface CudnnLibrary extends StdCallLibrary
{
  public static class Instance
  {
    private Path libraryPath;
    private CudnnLibrary instance;

    public Instance(Path libraryPath)
    {
      this.libraryPath = libraryPath.toAbsolutePath();
      instance = (CudnnLibrary)Native.loadLibrary(libraryPath.toString(), CudnnLibrary.class);
    }

    public CudnnLibrary get() {return instance;}
  }

  /** enum values */
  public static interface cudnnStatus_t {
    /** <i>native declaration : line 83</i> */
    public static final int CUDNN_STATUS_SUCCESS = 0;
    /** <i>native declaration : line 84</i> */
    public static final int CUDNN_STATUS_NOT_INITIALIZED = 1;
    /** <i>native declaration : line 85</i> */
    public static final int CUDNN_STATUS_ALLOC_FAILED = 2;
    /** <i>native declaration : line 86</i> */
    public static final int CUDNN_STATUS_BAD_PARAM = 3;
    /** <i>native declaration : line 87</i> */
    public static final int CUDNN_STATUS_INTERNAL_ERROR = 4;
    /** <i>native declaration : line 88</i> */
    public static final int CUDNN_STATUS_INVALID_VALUE = 5;
    /** <i>native declaration : line 89</i> */
    public static final int CUDNN_STATUS_ARCH_MISMATCH = 6;
    /** <i>native declaration : line 90</i> */
    public static final int CUDNN_STATUS_MAPPING_ERROR = 7;
    /** <i>native declaration : line 91</i> */
    public static final int CUDNN_STATUS_EXECUTION_FAILED = 8;
    /** <i>native declaration : line 92</i> */
    public static final int CUDNN_STATUS_NOT_SUPPORTED = 9;
    /** <i>native declaration : line 93</i> */
    public static final int CUDNN_STATUS_LICENSE_ERROR = 10;
  };
  /** enum values */
  public static interface cudnnDataType_t {
    /** <i>native declaration : line 116</i> */
    public static final int CUDNN_DATA_FLOAT = 0;
    /** <i>native declaration : line 117</i> */
    public static final int CUDNN_DATA_DOUBLE = 1;
  };
  /** enum values */
  public static interface cudnnTensorFormat_t {
    /**
     * row major (wStride = 1, hStride = w)<br>
     * <i>native declaration : line 125</i>
     */
    public static final int CUDNN_TENSOR_NCHW = 0;
    /**
     * feature maps interleaved ( cStride = 1 )<br>
     * <i>native declaration : line 126</i>
     */
    public static final int CUDNN_TENSOR_NHWC = 1;
  };
  /** enum values */
  public static interface cudnnAddMode_t {
    /**
     * add one image to every feature maps of each input<br>
     * <i>native declaration : line 217</i>
     */
    public static final int CUDNN_ADD_IMAGE = 0;
    /** <i>native declaration : line 218</i> */
    public static final int CUDNN_ADD_SAME_HW = 0;
    /**
     * add a set of feature maps to a batch of inputs : tensorBias has n=1 , same nb feature than Src/dest<br>
     * <i>native declaration : line 220</i>
     */
    public static final int CUDNN_ADD_FEATURE_MAP = 1;
    /** <i>native declaration : line 221</i> */
    public static final int CUDNN_ADD_SAME_CHW = 1;
    /**
     * add a tensor of size 1,c,1,1 to every corresponding point of n,c,h,w input<br>
     * <i>native declaration : line 223</i>
     */
    public static final int CUDNN_ADD_SAME_C = 2;
    /**
     * add 2 tensors with same n,c,h,w<br>
     * <i>native declaration : line 225</i>
     */
    public static final int CUDNN_ADD_FULL_TENSOR = 3;
  };
  /** enum values */
  public static interface cudnnConvolutionMode_t {
    /** <i>native declaration : line 258</i> */
    public static final int CUDNN_CONVOLUTION = 0;
    /** <i>native declaration : line 259</i> */
    public static final int CUDNN_CROSS_CORRELATION = 1;
  };
  /** enum values */
  public static interface cudnnConvolutionFwdPreference_t {
    /** <i>native declaration : line 365</i> */
    public static final int CUDNN_CONVOLUTION_FWD_NO_WORKSPACE = 0;
    /** <i>native declaration : line 366</i> */
    public static final int CUDNN_CONVOLUTION_FWD_PREFER_FASTEST = 1;
    /** <i>native declaration : line 367</i> */
    public static final int CUDNN_CONVOLUTION_FWD_SPECIFY_WORKSPACE_LIMIT = 2;
  };
  /** enum values */
  public static interface cudnnConvolutionFwdAlgo_t {
    /** <i>native declaration : line 372</i> */
    public static final int CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_GEMM = 0;
    /** <i>native declaration : line 373</i> */
    public static final int CUDNN_CONVOLUTION_FWD_ALGO_IMPLICIT_PRECOMP_GEMM = 1;
    /** <i>native declaration : line 374</i> */
    public static final int CUDNN_CONVOLUTION_FWD_ALGO_GEMM = 2;
    /** <i>native declaration : line 375</i> */
    public static final int CUDNN_CONVOLUTION_FWD_ALGO_DIRECT = 3;
  };
  /** enum values */
  public static interface cudnnSoftmaxAlgorithm_t {
    /**
     * straightforward implementation<br>
     * <i>native declaration : line 474</i>
     */
    public static final int CUDNN_SOFTMAX_FAST = 0;
    /**
     * subtract max from every point to avoid overflow<br>
     * <i>native declaration : line 475</i>
     */
    public static final int CUDNN_SOFTMAX_ACCURATE = 1;
  };
  /** enum values */
  public static interface cudnnSoftmaxMode_t {
    /**
     * compute the softmax over all C, H, W for each N<br>
     * <i>native declaration : line 480</i>
     */
    public static final int CUDNN_SOFTMAX_MODE_INSTANCE = 0;
    /**
     * compute the softmax over all C for each H, W, N<br>
     * <i>native declaration : line 481</i>
     */
    public static final int CUDNN_SOFTMAX_MODE_CHANNEL = 1;
  };
  /** enum values */
  public static interface cudnnPoolingMode_t {
    /** <i>native declaration : line 517</i> */
    public static final int CUDNN_POOLING_MAX = 0;
```

```java
    /**
     * count for average includes padded values<br>
     * <i>native declaration : line 518</i>
     */
    public static final int CUDNN_POOLING_AVERAGE_COUNT_INCLUDE_PADDING = 1;
    /**
     * count for average does not include padded values<br>
     * <i>native declaration : line 519</i>
     */
    public static final int CUDNN_POOLING_AVERAGE_COUNT_EXCLUDE_PADDING = 2;
  };
  /** enum values */
  public static interface cudnnActivationMode_t {
    /** <i>native declaration : line 611</i> */
    public static final int CUDNN_ACTIVATION_SIGMOID = 0;
    /** <i>native declaration : line 612</i> */
    public static final int CUDNN_ACTIVATION_RELU = 1;
    /** <i>native declaration : line 613</i> */
    public static final int CUDNN_ACTIVATION_TANH = 2;
  };
  public static final int CUDNN_VERSION = (int)2000;
  /**
   * Original signature : <code>size_t cudnnGetVersion()</code><br>
   * <i>native declaration : line 76</i>
   */
//NativeSize cudnnGetVersion();
  int cudnnGetVersion();
  /**
   * human-readable error messages<br>
   * Original signature : <code>char* cudnnGetErrorString(cudnnStatus_t)</code><br>
   * <i>native declaration : line 97</i>
   */
  String cudnnGetErrorString(int status);
///**
// * Original signature : <code>cudnnStatus_t cudnnCreate(cudnnHandle_t*)</code><br>
// * <i>native declaration : line 99</i><br>
// * @deprecated use the safer method {@link #cudnnCreate(com.sun.jna.ptr.PointerByReference)} instead
// */
//@Deprecated
//int cudnnCreate(Pointer handle);
///**
// * Original signature : <code>cudnnStatus_t cudnnCreate(cudnnHandle_t*)</code><br>
// * <i>native declaration : line 99</i>
// */
  int cudnnCreate(PointerByReference handle);
///**
// * Original signature : <code>cudnnStatus_t cudnnDestroy(cudnnHandle_t)</code><br>
// * <i>native declaration : line 100</i><br>
// * @deprecated use the safer methods {@link #cudnnDestroy(cudnn.CudnnLibrary.cudnnHandle_t)} and {@link
#cudnnDestroy(com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnDestroy(Pointer handle);
///**
// * Original signature : <code>cudnnStatus_t cudnnDestroy(cudnnHandle_t)</code><br>
// * <i>native declaration : line 100</i>
// */
  int cudnnDestroy(CudnnLibrary.cudnnHandle_t handle);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetStream(cudnnHandle_t, cudaStream_t)</code><br>
// * <i>native declaration : line 101</i><br>
// * @deprecated use the safer methods {@link #cudnnSetStream(cudnn.CudnnLibrary.cudnnHandle_t,
cudnn.CudnnLibrary.cudaStream_t)} and {@link #cudnnSetStream(com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudaStream_t)} instead
// */
//@Deprecated
//int cudnnSetStream(Pointer handle, CudnnLibrary.cudaStream_t streamId);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetStream(cudnnHandle_t, cudaStream_t)</code><br>
// * <i>native declaration : line 101</i>
// */
//int cudnnSetStream(CudnnLibrary.cudnnHandle_t handle, CudnnLibrary.cudaStream_t streamId);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetStream(cudnnHandle_t, cudaStream_t*)</code><br>
// * <i>native declaration : line 102</i><br>
// * @deprecated use the safer methods {@link #cudnnGetStream(cudnn.CudnnLibrary.cudnnHandle_t,
cudnn.CudnnLibrary.cudaStream_t)} and {@link #cudnnGetStream(com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudaStream_t)} instead
// */
//@Deprecated
//int cudnnGetStream(Pointer handle, CudnnLibrary.cudaStream_t streamId);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetStream(cudnnHandle_t, cudaStream_t*)</code><br>
// * <i>native declaration : line 102</i>
// */
//int cudnnGetStream(CudnnLibrary.cudnnHandle_t handle, CudnnLibrary.cudaStream_t streamId);
///**
// * Create an instance of a generic Tensor descriptor<br>
// * Original signature : <code>cudnnStatus_t cudnnCreateTensorDescriptor(cudnnTensorDescriptor_t*)</code><br>
// * <i>native declaration : line 121</i><br>
// * @deprecated use the safer method {@link #cudnnCreateTensorDescriptor(com.sun.jna.ptr.PointerByReference)}
instead
// */
//@Deprecated
//int cudnnCreateTensorDescriptor(Pointer tensorDesc);
  /**
   * Create an instance of a generic Tensor descriptor<br>
   * Original signature : <code>cudnnStatus_t cudnnCreateTensorDescriptor(cudnnTensorDescriptor_t*)</code><br>
   * <i>native declaration : line 121</i>
   */
  int cudnnCreateTensorDescriptor(PointerByReference tensorDesc);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetTensor4dDescriptor(cudnnTensorDescriptor_t,
cudnnTensorFormat_t, cudnnDataType_t, int, int, int, int)</code><br>
// * @param dataType image data type<br>
// * @param n number of inputs (batch size)<br>
// * @param c number of input feature maps<br>
// * @param h height of input section<br>
// * @param w width of input section<br>
// * <i>native declaration : line 129</i><br>
// * @deprecated use the safer methods {@link
#cudnnSetTensor4dDescriptor(cudnn.CudnnLibrary.cudnnTensorDescriptor_t, int, int, int, int, int, int)} and {@link
#cudnnSetTensor4dDescriptor(com.sun.jna.Pointer, int, int, int, int, int, int)} instead
// */
//@Deprecated
//int cudnnSetTensor4dDescriptor(Pointer tensorDesc, int format, int dataType, int n, int c, int h, int w);
  /**
   * Original signature : <code>cudnnStatus_t cudnnSetTensor4dDescriptor(cudnnTensorDescriptor_t,
cudnnTensorFormat_t, cudnnDataType_t, int, int, int, int)</code><br>
   * @param dataType image data type<br>
   * @param n number of inputs (batch size)<br>
   * @param c number of input feature maps<br>
   * @param h height of input section<br>
   * @param w width of input section<br>
   * <i>native declaration : line 129</i>
   */
  int cudnnSetTensor4dDescriptor(CudnnLibrary.cudnnTensorDescriptor_t tensorDesc, int format, int dataType, int
n, int c, int h, int w);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetTensor4dDescriptorEx(cudnnTensorDescriptor_t,
cudnnDataType_t, int, int, int, int, int, int, int, int)</code><br>
// * @param dataType image data type<br>
// * @param n number of inputs (batch size)<br>
// * @param c number of input feature maps<br>
// * @param h height of input section<br>
// * @param w width of input section<br>
// * <i>native declaration : line 139</i><br>
// * @deprecated use the safer methods {@link
#cudnnSetTensor4dDescriptorEx(cudnn.CudnnLibrary.cudnnTensorDescriptor_t, int, int, int, int, int, int, int, int,
int)} and {@link #cudnnSetTensor4dDescriptorEx(com.sun.jna.Pointer, int, int, int, int, int, int, int, int, int)}
instead
// */
//@Deprecated
//int cudnnSetTensor4dDescriptorEx(Pointer tensorDesc, int dataType, int n, int c, int h, int w, int nStride, int
cStride, int hStride, int wStride);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetTensor4dDescriptorEx(cudnnTensorDescriptor_t,
cudnnDataType_t, int, int, int, int, int, int, int, int)</code><br>
```

```
// * @param dataType image data type<br>
// * @param n number of inputs (batch size)<br>
// * @param c number of input feature maps<br>
// * @param h height of input section<br>
// * @param w width of input section<br>
// * <i>native declaration : line 139</i>
// */
//int cudnnSetTensor4dDescriptorEx(CudnnLibrary.cudnnTensorDescriptor_t tensorDesc, int dataType, int n, int c,
int h, int w, int nStride, int cStride, int hStride, int wStride);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetTensor4dDescriptor(const cudnnTensorDescriptor_t,
cudnnDataType_t*, int*, int*, int*, int*, int*, int*, int*, int*)</code><br>
// * @param dataType image data type<br>
// * @param n number of inputs (batch size)<br>
// * @param c number of input feature maps<br>
// * @param h height of input section<br>
// * @param w width of input section<br>
// * <i>native declaration : line 151</i><br>
// * @deprecated use the safer methods {@link
#cudnnGetTensor4dDescriptor(cudnn.CudnnLibrary.cudnnTensorDescriptor_t, java.nio.IntByReference,
java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference,
java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference)} and {@link
#cudnnGetTensor4dDescriptor(com.sun.jna.Pointer, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnGetTensor4dDescriptor(Pointer tensorDesc, IntByReference dataType, IntByReference n, IntByReference c,
IntByReference h, IntByReference w, IntByReference nStride, IntByReference cStride, IntByReference hStride,
IntByReference wStride);
  /**
   * Original signature : <code>cudnnStatus_t cudnnGetTensor4dDescriptor(const cudnnTensorDescriptor_t,
cudnnDataType_t*, int*, int*, int*, int*, int*, int*, int*, int*)</code><br>
   * @param dataType image data type<br>
   * @param n number of inputs (batch size)<br>
   * @param c number of input feature maps<br>
   * @param h height of input section<br>
   * @param w width of input section<br>
   * <i>native declaration : line 151</i>
   */
  int cudnnGetTensor4dDescriptor(CudnnLibrary.cudnnTensorDescriptor_t tensorDesc, IntByReference dataType,
IntByReference n, IntByReference c, IntByReference h, IntByReference w, IntByReference nStride, IntByReference
cStride, IntByReference hStride, IntByReference wStride);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetTensorNdDescriptor(cudnnTensorDescriptor_t, cudnnDataType_t,
int, const int[], const int[])</code><br>
// * <i>native declaration : line 163</i><br>
// * @deprecated use the safer methods {@link
#cudnnSetTensorNdDescriptor(cudnn.CudnnLibrary.cudnnTensorDescriptor_t, int, int, int[], int[])} and {@link
#cudnnSetTensorNdDescriptor(com.sun.jna.Pointer, int, int, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnSetTensorNdDescriptor(Pointer tensorDesc, int dataType, int nbDims, IntByReference dimA, IntByReference
strideA);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetTensorNdDescriptor(cudnnTensorDescriptor_t, cudnnDataType_t,
int, const int[], const int[])</code><br>
// * <i>native declaration : line 163</i>
// */
//int cudnnSetTensorNdDescriptor(CudnnLibrary.cudnnTensorDescriptor_t tensorDesc, int dataType, int nbDims, int
dimA[], int strideA[]);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetTensorNdDescriptor(const cudnnTensorDescriptor_t, int,
cudnnDataType_t*, int*, int[], int[])</code><br>
// * <i>native declaration : line 170</i><br>
// * @deprecated use the safer methods {@link
#cudnnGetTensorNdDescriptor(cudnn.CudnnLibrary.cudnnTensorDescriptor_t, int, java.nio.IntByReference,
java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference)} and {@link
#cudnnGetTensorNdDescriptor(com.sun.jna.Pointer, int, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnGetTensorNdDescriptor(Pointer tensorDesc, int nbDimsRequested, IntByReference dataType, IntByReference
nbDims, IntByReference dimA, IntByReference strideA);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetTensorNdDescriptor(const cudnnTensorDescriptor_t, int,
cudnnDataType_t*, int*, int[], int[])</code><br>
// * <i>native declaration : line 170</i>
// */
//int cudnnGetTensorNdDescriptor(CudnnLibrary.cudnnTensorDescriptor_t tensorDesc, int nbDimsRequested,
IntByReference dataType, IntByReference nbDims, IntByReference dimA, IntByReference strideA);
///**
// * Destroy an instance of Tensor4d descriptor<br>
// * Original signature : <code>cudnnStatus_t cudnnDestroyTensorDescriptor(cudnnTensorDescriptor_t)</code><br>
// * <i>native declaration : line 202</i><br>
// * @deprecated use the safer methods {@link
#cudnnDestroyTensorDescriptor(cudnn.CudnnLibrary.cudnnTensorDescriptor_t)} and {@link
#cudnnDestroyTensorDescriptor(com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnDestroyTensorDescriptor(Pointer tensorDesc);
  /**
   * Destroy an instance of Tensor4d descriptor<br>
   * Original signature : <code>cudnnStatus_t cudnnDestroyTensorDescriptor(cudnnTensorDescriptor_t)</code><br>
   * <i>native declaration : line 202</i>
   */
  int cudnnDestroyTensorDescriptor(CudnnLibrary.cudnnTensorDescriptor_t tensorDesc);
///**
// * Tensor layout conversion helper (dest = alpha * src + beta * dest)<br>
// * Original signature : <code>cudnnStatus_t cudnnTransformTensor(cudnnHandle_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
// * <i>native declaration : line 206</i><br>
// * @deprecated use the safer methods {@link #cudnnTransformTensor(cudnn.CudnnLibrary.cudnnHandle_t,
com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer)} and {@link
#cudnnTransformTensor(com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnTransformTensor(Pointer handle, ByReference alpha, Pointer srcDesc, Pointer srcData, ByReference beta,
Pointer destDesc, Pointer destData);
///**
// * Tensor layout conversion helper (dest = alpha * src + beta * dest)<br>
// * Original signature : <code>cudnnStatus_t cudnnTransformTensor(cudnnHandle_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
// * <i>native declaration : line 206</i>
// */
//int cudnnTransformTensor(CudnnLibrary.cudnnHandle_t handle, ByReference alpha,
CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer srcData, ByReference beta,
CudnnLibrary.cudnnTensorDescriptor_t destDesc, Pointer destData);
///**
// * Tensor Bias addition : srcDest = alpha * bias + beta * srcDestDesc<br>
// * Original signature : <code>cudnnStatus_t cudnnAddTensor(cudnnHandle_t, cudnnAddMode_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, cudnnTensorDescriptor_t, void*)</code><br>
// * <i>native declaration : line 229</i><br>
// * @deprecated use the safer methods {@link #cudnnAddTensor(cudnn.CudnnLibrary.cudnnHandle_t, int,
com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer)} and {@link #cudnnAddTensor(com.sun.jna.Pointer,
int, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnAddTensor(Pointer handle, int mode, ByReference alpha, Pointer biasDesc, Pointer biasData, ByReference
beta, Pointer srcDestDesc, Pointer srcDestData);
  /**
   * Tensor Bias addition : srcDest = alpha * bias + beta * srcDestDesc<br>
   * Original signature : <code>cudnnStatus_t cudnnAddTensor(cudnnHandle_t, cudnnAddMode_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, cudnnTensorDescriptor_t, void*)</code><br>
   * <i>native declaration : line 229</i>
   */
  int cudnnAddTensor(CudnnLibrary.cudnnHandle_t handle, int mode, ByReference alpha,
CudnnLibrary.cudnnTensorDescriptor_t biasDesc, Pointer biasData, ByReference beta,
CudnnLibrary.cudnnTensorDescriptor_t srcDestDesc, Pointer srcDestData);
///**
// * Set all data points of a tensor to a given value : srcDest = value<br>
```

```
// * Original signature : <code>cudnnStatus_t cudnnSetTensor(cudnnHandle_t, const cudnnTensorDescriptor_t, void*,
const void*)</code><br>
// * <i>native declaration : line 240</i><br>
// * @deprecated use the safer methods {@link #cudnnSetTensor(cudnn.CudnnLibrary.cudnnHandle_t,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer, com.sun.jna.Pointer)} and {@link
#cudnnSetTensor(com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnSetTensor(Pointer handle, Pointer srcDestDesc, Pointer srcDestData, Pointer value);
///**
// * Set all data points of a tensor to a given value : srcDest = value<br>
// * Original signature : <code>cudnnStatus_t cudnnSetTensor(cudnnHandle_t, const cudnnTensorDescriptor_t, void*,
const void*)</code><br>
// * <i>native declaration : line 240</i>
// */
//int cudnnSetTensor(CudnnLibrary.cudnnHandle_t handle, CudnnLibrary.cudnnTensorDescriptor_t srcDestDesc, Pointer
srcDestData, Pointer value);
///**
// * Set all data points of a tensor to a given value : srcDest = alpha * srcDest<br>
// * Original signature : <code>cudnnStatus_t cudnnScaleTensor(cudnnHandle_t, const cudnnTensorDescriptor_t, void*,
const void*)</code><br>
// * <i>native declaration : line 247</i><br>
// * @deprecated use the safer methods {@link #cudnnScaleTensor(cudnn.CudnnLibrary.cudnnHandle_t,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer, com.sun.jna.Pointer)} and {@link
#cudnnScaleTensor(com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnScaleTensor(Pointer handle, Pointer srcDestDesc, Pointer srcDestData, ByReference alpha);
///**
// * Set all data points of a tensor to a given value : srcDest = alpha * srcDest<br>
// * Original signature : <code>cudnnStatus_t cudnnScaleTensor(cudnnHandle_t, const cudnnTensorDescriptor_t, void*,
const void*)</code><br>
// * <i>native declaration : line 247</i>
// */
//int cudnnScaleTensor(CudnnLibrary.cudnnHandle_t handle, CudnnLibrary.cudnnTensorDescriptor_t srcDestDesc,
Pointer srcDestData, ByReference alpha);
///**
// * Create an instance of FilterStruct<br>
// * Original signature : <code>cudnnStatus_t cudnnCreateFilterDescriptor(cudnnFilterDescriptor_t*)</code><br>
// * <i>native declaration : line 264</i><br>
// * @deprecated use the safer method {@link #cudnnCreateFilterDescriptor(com.sun.jna.ptr.PointerByReference)}
instead
// */
//@Deprecated
//int cudnnCreateFilterDescriptor(Pointer filterDesc);
  /**
   * Create an instance of FilterStruct<br>
   * Original signature : <code>cudnnStatus_t cudnnCreateFilterDescriptor(cudnnFilterDescriptor_t*)</code><br>
   * <i>native declaration : line 264</i>
   */
  int cudnnCreateFilterDescriptor(PointerByReference filterDesc);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetFilter4dDescriptor(cudnnFilterDescriptor_t, cudnnDataType_t,
int, int, int, int)</code><br>
// * @param dataType image data type<br>
// * @param k number of output feature maps<br>
// * @param c number of input feature maps<br>
// * @param h height of each input filter<br>
// * @param w width of  each input fitler<br>
// * <i>native declaration : line 266</i><br>
// * @deprecated use the safer methods {@link
#cudnnSetFilter4dDescriptor(cudnn.CudnnLibrary.cudnnFilterDescriptor_t, int, int, int, int, int)} and {@link
#cudnnSetFilter4dDescriptor(com.sun.jna.Pointer, int, int, int, int, int)} instead
// */
//@Deprecated
//int cudnnSetFilter4dDescriptor(Pointer filterDesc, int dataType, int k, int c, int h, int w);
  /**
   * Original signature : <code>cudnnStatus_t cudnnSetFilter4dDescriptor(cudnnFilterDescriptor_t, cudnnDataType_t,
int, int, int, int)</code><br>
   * @param dataType image data type<br>
   * @param k number of output feature maps<br>
   * @param c number of input feature maps<br>
   * @param h height of each input filter<br>
   * @param w width of  each input fitler<br>
   * <i>native declaration : line 266</i>
   */
  int cudnnSetFilter4dDescriptor(CudnnLibrary.cudnnFilterDescriptor_t filterDesc, int dataType, int k, int c, int
h, int w);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetFilter4dDescriptor(const cudnnFilterDescriptor_t,
cudnnDataType_t*, int*, int*, int*, int*)</code><br>
// * @param dataType image data type<br>
// * @param k number of output feature maps<br>
// * @param c number of input feature maps<br>
// * @param h height of each input filter<br>
// * @param w width of  each input fitler<br>
// * <i>native declaration : line 274</i><br>
// * @deprecated use the safer methods {@link
#cudnnGetFilter4dDescriptor(cudnn.CudnnLibrary.cudnnFilterDescriptor_t, java.nio.IntByReference,
java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference)} and {@link
#cudnnGetFilter4dDescriptor(com.sun.jna.Pointer, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnGetFilter4dDescriptor(Pointer filterDesc, IntByReference dataType, IntByReference k, IntByReference c,
IntByReference h, IntByReference w);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetFilter4dDescriptor(const cudnnFilterDescriptor_t,
cudnnDataType_t*, int*, int*, int*, int*)</code><br>
// * @param dataType image data type<br>
// * @param k number of output feature maps<br>
// * @param c number of input feature maps<br>
// * @param h height of each input filter<br>
// * @param w width of  each input fitler<br>
// * <i>native declaration : line 274</i>
// */
//int cudnnGetFilter4dDescriptor(CudnnLibrary.cudnnFilterDescriptor_t filterDesc, IntByReference dataType,
IntByReference k, IntByReference c, IntByReference h, IntByReference w);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetFilterNdDescriptor(cudnnFilterDescriptor_t, cudnnDataType_t,
int, const int[])</code><br>
// * @param dataType image data type<br>
// * <i>native declaration : line 282</i><br>
// * @deprecated use the safer methods {@link
#cudnnSetFilterNdDescriptor(cudnn.CudnnLibrary.cudnnFilterDescriptor_t, int, int, int[])} and {@link
#cudnnSetFilterNdDescriptor(com.sun.jna.Pointer, int, int, com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnSetFilterNdDescriptor(Pointer filterDesc, int dataType, int nbDims, IntByReference filterDimA);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetFilterNdDescriptor(cudnnFilterDescriptor_t, cudnnDataType_t,
int, const int[])</code><br>
// * @param dataType image data type<br>
// * <i>native declaration : line 282</i>
// */
//int cudnnSetFilterNdDescriptor(CudnnLibrary.cudnnFilterDescriptor_t filterDesc, int dataType, int nbDims, int
filterDimA[]);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetFilterNdDescriptor(const cudnnFilterDescriptor_t, int,
cudnnDataType_t*, int*, int[])</code><br>
// * @param dataType image data type<br>
// * <i>native declaration : line 288</i><br>
// * @deprecated use the safer methods {@link
#cudnnGetFilterNdDescriptor(cudnn.CudnnLibrary.cudnnFilterDescriptor_t, int, java.nio.IntByReference,
java.nio.IntByReference, java.nio.IntByReference)} and {@link #cudnnGetFilterNdDescriptor(com.sun.jna.Pointer,
int, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnGetFilterNdDescriptor(Pointer filterDesc, int nbDimsRequested, IntByReference dataType, IntByReference
nbDims, IntByReference filterDimA);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetFilterNdDescriptor(const cudnnFilterDescriptor_t, int,
cudnnDataType_t*, int*, int[])</code><br>
// * @param dataType image data type<br>
// * <i>native declaration : line 288</i>
// */
//int cudnnGetFilterNdDescriptor(CudnnLibrary.cudnnFilterDescriptor_t filterDesc, int nbDimsRequested,
IntByReference dataType, IntByReference nbDims, IntByReference filterDimA);
```

```
///**
// * Original signature : <code>cudnnStatus_t cudnnDestroyFilterDescriptor(cudnnFilterDescriptor_t)</code><br>
// * <i>native declaration : line 295</i><br>
// * @deprecated use the safer methods {@link
#cudnnDestroyFilterDescriptor(cudnn.CudnnLibrary.cudnnFilterDescriptor_t)} and {@link
#cudnnDestroyFilterDescriptor(com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnDestroyFilterDescriptor(Pointer filterDesc);
  /**
   * Original signature : <code>cudnnStatus_t cudnnDestroyFilterDescriptor(cudnnFilterDescriptor_t)</code><br>
   * <i>native declaration : line 295</i>
   */
  int cudnnDestroyFilterDescriptor(CudnnLibrary.cudnnFilterDescriptor_t filterDesc);
///**
// * Create an instance of convolution descriptor<br>
// * Original signature : <code>cudnnStatus_t
cudnnCreateConvolutionDescriptor(cudnnConvolutionDescriptor_t*)</code><br>
// * <i>native declaration : line 298</i><br>
// * @deprecated use the safer method {@link
#cudnnCreateConvolutionDescriptor(com.sun.jna.ptr.PointerByReference)} instead
// */
//@Deprecated
//int cudnnCreateConvolutionDescriptor(Pointer convDesc);
  /**
   * Create an instance of convolution descriptor<br>
   * Original signature : <code>cudnnStatus_t
cudnnCreateConvolutionDescriptor(cudnnConvolutionDescriptor_t*)</code><br>
   * <i>native declaration : line 298</i>
   */
  int cudnnCreateConvolutionDescriptor(PointerByReference convDesc);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetConvolution2dDescriptor(cudnnConvolutionDescriptor_t, int,
int, int, int, int, int, cudnnConvolutionMode_t)</code><br>
// * @param pad_h zero-padding height<br>
// * @param pad_w zero-padding width<br>
// * @param u vertical filter stride<br>
// * @param v horizontal filter stride<br>
// * @param upscalex upscale the input in x-direction<br>
// * @param upscaley upscale the input in y-direction<br>
// * <i>native declaration : line 300</i><br>
// * @deprecated use the safer methods {@link
#cudnnSetConvolution2dDescriptor(cudnn.CudnnLibrary.cudnnConvolutionDescriptor_t, int, int, int, int, int, int,
int)} and {@link #cudnnSetConvolution2dDescriptor(com.sun.jna.Pointer, int, int, int, int, int, int, int)} instead
// */
//@Deprecated
//int cudnnSetConvolution2dDescriptor(Pointer convDesc, int pad_h, int pad_w, int u, int v, int upscalex, int upscaley,
int mode);
  /**
   * Original signature : <code>cudnnStatus_t cudnnSetConvolution2dDescriptor(cudnnConvolutionDescriptor_t, int,
int, int, int, int, int, cudnnConvolutionMode_t)</code><br>
   * @param pad_h zero-padding height<br>
   * @param pad_w zero-padding width<br>
   * @param u vertical filter stride<br>
   * @param v horizontal filter stride<br>
   * @param upscalex upscale the input in x-direction<br>
   * @param upscaley upscale the input in y-direction<br>
   * <i>native declaration : line 300</i>
   */
  int cudnnSetConvolution2dDescriptor(CudnnLibrary.cudnnConvolutionDescriptor_t convDesc, int pad_h, int pad_w,
int u, int v, int upscalex, int upscaley, int mode);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetConvolution2dDescriptor(const cudnnConvolutionDescriptor_t,
int*, int*, int*, int*, int*, int*, cudnnConvolutionMode_t*)</code><br>
// * @param pad_h zero-padding height<br>
// * @param pad_w zero-padding width<br>
// * @param u vertical filter stride<br>
// * @param v horizontal filter stride<br>
// * @param upscalex upscale the input in x-direction<br>
// * @param upscaley upscale the input in y-direction<br>
// * <i>native declaration : line 311</i><br>
// * @deprecated use the safer methods {@link
#cudnnGetConvolution2dDescriptor(cudnn.CudnnLibrary.cudnnConvolutionDescriptor_t, java.nio.IntByReference,
java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference,
java.nio.IntByReference, java.nio.IntByReference)} and {@link
#cudnnGetConvolution2dDescriptor(com.sun.jna.Pointer, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnGetConvolution2dDescriptor(Pointer convDesc, IntByReference pad_h, IntByReference pad_w, IntByReference
u, IntByReference v, IntByReference upscalex, IntByReference upscaley, IntByReference mode);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetConvolution2dDescriptor(const cudnnConvolutionDescriptor_t,
int*, int*, int*, int*, int*, int*, cudnnConvolutionMode_t*)</code><br>
// * @param pad_h zero-padding height<br>
// * @param pad_w zero-padding width<br>
// * @param u vertical filter stride<br>
// * @param v horizontal filter stride<br>
// * @param upscalex upscale the input in x-direction<br>
// * @param upscaley upscale the input in y-direction<br>
// * <i>native declaration : line 311</i>
// */
//int cudnnGetConvolution2dDescriptor(CudnnLibrary.cudnnConvolutionDescriptor_t convDesc, IntByReference pad_h,
IntByReference pad_w, IntByReference u, IntByReference v, IntByReference upscalex, IntByReference upscaley,
IntByReference mode);
///**
// * Helper function to return the dimensions of the output tensor given a convolution descriptor<br>
// * Original signature : <code>cudnnStatus_t cudnnGetConvolution2dForwardOutputDim(const
cudnnConvolutionDescriptor_t, const cudnnTensorDescriptor_t, const cudnnFilterDescriptor_t, int*, int*, int*,
int*)</code><br>
// * <i>native declaration : line 322</i><br>
// * @deprecated use the safer methods {@link
#cudnnGetConvolution2dForwardOutputDim(cudnn.CudnnLibrary.cudnnConvolutionDescriptor_t,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, cudnn.CudnnLibrary.cudnnFilterDescriptor_t, java.nio.IntByReference,
java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference)} and {@link
#cudnnGetConvolution2dForwardOutputDim(com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnGetConvolution2dForwardOutputDim(Pointer convDesc, Pointer inputTensorDesc, Pointer filterDesc,
IntByReference n, IntByReference c, IntByReference h, IntByReference w);
  /**
   * Helper function to return the dimensions of the output tensor given a convolution descriptor<br>
   * Original signature : <code>cudnnStatus_t cudnnGetConvolution2dForwardOutputDim(const
cudnnConvolutionDescriptor_t, const cudnnTensorDescriptor_t, const cudnnFilterDescriptor_t, int*, int*, int*,
int*)</code><br>
   * <i>native declaration : line 322</i>
   */
  int cudnnGetConvolution2dForwardOutputDim(CudnnLibrary.cudnnConvolutionDescriptor_t convDesc,
CudnnLibrary.cudnnTensorDescriptor_t inputTensorDesc, CudnnLibrary.cudnnFilterDescriptor_t filterDesc,
IntByReference n, IntByReference c, IntByReference h, IntByReference w);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetConvolutionNdDescriptor(cudnnConvolutionDescriptor_t, int,
const int[], const int[], const int[], cudnnConvolutionMode_t)</code><br>
// * @param arrayLength nbDims-2 size<br>
// * <i>native declaration : line 332</i><br>
// * @deprecated use the safer methods {@link
#cudnnSetConvolutionNdDescriptor(cudnn.CudnnLibrary.cudnnConvolutionDescriptor_t, int, int[], int[], int[], int)}
and {@link #cudnnSetConvolutionNdDescriptor(com.sun.jna.Pointer, int, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference, int)} instead
// */
//@Deprecated
//int cudnnSetConvolutionNdDescriptor(Pointer convDesc, int arrayLength, IntByReference padA, IntByReference
filterStrideA, IntByReference upscaleA, int mode);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetConvolutionNdDescriptor(cudnnConvolutionDescriptor_t, int,
const int[], const int[], const int[], cudnnConvolutionMode_t)</code><br>
// * @param arrayLength nbDims-2 size<br>
// * <i>native declaration : line 332</i>
// */
//int cudnnSetConvolutionNdDescriptor(CudnnLibrary.cudnnConvolutionDescriptor_t convDesc, int arrayLength, int
padA[], int filterStrideA[], int upscaleA[], int mode);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetConvolutionNdDescriptor(const cudnnConvolutionDescriptor_t,
int, int*, int[], int[], int[], cudnnConvolutionMode_t*)</code><br>
// * <i>native declaration : line 340</i><br>
```

```
// * @deprecated use the safer methods {@link
#cudnnGetConvolutionNdDescriptor(cudnn.CudnnLibrary.cudnnConvolutionDescriptor_t, int, java.nio.IntByReference,
java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference)} and {@link
#cudnnGetConvolutionNdDescriptor(com.sun.jna.Pointer, int, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnGetConvolutionNdDescriptor(Pointer convDesc, int arrayLengthRequested, IntByReference arrayLength,
IntByReference padA, IntByReference strideA, IntByReference upscaleA, IntByReference mode);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetConvolutionNdDescriptor(const cudnnConvolutionDescriptor_t,
int, int*, int[], int[], int[], cudnnConvolutionMode_t*)</code><br>
// * <i>native declaration : line 340</i>
// */
//int cudnnGetConvolutionNdDescriptor(CudnnLibrary.cudnnConvolutionDescriptor_t convDesc, int
arrayLengthRequested, IntByReference arrayLength, IntByReference padA, IntByReference strideA, IntByReference
upscaleA, IntByReference mode);
///**
// * Helper function to return the dimensions of the output tensor given a convolution descriptor<br>
// * Original signature : <code>cudnnStatus_t cudnnGetConvolutionNdForwardOutputDim(const
cudnnConvolutionDescriptor_t, const cudnnTensorDescriptor_t, const cudnnFilterDescriptor_t, int, int[])</code><br>
// * <i>native declaration : line 351</i><br>
// * @deprecated use the safer methods {@link
#cudnnGetConvolutionNdForwardOutputDim(cudnn.CudnnLibrary.cudnnConvolutionDescriptor_t,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, cudnn.CudnnLibrary.cudnnFilterDescriptor_t, int,
java.nio.IntByReference)} and {@link #cudnnGetConvolutionNdForwardOutputDim(com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, int, com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnGetConvolutionNdForwardOutputDim(Pointer convDesc, Pointer inputTensorDesc, Pointer filterDesc, int
nbDims, IntByReference tensorOuputDimA);
///**
// * Helper function to return the dimensions of the output tensor given a convolution descriptor<br>
// * Original signature : <code>cudnnStatus_t cudnnGetConvolutionNdForwardOutputDim(const
cudnnConvolutionDescriptor_t, const cudnnTensorDescriptor_t, const cudnnFilterDescriptor_t, int, int[])</code><br>
// * <i>native declaration : line 351</i>
// */
//int cudnnGetConvolutionNdForwardOutputDim(CudnnLibrary.cudnnConvolutionDescriptor_t convDesc,
CudnnLibrary.cudnnTensorDescriptor_t inputTensorDesc, CudnnLibrary.cudnnFilterDescriptor_t filterDesc, int nbDims,
IntByReference tensorOuputDimA);
///**
// * Destroy an instance of convolution descriptor<br>
// * Original signature : <code>cudnnStatus_t
cudnnDestroyConvolutionDescriptor(cudnnConvolutionDescriptor_t)</code><br>
// * <i>native declaration : line 359</i><br>
// * @deprecated use the safer methods {@link
#cudnnDestroyConvolutionDescriptor(cudnn.CudnnLibrary.cudnnConvolutionDescriptor_t)} and {@link
#cudnnDestroyConvolutionDescriptor(com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnDestroyConvolutionDescriptor(Pointer convDesc);
  /**
   * Destroy an instance of convolution descriptor<br>
   * Original signature : <code>cudnnStatus_t
cudnnDestroyConvolutionDescriptor(cudnnConvolutionDescriptor_t)</code><br>
   * <i>native declaration : line 359</i>
   */
  int cudnnDestroyConvolutionDescriptor(CudnnLibrary.cudnnConvolutionDescriptor_t convDesc);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetConvolutionForwardAlgorithm(cudnnHandle_t, const
cudnnTensorDescriptor_t, const cudnnFilterDescriptor_t, const cudnnConvolutionDescriptor_t, const
cudnnTensorDescriptor_t, cudnnConvolutionFwdPreference_t, size_t, cudnnConvolutionFwdAlgo_t*)</code><br>
// * <i>native declaration : line 379</i><br>
// * @deprecated use the safer methods {@link
#cudnnGetConvolutionForwardAlgorithm(cudnn.CudnnLibrary.cudnnHandle_t,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, cudnn.CudnnLibrary.cudnnFilterDescriptor_t,
cudnn.CudnnLibrary.cudnnConvolutionDescriptor_t, cudnn.CudnnLibrary.cudnnTensorDescriptor_t, int,
com.ochafik.lang.jnaerator.runtime.NativeSize, java.nio.IntByReference)} and {@link
#cudnnGetConvolutionForwardAlgorithm(com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, int, com.ochafik.lang.jnaerator.runtime.NativeSize,
com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnGetConvolutionForwardAlgorithm(Pointer handle, Pointer srcDesc, Pointer filterDesc, Pointer convDesc,
Pointer destDesc, int preference, NativeSize memoryLimitInbytes, IntByReference algo);
  /**
   * Original signature : <code>cudnnStatus_t cudnnGetConvolutionForwardAlgorithm(cudnnHandle_t, const
cudnnTensorDescriptor_t, const cudnnFilterDescriptor_t, const cudnnConvolutionDescriptor_t, const
cudnnTensorDescriptor_t, cudnnConvolutionFwdPreference_t, size_t, cudnnConvolutionFwdAlgo_t*)</code><br>
   * <i>native declaration : line 379</i>
   */
  int cudnnGetConvolutionForwardAlgorithm(CudnnLibrary.cudnnHandle_t handle,
CudnnLibrary.cudnnTensorDescriptor_t srcDesc, CudnnLibrary.cudnnFilterDescriptor_t filterDesc,
CudnnLibrary.cudnnConvolutionDescriptor_t convDesc, CudnnLibrary.cudnnTensorDescriptor_t destDesc, int preference,
int memoryLimitInbytes, IntByReference algo);
///**
// * Helper function to return the minimum size of the workspace to be passed to the convolution given an algo<br>
// * Original signature : <code>cudnnStatus_t cudnnGetConvolutionForwardWorkspaceSize(cudnnHandle_t, const
cudnnTensorDescriptor_t, const cudnnFilterDescriptor_t, const cudnnConvolutionDescriptor_t, const
cudnnTensorDescriptor_t, cudnnConvolutionFwdAlgo_t, size_t*)</code><br>
// * <i>native declaration : line 394</i><br>
// * @deprecated use the safer methods {@link
#cudnnGetConvolutionForwardWorkspaceSize(cudnn.CudnnLibrary.cudnnHandle_t,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, cudnn.CudnnLibrary.cudnnFilterDescriptor_t,
cudnn.CudnnLibrary.cudnnConvolutionDescriptor_t, cudnn.CudnnLibrary.cudnnTensorDescriptor_t, int,
com.ochafik.lang.jnaerator.runtime.NativeSizeByReference)} and {@link
#cudnnGetConvolutionForwardWorkspaceSize(com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, int, com.ochafik.lang.jnaerator.runtime.NativeSizeByReference)} instead
// */
//@Deprecated
//int cudnnGetConvolutionForwardWorkspaceSize(Pointer handle, Pointer srcDesc, Pointer filterDesc, Pointer convDesc,
Pointer destDesc, int algo, NativeSizeByReference sizeInBytes);
  /**
   * Helper function to return the minimum size of the workspace to be passed to the convolution given an algo<br>
   * Original signature : <code>cudnnStatus_t cudnnGetConvolutionForwardWorkspaceSize(cudnnHandle_t, const
cudnnTensorDescriptor_t, const cudnnFilterDescriptor_t, const cudnnConvolutionDescriptor_t, const
cudnnTensorDescriptor_t, cudnnConvolutionFwdAlgo_t, size_t*)</code><br>
   * <i>native declaration : line 394</i>
   */
  int cudnnGetConvolutionForwardWorkspaceSize(CudnnLibrary.cudnnHandle_t handle,
CudnnLibrary.cudnnTensorDescriptor_t srcDesc, CudnnLibrary.cudnnFilterDescriptor_t filterDesc,
CudnnLibrary.cudnnConvolutionDescriptor_t convDesc, CudnnLibrary.cudnnTensorDescriptor_t destDesc, int algo,
IntByReference sizeInBytes);
///**
// * Function to perform the forward multiconvolution<br>
// * Original signature : <code>cudnnStatus_t cudnnConvolutionForward(cudnnHandle_t, const void*, const
cudnnTensorDescriptor_t, const void*, const cudnnFilterDescriptor_t, const void*, const
cudnnConvolutionDescriptor_t, cudnnConvolutionFwdAlgo_t, void*, size_t, const void*, const cudnnTensorDescriptor_t,
void*)</code><br>
// * <i>native declaration : line 407</i><br>
// * @deprecated use the safer methods {@link #cudnnConvolutionForward(cudnn.CudnnLibrary.cudnnHandle_t,
com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnFilterDescriptor_t, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnConvolutionDescriptor_t, int, com.sun.jna.Pointer,
com.ochafik.lang.jnaerator.runtime.NativeSize, com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t,
com.sun.jna.Pointer)} and {@link #cudnnConvolutionForward(com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, int,
com.sun.jna.Pointer, com.ochafik.lang.jnaerator.runtime.NativeSize, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnConvolutionForward(Pointer handle, ByReference alpha, Pointer srcDesc, Pointer srcData, Pointer
filterDesc, Pointer filterData, Pointer convDesc, int algo, Pointer workSpace, NativeSize workSpaceSizeInBytes,
ByReference beta, Pointer destDesc, Pointer destData);
  /**
   * Function to perform the forward multiconvolution<br>
   * Original signature : <code>cudnnStatus_t cudnnConvolutionForward(cudnnHandle_t, const void*, const
cudnnTensorDescriptor_t, const void*, const cudnnFilterDescriptor_t, const void*, const
cudnnConvolutionDescriptor_t, cudnnConvolutionFwdAlgo_t, void*, size_t, const void*, const cudnnTensorDescriptor_t,
void*)</code><br>
   * <i>native declaration : line 407</i>
   */
  int cudnnConvolutionForward(CudnnLibrary.cudnnHandle_t handle, ByReference alpha,
CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer srcData, CudnnLibrary.cudnnFilterDescriptor_t filterDesc,
Pointer filterData, CudnnLibrary.cudnnConvolutionDescriptor_t convDesc, int algo, Pointer workSpace, int
workSpaceSizeInBytes, ByReference beta, CudnnLibrary.cudnnTensorDescriptor_t destDesc, Pointer destData);
///**
```

```
// * Functions to perform the backward multiconvolution<br>
// * Original signature : <code>cudnnStatus_t cudnnConvolutionBackwardBias(cudnnHandle_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
// * <i>native declaration : line 423</i><br>
// * @deprecated use the safer methods {@link #cudnnConvolutionBackwardBias(cudnn.CudnnLibrary.cudnnHandle_t,
com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer)} and {@link
#cudnnConvolutionBackwardBias(com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnConvolutionBackwardBias(Pointer handle, ByReference alpha, Pointer srcDesc, Pointer srcData, ByReference
beta, Pointer destDesc, Pointer destData);
  /**
   * Functions to perform the backward multiconvolution<br>
   * Original signature : <code>cudnnStatus_t cudnnConvolutionBackwardBias(cudnnHandle_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
   * <i>native declaration : line 423</i>
   */
  int cudnnConvolutionBackwardBias(CudnnLibrary.cudnnHandle_t handle, ByReference alpha,
CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer srcData, ByReference beta,
CudnnLibrary.cudnnTensorDescriptor_t destDesc, Pointer destData);
///**
// * Original signature : <code>cudnnStatus_t cudnnConvolutionBackwardFilter(cudnnHandle_t, const void*, const
cudnnTensorDescriptor_t, const void*, const cudnnTensorDescriptor_t, const void*, const
cudnnConvolutionDescriptor_t, const void*, const cudnnFilterDescriptor_t, void*)</code><br>
// * <i>native declaration : line 434</i><br>
// * @deprecated use the safer methods {@link #cudnnConvolutionBackwardFilter(cudnn.CudnnLibrary.cudnnHandle_t,
com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnConvolutionDescriptor_t, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnFilterDescriptor_t, com.sun.jna.Pointer)} and {@link
#cudnnConvolutionBackwardFilter(com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnConvolutionBackwardFilter(Pointer handle, ByReference alpha, Pointer srcDesc, Pointer srcData, Pointer
diffDesc, Pointer diffData, Pointer convDesc, ByReference beta, Pointer gradDesc, Pointer gradData);
  /**
   * Original signature : <code>cudnnStatus_t cudnnConvolutionBackwardFilter(cudnnHandle_t, const void*, const
cudnnTensorDescriptor_t, const void*, const cudnnTensorDescriptor_t, const void*, const
cudnnConvolutionDescriptor_t, const void*, const cudnnFilterDescriptor_t, void*)</code><br>
   * <i>native declaration : line 434</i>
   */
  int cudnnConvolutionBackwardFilter(CudnnLibrary.cudnnHandle_t handle, ByReference alpha,
CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer srcData, CudnnLibrary.cudnnTensorDescriptor_t diffDesc,
Pointer diffData, CudnnLibrary.cudnnConvolutionDescriptor_t convDesc, ByReference beta,
CudnnLibrary.cudnnFilterDescriptor_t gradDesc, Pointer gradData);
///**
// * Original signature : <code>cudnnStatus_t cudnnConvolutionBackwardData(cudnnHandle_t, const void*, const
cudnnFilterDescriptor_t, const void*, const cudnnTensorDescriptor_t, const void*, const
cudnnConvolutionDescriptor_t, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
// * <i>native declaration : line 447</i><br>
// * @deprecated use the safer methods {@link #cudnnConvolutionBackwardData(cudnn.CudnnLibrary.cudnnHandle_t,
com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnFilterDescriptor_t, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnConvolutionDescriptor_t, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer)} and {@link
#cudnnConvolutionBackwardData(com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnConvolutionBackwardData(Pointer handle, ByReference alpha, Pointer filterDesc, Pointer filterData,
Pointer diffDesc, Pointer diffData, Pointer convDesc, ByReference beta, Pointer gradDesc, Pointer gradData);
  /**
   * Original signature : <code>cudnnStatus_t cudnnConvolutionBackwardData(cudnnHandle_t, const void*, const
cudnnFilterDescriptor_t, const void*, const cudnnTensorDescriptor_t, const void*, const
cudnnConvolutionDescriptor_t, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
   * <i>native declaration : line 447</i>
   */
  int cudnnConvolutionBackwardData(CudnnLibrary.cudnnHandle_t handle, ByReference alpha,
CudnnLibrary.cudnnFilterDescriptor_t filterDesc, Pointer filterData, CudnnLibrary.cudnnTensorDescriptor_t
diffDesc, Pointer diffData, CudnnLibrary.cudnnConvolutionDescriptor_t convDesc, ByReference beta,
CudnnLibrary.cudnnTensorDescriptor_t gradDesc, Pointer gradData);
///**
// * Original signature : <code>cudnnStatus_t cudnnIm2Col(cudnnHandle_t, const void*, const cudnnTensorDescriptor_t,
const void*, const cudnnFilterDescriptor_t, const cudnnConvolutionDescriptor_t, void*)</code><br>
// * <i>native declaration : line 459</i><br>
// * @deprecated use the safer methods {@link #cudnnIm2Col(cudnn.CudnnLibrary.cudnnHandle_t, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnFilterDescriptor_t,
cudnn.CudnnLibrary.cudnnConvolutionDescriptor_t, com.sun.jna.Pointer)} and {@link
#cudnnIm2Col(com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnIm2Col(Pointer handle, ByReference alpha, Pointer srcDesc, Pointer srcData, Pointer filterDesc, Pointer
convDesc, Pointer colBuffer);
///**
// * Original signature : <code>cudnnStatus_t cudnnIm2Col(cudnnHandle_t, const void*, const cudnnTensorDescriptor_t,
const void*, const cudnnFilterDescriptor_t, const cudnnConvolutionDescriptor_t, void*)</code><br>
// * <i>native declaration : line 459</i>
// */
//int cudnnIm2Col(CudnnLibrary.cudnnHandle_t handle, ByReference alpha, CudnnLibrary.cudnnTensorDescriptor_t
srcDesc, Pointer srcData, CudnnLibrary.cudnnFilterDescriptor_t filterDesc,
CudnnLibrary.cudnnConvolutionDescriptor_t convDesc, Pointer colBuffer);
///**
// * Function to perform forward softmax<br>
// * Original signature : <code>cudnnStatus_t cudnnSoftmaxForward(cudnnHandle_t, cudnnSoftmaxAlgorithm_t,
cudnnSoftmaxMode_t, const void*, const cudnnTensorDescriptor_t, const void*, const void*, const
cudnnTensorDescriptor_t, void*)</code><br>
// * <i>native declaration : line 487</i><br>
// * @deprecated use the safer methods {@link #cudnnSoftmaxForward(cudnn.CudnnLibrary.cudnnHandle_t, int, int,
com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer)} and {@link
#cudnnSoftmaxForward(com.sun.jna.Pointer, int, int, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnSoftmaxForward(Pointer handle, int algorithm, int mode, ByReference alpha, Pointer srcDesc, Pointer
srcData, ByReference beta, Pointer destDesc, Pointer destData);
  /**
   * Function to perform forward softmax<br>
   * Original signature : <code>cudnnStatus_t cudnnSoftmaxForward(cudnnHandle_t, cudnnSoftmaxAlgorithm_t,
cudnnSoftmaxMode_t, const void*, const cudnnTensorDescriptor_t, const void*, const void*, const
cudnnTensorDescriptor_t, void*)</code><br>
   * <i>native declaration : line 487</i>
   */
  int cudnnSoftmaxForward(CudnnLibrary.cudnnHandle_t handle, int algorithm, int mode, ByReference alpha,
CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer srcData, ByReference beta,
CudnnLibrary.cudnnTensorDescriptor_t destDesc, Pointer destData);
///**
// * Function to perform backward softmax<br>
// * Original signature : <code>cudnnStatus_t cudnnSoftmaxBackward(cudnnHandle_t, cudnnSoftmaxAlgorithm_t,
cudnnSoftmaxMode_t, const void*, const cudnnTensorDescriptor_t, const void*, const cudnnTensorDescriptor_t, const
void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
// * <i>native declaration : line 499</i><br>
// * @deprecated use the safer methods {@link #cudnnSoftmaxBackward(cudnn.CudnnLibrary.cudnnHandle_t, int, int,
com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer)} and {@link
#cudnnSoftmaxBackward(com.sun.jna.Pointer, int, int, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnSoftmaxBackward(Pointer handle, int algorithm, int mode, ByReference alpha, Pointer srcDesc, Pointer
srcData, Pointer srcDiffDesc, Pointer srcDiffData, ByReference beta, Pointer destDiffDesc, Pointer destDiffData);
///**
// * Function to perform backward softmax<br>
// * Original signature : <code>cudnnStatus_t cudnnSoftmaxBackward(cudnnHandle_t, cudnnSoftmaxAlgorithm_t,
cudnnSoftmaxMode_t, const void*, const cudnnTensorDescriptor_t, const void*, const cudnnTensorDescriptor_t, const
void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
// * <i>native declaration : line 499</i>
// */
```

```
//int cudnnSoftmaxBackward(CudnnLibrary.cudnnHandle_t handle, int algorithm, int mode, ByReference alpha,
CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer srcData, CudnnLibrary.cudnnTensorDescriptor_t srcDiffDesc,
Pointer srcDiffData, ByReference beta, CudnnLibrary.cudnnTensorDescriptor_t destDiffDesc, Pointer destDiffData);
///**
// * Create an instance of pooling descriptor<br>
// * Original signature : <code>cudnnStatus_t cudnnCreatePoolingDescriptor(cudnnPoolingDescriptor_t*)</code><br>
// * <i>native declaration : line 523</i><br>
// * @deprecated use the safer method {@link #cudnnCreatePoolingDescriptor(com.sun.jna.ptr.PointerByReference)}
instead
// */
//@Deprecated
//int cudnnCreatePoolingDescriptor(Pointer poolingDesc);
  /**
   * Create an instance of pooling descriptor<br>
   * Original signature : <code>cudnnStatus_t cudnnCreatePoolingDescriptor(cudnnPoolingDescriptor_t*)</code><br>
   * <i>native declaration : line 523</i>
   */
  int cudnnCreatePoolingDescriptor(PointerByReference poolingDesc);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetPooling2dDescriptor(cudnnPoolingDescriptor_t,
cudnnPoolingMode_t, int, int, int, int, int, int)</code><br>
// * <i>native declaration : line 525</i><br>
// * @deprecated use the safer methods {@link
#cudnnSetPooling2dDescriptor(cudnn.CudnnLibrary.cudnnPoolingDescriptor_t, int, int, int, int, int, int, int)} and
{@link #cudnnSetPooling2dDescriptor(com.sun.jna.Pointer, int, int, int, int, int, int, int)} instead
// */
//@Deprecated
//int cudnnSetPooling2dDescriptor(Pointer poolingDesc, int mode, int windowHeight, int windowWidth, int
verticalPadding, int horizontalPadding, int verticalStride, int horizontalStride);
  /**
   * Original signature : <code>cudnnStatus_t cudnnSetPooling2dDescriptor(cudnnPoolingDescriptor_t,
cudnnPoolingMode_t, int, int, int, int, int, int)</code><br>
   * <i>native declaration : line 525</i>
   */
  int cudnnSetPooling2dDescriptor(CudnnLibrary.cudnnPoolingDescriptor_t poolingDesc, int mode, int windowHeight,
int windowWidth, int verticalPadding, int horizontalPadding, int verticalStride, int horizontalStride);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetPooling2dDescriptor(const cudnnPoolingDescriptor_t,
cudnnPoolingMode_t*, int*, int*, int*, int*, int*, int*)</code><br>
// * <i>native declaration : line 535</i><br>
// * @deprecated use the safer methods {@link
#cudnnGetPooling2dDescriptor(cudnn.CudnnLibrary.cudnnPoolingDescriptor_t, java.nio.IntByReference,
java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference,
java.nio.IntByReference, java.nio.IntByReference)} and {@link #cudnnGetPooling2dDescriptor(com.sun.jna.Pointer,
com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnGetPooling2dDescriptor(Pointer poolingDesc, IntByReference mode, IntByReference windowHeight,
IntByReference windowWidth, IntByReference verticalPadding, IntByReference horizontalPadding, IntByReference
verticalStride, IntByReference horizontalStride);
  /**
   * Original signature : <code>cudnnStatus_t cudnnGetPooling2dDescriptor(const cudnnPoolingDescriptor_t,
cudnnPoolingMode_t*, int*, int*, int*, int*, int*, int*)</code><br>
   * <i>native declaration : line 535</i>
   */
  int cudnnGetPooling2dDescriptor(CudnnLibrary.cudnnPoolingDescriptor_t poolingDesc, IntByReference mode,
IntByReference windowHeight, IntByReference windowWidth, IntByReference verticalPadding, IntByReference
horizontalPadding, IntByReference verticalStride, IntByReference horizontalStride);
///**
// * Original signature : <code>cudnnStatus_t cudnnSetPoolingNdDescriptor(cudnnPoolingDescriptor_t, const
cudnnPoolingMode_t, int, const int[], const int[], const int[])</code><br>
// * <i>native declaration : line 545</i><br>
// * @deprecated use the safer methods {@link
#cudnnSetPoolingNdDescriptor(cudnn.CudnnLibrary.cudnnPoolingDescriptor_t, int, int, int[], int[], int[])} and
{@link #cudnnSetPoolingNdDescriptor(com.sun.jna.Pointer, int, int, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnSetPoolingNdDescriptor(Pointer poolingDesc, int mode, int nbDims, IntByReference windowDimA,
IntByReference paddingA, IntByReference strideA);
///**
```



```
// * Original signature : <code>cudnnStatus_t cudnnSetPoolingNdDescriptor(cudnnPoolingDescriptor_t, const
cudnnPoolingMode_t, int, const int[], const int[], const int[])</code><br>
// * <i>native declaration : line 545</i>
// */
//int cudnnSetPoolingNdDescriptor(CudnnLibrary.cudnnPoolingDescriptor_t poolingDesc, int mode, int nbDims, int
windowDimA[], int paddingA[], int strideA[]);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetPoolingNdDescriptor(const cudnnPoolingDescriptor_t, const
int, cudnnPoolingMode_t*, int*, int[], int[], int[])</code><br>
// * <i>native declaration : line 553</i><br>
// * @deprecated use the safer methods {@link
#cudnnGetPoolingNdDescriptor(cudnn.CudnnLibrary.cudnnPoolingDescriptor_t, int, java.nio.IntByReference,
java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference, java.nio.IntByReference)} and {@link
#cudnnGetPoolingNdDescriptor(com.sun.jna.Pointer, int, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnGetPoolingNdDescriptor(Pointer poolingDesc, int nbDimsRequested, IntByReference mode, IntByReference
nbDims, IntByReference windowDimA, IntByReference paddingA, IntByReference strideA);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetPoolingNdDescriptor(const cudnnPoolingDescriptor_t, const
int, cudnnPoolingMode_t*, int*, int[], int[], int[])</code><br>
// * <i>native declaration : line 553</i>
// */
//int cudnnGetPoolingNdDescriptor(CudnnLibrary.cudnnPoolingDescriptor_t poolingDesc, int nbDimsRequested,
IntByReference mode, IntByReference nbDims, IntByReference windowDimA, IntByReference paddingA, IntByReference
strideA);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetPoolingNdForwardOutputDim(const cudnnPoolingDescriptor_t,
const cudnnTensorDescriptor_t, int, int[])</code><br>
// * <i>native declaration : line 562</i><br>
// * @deprecated use the safer methods {@link
#cudnnGetPoolingNdForwardOutputDim(cudnn.CudnnLibrary.cudnnPoolingDescriptor_t,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, int, java.nio.IntByReference)} and {@link
#cudnnGetPoolingNdForwardOutputDim(com.sun.jna.Pointer, com.sun.jna.Pointer, int,
com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnGetPoolingNdForwardOutputDim(Pointer poolingDesc, Pointer inputTensorDesc, int nbDims, IntByReference
outputTensorDimA);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetPoolingNdForwardOutputDim(const cudnnPoolingDescriptor_t,
const cudnnTensorDescriptor_t, int, int[])</code><br>
// * <i>native declaration : line 562</i>
// */
//int cudnnGetPoolingNdForwardOutputDim(CudnnLibrary.cudnnPoolingDescriptor_t poolingDesc,
CudnnLibrary.cudnnTensorDescriptor_t inputTensorDesc, int nbDims, IntByReference outputTensorDimA);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetPooling2dForwardOutputDim(const cudnnPoolingDescriptor_t,
const cudnnTensorDescriptor_t, int*, int*, int*, int*)</code><br>
// * <i>native declaration : line 567</i><br>
// * @deprecated use the safer methods {@link
#cudnnGetPooling2dForwardOutputDim(cudnn.CudnnLibrary.cudnnPoolingDescriptor_t,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, java.nio.IntByReference, java.nio.IntByReference,
java.nio.IntByReference, java.nio.IntByReference)} and {@link
#cudnnGetPooling2dForwardOutputDim(com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.ptr.IntByReference,
com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference, com.sun.jna.ptr.IntByReference)} instead
// */
//@Deprecated
//int cudnnGetPooling2dForwardOutputDim(Pointer poolingDesc, Pointer inputTensorDesc, IntByReference outN,
IntByReference outC, IntByReference outH, IntByReference outW);
///**
// * Original signature : <code>cudnnStatus_t cudnnGetPooling2dForwardOutputDim(const cudnnPoolingDescriptor_t,
const cudnnTensorDescriptor_t, int*, int*, int*, int*)</code><br>
// * <i>native declaration : line 567</i>
// */
//int cudnnGetPooling2dForwardOutputDim(CudnnLibrary.cudnnPoolingDescriptor_t poolingDesc,
CudnnLibrary.cudnnTensorDescriptor_t inputTensorDesc, IntByReference outN, IntByReference outC, IntByReference
outH, IntByReference outW);
///**
// * Destroy an instance of pooling descriptor<br>
// * Original signature : <code>cudnnStatus_t cudnnDestroyPoolingDescriptor(cudnnPoolingDescriptor_t)</code><br>
// * <i>native declaration : line 576</i><br>
```

```
// * @deprecated use the safer methods {@link
#cudnnDestroyPoolingDescriptor(cudnn.CudnnLibrary.cudnnPoolingDescriptor_t)} and {@link
#cudnnDestroyPoolingDescriptor(com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnDestroyPoolingDescriptor(Pointer poolingDesc);
  /**
   * Destroy an instance of pooling descriptor<br>
   * Original signature : <code>cudnnStatus_t cudnnDestroyPoolingDescriptor(cudnnPoolingDescriptor_t)</code><br>
   * <i>native declaration : line 576</i>
   */
  int cudnnDestroyPoolingDescriptor(CudnnLibrary.cudnnPoolingDescriptor_t poolingDesc);
///**
// * Function to perform forward pooling<br>
// * Original signature : <code>cudnnStatus_t cudnnPoolingForward(cudnnHandle_t, const cudnnPoolingDescriptor_t,
const void*, const cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t,
void*)</code><br>
// * <i>native declaration : line 581</i><br>
// * @deprecated use the safer methods {@link #cudnnPoolingForward(cudnn.CudnnLibrary.cudnnHandle_t,
cudnn.CudnnLibrary.cudnnPoolingDescriptor_t, com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t,
com.sun.jna.Pointer, com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer)} and
{@link #cudnnPoolingForward(com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnPoolingForward(Pointer handle, Pointer poolingDesc, ByReference alpha, Pointer srcDesc, Pointer srcData,
ByReference beta, Pointer destDesc, Pointer destData);
  /**
   * Function to perform forward pooling<br>
   * Original signature : <code>cudnnStatus_t cudnnPoolingForward(cudnnHandle_t, const cudnnPoolingDescriptor_t,
const void*, const cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t,
void*)</code><br>
   * <i>native declaration : line 581</i>
   */
  int cudnnPoolingForward(CudnnLibrary.cudnnHandle_t handle, CudnnLibrary.cudnnPoolingDescriptor_t poolingDesc,
ByReference alpha, CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer srcData, ByReference beta,
CudnnLibrary.cudnnTensorDescriptor_t destDesc, Pointer destData);
///**
// * Function to perform backward pooling<br>
// * Original signature : <code>cudnnStatus_t cudnnPoolingBackward(cudnnHandle_t, const cudnnPoolingDescriptor_t,
const void*, const cudnnTensorDescriptor_t, const void*, const cudnnTensorDescriptor_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
// * <i>native declaration : line 592</i><br>
// * @deprecated use the safer methods {@link #cudnnPoolingBackward(cudnn.CudnnLibrary.cudnnHandle_t,
cudnn.CudnnLibrary.cudnnPoolingDescriptor_t, com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t,
com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer)} and {@link
#cudnnPoolingBackward(com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnPoolingBackward(Pointer handle, Pointer poolingDesc, ByReference alpha, Pointer srcDesc, Pointer srcData,
Pointer srcDiffDesc, Pointer srcDiffData, Pointer destDesc, Pointer destData, ByReference beta, Pointer destDiffDesc,
Pointer destDiffData);
  /**
   * Function to perform backward pooling<br>
   * Original signature : <code>cudnnStatus_t cudnnPoolingBackward(cudnnHandle_t, const cudnnPoolingDescriptor_t,
const void*, const cudnnTensorDescriptor_t, const void*, const cudnnTensorDescriptor_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
   * <i>native declaration : line 592</i>
   */
  int cudnnPoolingBackward(CudnnLibrary.cudnnHandle_t handle, CudnnLibrary.cudnnPoolingDescriptor_t poolingDesc,
ByReference alpha, CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer srcData,
CudnnLibrary.cudnnTensorDescriptor_t srcDiffDesc, Pointer srcDiffData, CudnnLibrary.cudnnTensorDescriptor_t
destDesc, Pointer destData, ByReference beta, CudnnLibrary.cudnnTensorDescriptor_t destDiffDesc, Pointer
destDiffData);
///**
// * Function to perform forward activation<br>
// * Original signature : <code>cudnnStatus_t cudnnActivationForward(cudnnHandle_t, cudnnActivationMode_t, const
void*, const cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
// * <i>native declaration : line 619</i><br>
// * @deprecated use the safer methods {@link #cudnnActivationForward(cudnn.CudnnLibrary.cudnnHandle_t, int,
com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer)} and {@link
#cudnnActivationForward(com.sun.jna.Pointer, int, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnActivationForward(Pointer handle, int mode, ByReference alpha, Pointer srcDesc, Pointer srcData,
ByReference beta, Pointer destDesc, Pointer destData);
  /**
   * Function to perform forward activation<br>
   * Original signature : <code>cudnnStatus_t cudnnActivationForward(cudnnHandle_t, cudnnActivationMode_t, const
void*, const cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
   * <i>native declaration : line 619</i>
   */
  int cudnnActivationForward(CudnnLibrary.cudnnHandle_t handle, int mode, ByReference alpha,
CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer srcData, ByReference beta,
CudnnLibrary.cudnnTensorDescriptor_t destDesc, Pointer destData);
///**
// * Function to perform backward activation<br>
// * Original signature : <code>cudnnStatus_t cudnnActivationBackward(cudnnHandle_t, cudnnActivationMode_t, const
void*, const cudnnTensorDescriptor_t, const void*, const cudnnTensorDescriptor_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
// * <i>native declaration : line 630</i><br>
// * @deprecated use the safer methods {@link #cudnnActivationBackward(cudnn.CudnnLibrary.cudnnHandle_t, int,
com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer,
cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t,
com.sun.jna.Pointer, com.sun.jna.Pointer, cudnn.CudnnLibrary.cudnnTensorDescriptor_t, com.sun.jna.Pointer)} and
{@link #cudnnActivationBackward(com.sun.jna.Pointer, int, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer,
com.sun.jna.Pointer, com.sun.jna.Pointer, com.sun.jna.Pointer)} instead
// */
//@Deprecated
//int cudnnActivationBackward(Pointer handle, int mode, ByReference alpha, Pointer srcDesc, Pointer srcData, Pointer
srcDiffDesc, Pointer srcDiffData, Pointer destDesc, Pointer destData, ByReference beta, Pointer destDiffDesc,
Pointer destDiffData);
  /**
   * Function to perform backward activation<br>
   * Original signature : <code>cudnnStatus_t cudnnActivationBackward(cudnnHandle_t, cudnnActivationMode_t, const
void*, const cudnnTensorDescriptor_t, const void*, const cudnnTensorDescriptor_t, const void*, const
cudnnTensorDescriptor_t, const void*, const void*, const cudnnTensorDescriptor_t, void*)</code><br>
   * <i>native declaration : line 630</i>
   */
  int cudnnActivationBackward(CudnnLibrary.cudnnHandle_t handle, int mode, ByReference alpha,
CudnnLibrary.cudnnTensorDescriptor_t srcDesc, Pointer srcData, CudnnLibrary.cudnnTensorDescriptor_t srcDiffDesc,
Pointer srcDiffData, CudnnLibrary.cudnnTensorDescriptor_t destDesc, Pointer destData, ByReference beta,
CudnnLibrary.cudnnTensorDescriptor_t destDiffDesc, Pointer destDiffData);
//public static class cudaStream_t extends PointerType {
//  public cudaStream_t(Pointer address) {
//    super(address);
//  }
//  public cudaStream_t() {
//    super();
//  }
//};
  public static class cudnnTensorDescriptor_t extends PointerType {
    public cudnnTensorDescriptor_t(Pointer address) {
      super(address);
    }
    public cudnnTensorDescriptor_t() {
      super();
    }

    /**
     * Original signature : <code>cudnnStatus_t cudnnSetTensor4dDescriptor(cudnnTensorDescriptor_t,
cudnnTensorFormat_t, cudnnDataType_t, int, int, int, int)</code><br>
     * @param dataType image data type<br>
     * @param n number of inputs (batch size)<br>
     * @param c number of input feature maps<br>
     * @param h height of input section<br>
     * @param w width of input section<br>
     * @throws CudnnException
     */
```

```java
    public void setTensor4dDescriptor(Cudnn cudnn, int format, int dataType, int n, int c, int h, int w) throws
CudnnException
    {
      cudnn.checkError(cudnn.getLibrary().get().cudnnSetTensor4dDescriptor(this, format, dataType, n, c, h, w));
    }

    /**
     * Destroy an instance of Tensor4d descriptor<br>
     * Original signature : <code>cudnnStatus_t cudnnDestroyTensorDescriptor(cudnnTensorDescriptor_t)</code><br>
     * <i>native declaration : line 202</i>
     * @throws CudnnException
     */
    public void destroy(Cudnn cudnn) throws CudnnException
    {
      cudnn.checkError(cudnn.getLibrary().get().cudnnDestroyTensorDescriptor(this));
    }
  };
  public static class cudnnConvolutionDescriptor_t extends PointerType {
    public cudnnConvolutionDescriptor_t(Pointer address) {
      super(address);
    }
    public cudnnConvolutionDescriptor_t() {
      super();
    }

    /**
     * Destroy an instance of convolution descriptor<br>
     * Original signature : <code>cudnnStatus_t
cudnnDestroyConvolutionDescriptor(cudnnConvolutionDescriptor_t)</code><br>
     * <i>native declaration : line 359</i>
     * @throws CudnnException
     */
    public void destroy(Cudnn cudnn) throws CudnnException
    {
      cudnn.checkError(cudnn.getLibrary().get().cudnnDestroyConvolutionDescriptor(this));
    }

    /**
     * Original signature : <code>cudnnStatus_t cudnnSetConvolution2dDescriptor(cudnnConvolutionDescriptor_t, int,
int, int, int, int, int, cudnnConvolutionMode_t)</code><br>
     * @param pad_h zero-padding height<br>
     * @param pad_w zero-padding width<br>
     * @param u vertical filter stride<br>
     * @param v horizontal filter stride<br>
     * @param upscalex upscale the input in x-direction<br>
     * @param upscaley upscale the input in y-direction<br>
     * <i>native declaration : line 300</i>
     * @throws CudnnException
     */
   public void setConvolution2dDescriptor(Cudnn cudnn, int pad_h, int pad_w, int u, int v, int upscalex, int upscaley,
int mode) throws CudnnException
    {
      cudnn.checkError(cudnn.getLibrary().get().cudnnSetConvolution2dDescriptor(this, pad_h, pad_w, u, v,
upscalex, upscaley, mode));
    }
  };
  public static class cudnnFilterDescriptor_t extends PointerType {
    public cudnnFilterDescriptor_t(Pointer address) {
      super(address);
    }
    public cudnnFilterDescriptor_t() {
      super();
    }

    /**
     * Original signature : <code>cudnnStatus_t cudnnDestroyFilterDescriptor(cudnnFilterDescriptor_t)</code><br>
     * <i>native declaration : line 295</i>
     */
    public void destroy(Cudnn cudnn) throws CudnnException
    {
      cudnn.checkError(cudnn.getLibrary().get().cudnnDestroyFilterDescriptor(this));
    }

    /**
     * Original signature : <code>cudnnStatus_t cudnnSetFilter4dDescriptor(cudnnFilterDescriptor_t,
cudnnDataType_t, int, int, int, int)</code><br>
     * @param dataType image data type<br>
     * @param k number of output feature maps<br>
     * @param c number of input feature maps<br>
     * @param h height of each input filter<br>
     * @param w width of  each input fitler<br>
     * <i>native declaration : line 266</i>
     * @throws CudnnException
     */
    public void setFilter4dDescriptor(Cudnn cudnn, int dataType, int k, int c, int h, int w) throws CudnnException
    {
      cudnn.checkError(cudnn.getLibrary().get().cudnnSetFilter4dDescriptor(this, dataType, k, c, h, w));
    }
  };
  public static class cudnnPoolingDescriptor_t extends PointerType {
    public cudnnPoolingDescriptor_t(Pointer address) {
      super(address);
    }
    public cudnnPoolingDescriptor_t() {
      super();
    }
    /**
     * Original signature : <code>cudnnStatus_t cudnnSetPooling2dDescriptor(cudnnPoolingDescriptor_t,
cudnnPoolingMode_t, int, int, int, int, int, int)</code><br>
     * <i>native declaration : line 525</i>
     * @throws CudnnException
     */
   public void setPooling2dDescriptor(Cudnn cudnn, int mode, int windowHeight, int windowWidth, int verticalPadding,
int horizontalPadding, int verticalStride, int horizontalStride) throws CudnnException
    {
      cudnn.checkError(cudnn.getLibrary().get().cudnnSetPooling2dDescriptor(this, mode, windowHeight,
windowWidth, verticalPadding, horizontalPadding, verticalStride, horizontalStride));
    }

    /**
     * Destroy an instance of pooling descriptor<br>
     * Original signature : <code>cudnnStatus_t
cudnnDestroyPoolingDescriptor(cudnnPoolingDescriptor_t)</code><br>
     * <i>native declaration : line 576</i>
     * @throws CudnnException
     */
    public void destroy(Cudnn cudnn) throws CudnnException
    {
      cudnn.checkError(cudnn.getLibrary().get().cudnnDestroyPoolingDescriptor(this));
    }
  };
  public static class cudnnHandle_t extends PointerType {
    public cudnnHandle_t(Pointer address) {
      super(address);
    }
    public cudnnHandle_t() {
      super();
    }
  };

  //v3
  public static interface cudnnConvolutionBwdFilterPreference_t
  {
    public static final int CUDNN_CONVOLUTION_BWD_FILTER_NO_WORKSPACE           = 0;
    public static final int CUDNN_CONVOLUTION_BWD_FILTER_PREFER_FASTEST         = 1;
    public static final int CUDNN_CONVOLUTION_BWD_FILTER_SPECIFY_WORKSPACE_LIMIT = 2;
  }

  public static interface cudnnConvolutionBwdFilterAlgo_t
  {
    public static final int CUDNN_CONVOLUTION_BWD_FILTER_ALGO_0        = 0;  // non-deterministic
    public static final int CUDNN_CONVOLUTION_BWD_FILTER_ALGO_1        = 1;
    public static final int CUDNN_CONVOLUTION_BWD_FILTER_ALGO_FFT      = 2;
  }

  public static interface cudnnConvolutionBwdDataAlgo_t
  {
    public static final int CUDNN_CONVOLUTION_BWD_DATA_ALGO_0        = 0; // non-deterministic
```

```
    public static final int CUDNN_CONVOLUTION_BWD_DATA_ALGO_1        = 1;
    public static final int CUDNN_CONVOLUTION_BWD_DATA_ALGO_FFT      = 2;
  }

  public int cudnnGetConvolutionBackwardFilterAlgorithm(
      cudnnHandle_t handle,
      cudnnTensorDescriptor_t srcDesc,
      cudnnTensorDescriptor_t diffDesc,
      cudnnConvolutionDescriptor_t convDesc,
      cudnnFilterDescriptor_t gradDesc,
      int preference,
      int memoryLimitInbytes,
      IntByReference algo);

  public int cudnnConvolutionBackwardFilter_v3(
      cudnnHandle_t handle,
      ByReference alpha,
      cudnnTensorDescriptor_t srcDesc,
      Pointer srcData,
      cudnnTensorDescriptor_t diffDesc,
      Pointer diffData,
      cudnnConvolutionDescriptor_t convDesc,
      int algo,
      Pointer workSpace,
      int workSpaceSizeInBytes,
      ByReference beta,
      cudnnFilterDescriptor_t gradDesc,
      Pointer gradData);

  public int cudnnConvolutionBackwardData_v3(
      cudnnHandle_t handle,
      ByReference alpha,
      cudnnFilterDescriptor_t filterDesc,
      Pointer filterData,
      cudnnTensorDescriptor_t diffDesc,
      Pointer diffData,
      cudnnConvolutionDescriptor_t convDesc,
      int algo,
      Pointer workSpace,
      int workSpaceSizeInBytes,
      ByReference beta,
      cudnnTensorDescriptor_t gradDesc,
      Pointer gradData);

  public int cudnnGetConvolutionBackwardFilterWorkspaceSize(
      cudnnHandle_t handle,
      cudnnTensorDescriptor_t srcDesc,
      cudnnTensorDescriptor_t diffDesc,
      cudnnConvolutionDescriptor_t convDesc,
      cudnnFilterDescriptor_t gradDesc,
      int algo,
      IntByReference sizeInBytes);

  public int cudnnGetConvolutionBackwardDataWorkspaceSize(
      cudnnHandle_t handle,
      cudnnFilterDescriptor_t filterDesc,
      cudnnTensorDescriptor_t diffDesc,
      cudnnConvolutionDescriptor_t convDesc,
      cudnnTensorDescriptor_t gradDesc,
      int algo,
      IntByReference sizeInBytes);
}
```

FloatType.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn;

/**
 * Types of floating points. The type must be consistent throughout the computation.
 * @author koumura
 *
 */
public enum FloatType
{
  SINGLE(Float.BYTES, CudnnLibrary.cudnnDataType_t.CUDNN_DATA_FLOAT), DOUBLE(Double.BYTES,
CudnnLibrary.cudnnDataType_t.CUDNN_DATA_DOUBLE);

  private int bytes, dataTypeValue;

  private FloatType(int bytes, int dataTypeValue) {
    this.bytes= bytes;
    this.dataTypeValue = dataTypeValue;
  }

  public int getBytes() {
    return bytes;
  }

  public int getDataTypeValue() {
    return dataTypeValue;
  }

}
```

IntType.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn;

/**
 * Types of integers. Used as the type of labels. When the number of labels are more than {@link Byte#MAX_VALUE},
{@link IntType#INT} must be used.
 * The type must be consistent throughout the computation.
 * @author koumura
 *
 */
public enum IntType
{
  BYTE(Byte.BYTES), INT(Integer.BYTES);
```

```
  private int bytes;

  private IntType(int bytes) {
    this.bytes= bytes;
  }

  public int getBytes() {
    return bytes;
  }
}
```

Pointer.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn;

import java.util.LinkedList;

import com.sun.jna.PointerUtils;

/**
 * A class to bridge {@link com.sun.jna.Pointer} and {@link jcuda.Pointer}
 * @author koumura
 *
 */
public class Pointer extends jcuda.Pointer
{
  private com.sun.jna.Pointer jnaPointer;
  private LinkedList<Pointer> child;
  private Pointer parent;

  private Pointer(jcuda.Pointer jcudaPointer, com.sun.jna.Pointer jnaPointer)
  {
    super(jcudaPointer);
    this.jnaPointer = jnaPointer;
  }
  private Pointer(jcuda.Pointer jcudaPointer)
  {
    super(jcudaPointer);
    this.jnaPointer=PointerUtils.fromAddress(address());
  }
  private Pointer()
  {
    super();
    this.jnaPointer=PointerUtils.fromAddress(address());
  }

  private long address(){return getNativePointer()+getByteOffset();}

  public com.sun.jna.Pointer getJnaPointer() {
    return jnaPointer;
  }

  public Pointer withByteOffset(long offset)
  {
    Pointer ch=fromJcuda(super.withByteOffset(offset));
    if(child==null) child=new LinkedList<>();
    child.add(ch);
    ch.parent=this;
    return ch;
  }

  public void free() throws CudaException
  {
    if(parent==null) Cuda.free(this);
  }

  public static Pointer fromJcuda(jcuda.Pointer jcudaPointer)
  {
    if(jcudaPointer==null) return createNull();
    return new Pointer(jcudaPointer);
  }

  public static Pointer createNull()
  {
    return new Pointer();
  }
}
```

PoolingMode.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn;

/**
 * Types of pooling in pooling layers.
 * @author koumura
 *
 */
public enum PoolingMode
{
  MAX(CudnnLibrary.cudnnPoolingMode_t.CUDNN_POOLING_MAX),
  AVE_INCLUDE(CudnnLibrary.cudnnPoolingMode_t.CUDNN_POOLING_AVERAGE_COUNT_INCLUDE_PADDING),
  AVE_EXCLUDE(CudnnLibrary.cudnnPoolingMode_t.CUDNN_POOLING_AVERAGE_COUNT_EXCLUDE_PADDING);

  private int value;

  private PoolingMode(int value) {
    this.value = value;
  }

  public int getValue(){return value;}
}
```

SoftmaxMode.java

```
/*
```

```
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package cudnn;

/**
 * Types of softmax computation.
 * @author koumura
 *
 */
public enum SoftmaxMode {
  /**
   * compute the softmax over all C, H, W for each N<br>
   * <i>native declaration : line 480</i>
   */
  INSTANCE(CudnnLibrary.cudnnSoftmaxMode_t.CUDNN_SOFTMAX_MODE_INSTANCE),

  /**
   * compute the softmax over all C for each H, W, N<br>
   * <i>native declaration : line 481</i>
   */
  CHANNEL(CudnnLibrary.cudnnSoftmaxMode_t.CUDNN_SOFTMAX_MODE_CHANNEL);

  private int value;

  private SoftmaxMode(int value) {
    this.value = value;
  }

  public int getValue(){return value;}
}
```

ErrorSaving.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package errorcomputation;

import java.io.IOException;
import java.nio.file.Path;
import org.w3c.dom.Element;

import utils.XmlUtils;

/**
 * A class for writing errors in an XML file.
 * @author koumura
 *
 */
public class ErrorSaving
{
  /**
   * Writes a Levenshtein error and a matching error in the XML file.
   * @param levenshteinError
   * @param matchingError
   * @param file
   * @throws IOException
   */
  public static void writeXml(double levenshteinError, double matchingError, Path file) throws IOException
  {
    Element rootEl=XmlUtils.rootElement("Errors");
    XmlUtils.addChild(rootEl, "LevenshteinError", levenshteinError);
    XmlUtils.addChild(rootEl, "MatchingError", matchingError);
    XmlUtils.write(rootEl.getOwnerDocument(), file);
  }
}
```

Levenshtein.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package errorcomputation;

import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

import computation.Sequence;
import computation.Sequence.Note;
import utils.ArrayUtils;
import utils.CollectionUtils;

/**
 * A class to compute the Levenshtein error.
 * @author koumura
 *
 */
public class Levenshtein
{
  private static <T> Result compute(ArrayList<T> sequence0, ArrayList<T> sequence1)
  {
    int[] distance=new int[sequence1.size()+1];
    int[] prevDistance=new int[distance.length];
    ArrayList<int[]> source=new ArrayList<>(sequence0.size());
```

```java
    prevDistance[0]=0;
    for(int i1=0; i1<sequence1.size(); ++i1) prevDistance[i1+1]=i1+1;
    for(int i0=0; i0<sequence0.size(); ++i0)
    {
      distance[0]=i0+1;
      int[] src=new int[sequence1.size()];
      source.add(src);
      for(int i1=0; i1<sequence1.size(); ++i1)
      {
        int min=prevDistance[i1];
        src[i1]=0;
        if(!sequence0.get(i0).equals(sequence1.get(i1))) ++min;
        if(prevDistance[i1+1]+1<min)
        {
          min=prevDistance[i1+1]+1;
          src[i1]=1;
        }
        if(distance[i1]+1<min)
        {
          min=distance[i1]+1;
          src[i1]=2;
        }
        distance[i1+1]=min;
      }
      int[] tmp=distance;
      distance=prevDistance;
      prevDistance=tmp;
    }
    //backtrack
    LinkedList<int[]> alignedIndex=new LinkedList<int[]>();
    int i0=sequence0.size()-1, i1=sequence1.size()-1;
    while(i0>=0||i1>=0)
    {
      int s;
      if(i0>=0&&i1>=0) s=source.get(i0)[i1];
      else if(i0>=0) s=1;
      else s=2;
      switch(s)
      {
      case 0:
        alignedIndex.add(new int[]{i0, i1});
        --i0;
        --i1;
        break;
      case 1:
        alignedIndex.add(new int[]{i0, -1});
        --i0;
        break;
      case 2:
        alignedIndex.add(new int[]{-1, i1});
        --i1;
        break;
      }
    }
    Collections.reverse(alignedIndex);
    int finalDistance=ArrayUtils.last(prevDistance);
    Result result=new Result(alignedIndex, finalDistance);
    return result;
  }

  /**
   * @param noteSequence0
   * @param noteSequence1
   * @return Levenshtein distance.
   */
  public static int computeDistance(List<Note> noteSequence0, List<Note> noteSequence1)
  {
    ArrayList<String>
labelSeq0=noteSequence0.stream().map(n->n.getLabel()).collect(CollectionUtils.arrayListCollector());
    ArrayList<String>
labelSeq1=noteSequence1.stream().map(n->n.getLabel()).collect(CollectionUtils.arrayListCollector());
    return compute(labelSeq0, labelSeq1).distance;
  }

  /**
   * @param outputSequence Map of correct sequences to output intervals.
   * @return Levenshtein error.
   */
  public static double computeError(Map<Sequence, ArrayList<Note>> outputSequence)
  {
    int numLabel=0, distance=0;
    for(Sequence seq: outputSequence.keySet())
    {
      numLabel+=seq.getNote().size();
      distance+=computeDistance(seq.getNote(), outputSequence.get(seq));
    }
    return (double)distance/numLabel;
  }

  private static class Result
  {
    private LinkedList<int[]> alignedIndex;
    private int distance;

    private Result(LinkedList<int[]> alignedIndex, int distance)
    {
      this.alignedIndex = alignedIndex;
      this.distance = distance;
    }

    private LinkedList<int[]> getAlignedIndex(){return alignedIndex;}

    private int getDistance(){return distance;}
  }
}
```

Matching.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package errorcomputation;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.function.Function;

import computation.STFTParam;
import computation.Sequence;
import computation.Sequence.LabelList;
import computation.Sequence.Note;
import utils.ArrayUtils;
import utils.CollectionUtils;

/**
 * A class to compute the matching error.
```

```java
 * @author koumura
 *
 */
public class Matching
{
  private static int[] indexArray(ArrayList<int[]> interval, int sequenceLength)
  {
    int[] array=ArrayUtils.createFilled(sequenceLength, -1);
    for(int i=0; i<interval.size(); ++i) Arrays.fill(array, interval.get(i)[0],
interval.get(i)[0]+interval.get(i)[1], i);
    return array;
  }

  private static ArrayList<int[]> remainingInterval(ArrayList<int[]> interval, List<int[]> matching, int
matchingIndex)
  {
    ArrayList<int[]> intervalNext=new ArrayList<int[]>();
    HashSet<Integer> index=new HashSet<Integer>();
    for(int[] m: matching) index.add(m[matchingIndex]);
    for(int i=0; i<interval.size(); ++i) if(!index.contains(i)) intervalNext.add(interval.get(i));
    return intervalNext;
  }

  private static LinkedList<Integer> matchPosition(int[] interval0, int[] interval1)
  {
    int begin=Math.max(interval0[0], interval1[0]);
    int end=Math.min(interval0[0]+interval0[1], interval1[0]+interval1[1]);
    LinkedList<Integer> matchPosition=new LinkedList<Integer>();
    for(int i=begin; i<end; ++i) matchPosition.add(i);
    return matchPosition;
  }
  private static LinkedList<Integer> matchPosition(ArrayList<int[]> correctInterval, ArrayList<int[]>
answerInterval, int[] matching)
  {
    LinkedList<Integer> matchPosition=matchPosition(correctInterval.get(matching[0]),
answerInterval.get(matching[1]));
    if(matchPosition.size()!=matching[2]) System.err.println("matchPosition.size()="+matchPosition.size()+" !=
matching.get2()="+matching[2]);
    return matchPosition;
  }

  /**
   * @return matchPosition
   */
  private static LinkedList<Integer> matchSingleLabel(ArrayList<int[]> correctInterval, ArrayList<int[]>
answerInterval, int sequenceLength)
  {
//  int[] correctIndexArray=indexArray(correctInterval, sequenceLength);
    int[] answerIndexArray=indexArray(answerInterval, sequenceLength);

    ArrayList<ArrayList<int[]>> overlapLength=new ArrayList<>(answerInterval.size());
    for(int a=0; a<answerInterval.size(); ++a) overlapLength.add(new ArrayList<int[]>());
    for(int c=0; c<correctInterval.size(); ++c)
    {
      HashMap<Integer, Integer> overlap=new HashMap<>();
      for(int t=correctInterval.get(c)[0]; t<correctInterval.get(c)[0]+correctInterval.get(c)[1]; ++t)
      {
        int ai=answerIndexArray[t];
        if(ai>=0)
        {
          overlap.putIfAbsent(ai, 0);
          overlap.put(ai, overlap.get(ai)+1);
        }
      }

      if(overlap.size()==0) continue;
      ArrayList<int[]> list=new ArrayList<>(overlap.size());
      for(Integer key: overlap.keySet()) list.add(new int[]{key, overlap.get(key)});
      list.sort((o1, o2)->-(o1[1]-o2[1]));
      for(int i=0; i<list.size()&&list.get(i)[1]==list.get(0)[1]; ++i) overlapLength.get(list.get(i)[0]).add(new
int[]{c, list.get(i)[1]});
    }

    LinkedList<int[]> candidate=new LinkedList<int[]>();  //int[](c, a, overlapLength)
    for(int a=0; a<answerInterval.size(); ++a)
    {
      ArrayList<int[]> ol=overlapLength.get(a);
      ol.sort((o1, o2)->-(o1[1]-o2[1]));
      for(int i=0; i<ol.size()&&ol.get(i)[1]==ol.get(0)[1]; ++i) candidate.add(new int[]{ol.get(i)[0], a,
ol.get(i)[1]});
    }

    int[] correctCandidateSize=new int[correctInterval.size()];
    int[] answerCandidateSize=new int[answerInterval.size()];
    for(int[] m: candidate)
    {
      ++correctCandidateSize[m[0]];
      ++answerCandidateSize[m[1]];
    }

    LinkedList<int[]> matching=new LinkedList<int[]>();
    for(int[] m: candidate)
    {
      if(correctCandidateSize[m[0]]==1&&answerCandidateSize[m[1]]==1) matching.add(m);
    }
    candidate.removeAll(matching);

    LinkedList<Integer> matchPosition=new LinkedList<Integer>();
    for(int[] m: matching)
    {
      LinkedList<Integer> mp=matchPosition(correctInterval, answerInterval, m);
      if(mp.size()!=m[2]) System.err.println("mp.size()="+mp.size()+" != m.get2()="+m[2]);
      matchPosition.addAll(mp);
    }
    if(candidate.size()==0) return matchPosition;

    if(matching.size()==0)
    {
      for(int[] fix: candidate)
      {
        ArrayList<int[]> matchingNext=new ArrayList<int[]>(matching);
        matchingNext.add(fix);

        ArrayList<int[]> correctIntervalNext=remainingInterval(correctInterval, matchingNext, 0);
        ArrayList<int[]> answerIntervalNext=remainingInterval(answerInterval, matchingNext, 1);
        LinkedList<Integer> nextMatchPosition=matchSingleLabel(correctIntervalNext, answerIntervalNext,
sequenceLength);
        nextMatchPosition.addAll(matchPosition(correctInterval, answerInterval, fix));
        if(nextMatchPosition.size()>matchPosition.size()) matchPosition=nextMatchPosition;
      }
      return matchPosition;
    }

    ArrayList<int[]> correctIntervalNext=remainingInterval(correctInterval, matching, 0);
    ArrayList<int[]> answerIntervalNext=remainingInterval(answerInterval, matching, 1);
    matchPosition.addAll(matchSingleLabel(correctIntervalNext, answerIntervalNext, sequenceLength));
    return matchPosition;
  }

  private static HashMap<Integer, ArrayList<int[]>> labelMap(ArrayList<int[]> labelInterval)
  {
    HashMap<Integer, ArrayList<int[]>> labelMap=new HashMap<>();
    for(int[] li: labelInterval)
    {
      labelMap.putIfAbsent(li[2], new ArrayList<>());
      labelMap.get(li[2]).add(new int[]{li[0], li[1]});
    }
    return labelMap;
  }

  /**
   * @param correctSpecLabelInterval int[](begin, length, label)
   * @param answerSpecLabelInterval int[](begin, length, label)
   * @return matchPosition
   */
  private static LinkedList<Integer> compMatching(ArrayList<int[]> correctSpecLabelInterval, ArrayList<int[]>
answerSpecLabelInterval, int sequenceSpecLength)
  {
```

```
    int silentLabel=0;
    for(int[] li: correctSpecLabelInterval) if(li[2]>=silentLabel) silentLabel=li[2]+1;
    for(int[] li: answerSpecLabelInterval) if(li[2]>=silentLabel) silentLabel=li[2]+1;

    HashMap<Integer, ArrayList<int[]>> correctLabelMap=labelMap(correctSpecLabelInterval);
    HashMap<Integer, ArrayList<int[]>> answerLabelMap=labelMap(answerSpecLabelInterval);

    LinkedList<Integer> matchPosition=new LinkedList<Integer>();
    for(int la: correctLabelMap.keySet()) if(answerLabelMap.containsKey(la))
    {
      matchPosition.addAll(matchSingleLabel(new ArrayList<int[]>(correctLabelMap.get(la)), new
ArrayList<int[]>(answerLabelMap.get(la)), sequenceSpecLength));
    }

    int[] correctIndexArray=indexArray(correctSpecLabelInterval, sequenceSpecLength);
    int[] answerIndexArray=indexArray(answerSpecLabelInterval, sequenceSpecLength);
    for(int i=0; i<sequenceSpecLength; ++i) if(correctIndexArray[i]==-1&&answerIndexArray[i]==-1)
matchPosition.add(i);
    return matchPosition;
  }

  /**
   * @param correctSpecLabelInterval int[](begin, length, label)
   * @param answerSpecLabelInterval int[](begin, length, label)
   * @return Total length of matched intervals.
   */
  public static int computeMatchedLength(ArrayList<int[]> correctSpecLabelInterval, ArrayList<int[]>
answerSpecLabelInterval, int sequenceSpecLength)
  {
    return compMatching(correctSpecLabelInterval, answerSpecLabelInterval, sequenceSpecLength).size();
  }

  /**
   * @return Total length of matched intervals.
   */
  public static int computeDistance(Sequence correctSequence, List<Note> outputNoteSequence, STFTParam stftParam,
LabelList labelList)
  {
    Function<Note, int[]> conv=n->{
      int begin=stftParam.spectrogramPosition(n.getPosition());
      int length=stftParam.spectrogramPosition(n.end())-begin;
      int label=labelList.indexOf(n.getLabel());
      return new int[]{begin, length, label};
    };
    ArrayList<int[]>
correctSpecLabelInterval=correctSequence.getNote().stream().map(conv).collect(CollectionUtils.arrayListCollect
or());
    ArrayList<int[]>
outputSpecLabelInterval=outputNoteSequence.stream().map(conv).collect(CollectionUtils.arrayListCollector());
    int sequenceSpecLength=stftParam.spectrogramLength(correctSequence.getLength());
    return sequenceSpecLength-computeMatchedLength(correctSpecLabelInterval, outputSpecLabelInterval,
sequenceSpecLength);
  }

  /**
   * @param outputSequence Map of correct sequences to output intervals.
   * @return Matching error.
   */
  public static double computeError(Map<Sequence, ArrayList<Note>> outputSequence, STFTParam stftParam, LabelList
labelList)
  {
    int specLength=0, distance=0;
    for(Sequence seq: outputSequence.keySet())
    {
      specLength+=stftParam.spectrogramLength(seq.getLength());
      distance+=computeDistance(seq, outputSequence.get(seq), stftParam, labelList);
    }
    return (double)distance/specLength;
  }
}
```

BdLcGs.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package main;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.concurrent.ExecutionException;
import java.util.function.IntBinaryOperator;
import java.util.stream.Collectors;

import javax.sound.sampled.UnsupportedAudioFileException;
import javax.xml.parsers.ParserConfigurationException;

import org.apache.commons.math3.random.MersenneTwister;
import org.xml.sax.SAXException;

import computation.DnnComputation;
import computation.DnnComputation.Config;
import computation.DnnComputation.HyperParam;
import computation.HmmComputation;
import computation.STFTParam;
import computation.Sequence;
import computation.Sequence.LabelList;
import computation.Sequence.Note;
import computation.Thresholding;
import cudnn.Cuda;
import cudnn.CudaException;
import cudnn.CudnnException;
import cudnn.layer.ConvLayer;
import errorcomputation.ErrorSaving;
import errorcomputation.Levenshtein;
import errorcomputation.Matching;
import no.uib.cipr.matrix.NotConvergedException;
import utils.DnnUtils;
import utils.Executor;
import utils.Pair;
import utils.SoundUtils;

/**
 * A class that contains an entry point for training and recognition by the BD -&gt; LC -&gt; GS arrangement.
 * Before the execution, paths to the necessary files must be set according to the users' computation environment.
 * Users can modify these codes to perform the computation matching their own purposes.
 * Step-by-step descriptions are given by comments in {@link #main(Executor)}
 * @author koumura
 */
public class BdLcGs
{
  public static void main(String... arg)
  {
    int numThread=4;
    Executor executor=new Executor(numThread);
    try
    {
```

```java
      main(executor);
      Cuda.deviceReset();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
    finally
    {
      executor.get().shutdown();
    }
  }

  public static void main(Executor executor) throws IOException, UnsupportedAudioFileException,
NotConvergedException, CudaException, CudnnException, SAXException, ParserConfigurationException,
InterruptedException, ExecutionException
  {
    /**********************************************************
     * Files.
     * Change them according to your environment.
     *********************************************************/
    //Cuda.
    Path fileCudnnLibrary=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥cudnn64_70.dll");
    Path fileCudaKernel=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥kernel.cu.ptx");

    //Data.
    Path dirWave=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥Wave");
    Path fileAllSequences=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥Annotations.xml");

    //Outputs.
    Path fileThresholdingParameter=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥ThresholdBdLcGs");
    Path fileDnnParameter=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥WeightBdLcGs");
    Path fileDnnOutput=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥OutputBdLcGs");
    Path fileOutputSequence=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥OutputSequenceBdLcGs.xml");
    Path fileError=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥ErrorBdLcGs.xml");


    /**********************************************************
     * Hyper parameters.
     * These hyper parameters may be determined using cross-validation within training data.
     *********************************************************/
    //Sequences.
    double trainingSequenceLengthSec=120;

    //Spectrogram.
    STFTParam stftParam=new STFTParam(512, 32);
    double frequencyStartHz=1000, frequencyEndHz=8000;
    int dpssParam=4;

    //DNN.
    double inputDataLengthUpperSec=8;
    int localInputHeight=96;
    int numIter=200;
    int batchSizeUpper=2;
    int randomSeed=0;
    int numConvChannel=16;
    int fullConnectionSize=240;

    //HMM.
    double smoothingConstant=1e-4;
    boolean posteriorToObservationProb=false;


    /**********************************************************
     * Other configurations.
     * Note that results will not be deterministic if ConvLayer.BackwardAlgorithm.FAST_NON_DETERMINISTIC is used.
     *********************************************************/
    boolean verbose=true;
    ConvLayer.BackwardAlgorithm backwardAlgorithm=ConvLayer.BackwardAlgorithm.FAST_NON_DETERMINISTIC;



    /**********************************************************
     * Reading sequences.
     *********************************************************/
    MersenneTwister random=new MersenneTwister(randomSeed);
    ArrayList<Sequence> allSequence=Sequence.readXml(fileAllSequences);
    int
samplingRate=(int)SoundUtils.checkSamplingRate(allSequence.stream().map(s->s.getWaveFileName()).collect(Collec
tors.toList()), dirWave);
    LabelList labelList=Sequence.LabelList.create(allSequence);
    int trainingSequenceLength=(int)(trainingSequenceLengthSec*samplingRate);
    Pair<ArrayList<Sequence>, ArrayList<Sequence>> divided=Sequence.extract(trainingSequenceLength, random,
allSequence);
    ArrayList<Sequence> trainingSequence=divided.get0();
    ArrayList<Sequence> validationSequence=divided.get1();

    /**********************************************************
     * Boundary detection.
     *********************************************************/
    int freqOffset=(int)(frequencyStartHz/stftParam.unitFrequency(samplingRate));
    int freqLength=(int)(frequencyEndHz/stftParam.unitFrequency(samplingRate))-freqOffset;
    int gapLengthLowerUpper=localInputHeight, noteLengthLowerUpper=localInputHeight;
    Thresholding.HyperParameter thresholdingHyperParameter=new Thresholding.HyperParameter(stftParam, dpssParam,
freqOffset, freqLength, gapLengthLowerUpper, noteLengthLowerUpper);
    Thresholding.Config thresholdingConfig=new Thresholding.Config(executor, dirWave, verbose);

    //Training.
    Thresholding.Parameter thresholdingParameter=Thresholding.train(trainingSequence,
thresholdingHyperParameter, thresholdingConfig);

    //Parameter saving.
    Files.createDirectories(fileThresholdingParameter.getParent());
    thresholdingParameter.writeXml(fileThresholdingParameter);

    //Parameter loading.
    //Uncomment to use pre-trained parameters.
//  Thresholding.Parameter thresholdingParameter=Thresholding.Parameter.parseXml(fileThresholdingParameter);

    //Boundary detection in the validation data.
    HashMap<Sequence, ArrayList<int[]>> soundInterval=Thresholding.boundaryDetection(validationSequence,
thresholdingParameter, thresholdingHyperParameter, thresholdingConfig);


    /**********************************************************
     * Local classification.
     *********************************************************/
    int inputHeightUpper=stftParam.spectrogramLength((int)(inputDataLengthUpperSec*samplingRate));
    IntBinaryOperator silentLabelFunc=(numUpperLabel, numLowreLabel)->-1;
    IntBinaryOperator softmaxSizeFunc=(numUpperLabel, numLowreLabel)->numUpperLabel*numLowreLabel;
    HyperParam dnnHyperParam=new HyperParam(stftParam, dpssParam, localInputHeight, localInputHeight, 1,
freqOffset, freqLength, inputHeightUpper, batchSizeUpper, numIter, numConvChannel, fullConnectionSize);

    //Computing mean & sd of training spectrogram for input normalization.
    HashMap<Sequence, float[]> spectrogram=SoundUtils.spectrogram(Sequence.wavePositionMap(trainingSequence,
dirWave), stftParam, dpssParam, freqOffset, freqLength);
    double[]
specMeanSd=SoundUtils.spectrogramMeanSd(trainingSequence.stream().map(s->spectrogram.get(s)).collect(Collector
s.toList()));
    Config dnnConfig=new Config(verbose, silentLabelFunc, softmaxSizeFunc, specMeanSd, fileCudaKernel,
fileCudnnLibrary, dirWave, backwardAlgorithm);

    //Training.
    DnnComputation.Param dnnParam=DnnComputation.training(trainingSequence, labelList, random, dnnHyperParam,
dnnConfig);

    //Parameter saving.
    Files.createDirectories(fileDnnParameter.getParent());
    DnnUtils.saveParam(dnnParam.getLayerParam(), fileDnnParameter);

    //Parameter loading.
    //Uncomment to use pre-computed parameters.
//  DnnComputation.Param dnnParam=new DnnComputation.Param(DnnUtils.loadParam(fileDnnParameter));

    //Local recognition in the validation data.
    HashMap<Sequence, float[]> dnnOutput=DnnComputation.recognition(validationSequence, labelList, dnnParam,
dnnHyperParam, dnnConfig);
```

```
    //Saving DNN output.
    Files.createDirectories(fileDnnOutput.getParent());
    DnnUtils.saveOutput(dnnOutput, fileDnnOutput);


    /*********************************************************
     * Global sequencing.
     *********************************************************/
    HmmComputation.HyperParam hmmHyperParam=new HmmComputation.HyperParam(smoothingConstant,
posteriorToObservationProb, 1, stftParam);
    HashMap<Sequence, ArrayList<double[]>> observationProb=Thresholding.averageOutput(soundInterval, dnnOutput,
stftParam);
    observationProb=HmmComputation.segmentedPosteriorToObservationProb(observationProb, trainingSequence,
labelList, hmmHyperParam);
    HmmComputation.Config hmmConfig=new HmmComputation.Config(executor, 0);

    //Training
    double[][][] transitionProb=HmmComputation.transitionProbability(trainingSequence, labelList,
hmmHyperParam);

    //Global sequencing in the validation data.
    HashMap<Sequence, ArrayList<Note>> outputSequence=HmmComputation.globalSequencing(observationProb,
transitionProb, labelList, soundInterval, hmmHyperParam, hmmConfig);

    //Saving output sequences.
    Files.createDirectories(fileOutputSequence.getParent());
    Sequence.writeOutputSequence(outputSequence, fileOutputSequence);


    /*********************************************************
     * Error computation.
     *********************************************************/
    double levenshteinError=Levenshtein.computeError(outputSequence);
    double matchingError=Matching.computeError(outputSequence, stftParam, labelList);
    System.out.printf("Levenshtein error =¥t%.2f%%", (levenshteinError*100));
    System.out.println();
    System.out.printf("Matching error =¥t%.2f%%", (matchingError*100));
    System.out.println();
    Files.createDirectories(fileError.getParent());
    ErrorSaving.writeXml(levenshteinError, matchingError, fileError);
  }
}
```

LcBdGs.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package main;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.concurrent.ExecutionException;
import java.util.function.IntBinaryOperator;
import java.util.stream.Collectors;

import javax.sound.sampled.UnsupportedAudioFileException;
import javax.xml.parsers.ParserConfigurationException;

import org.apache.commons.math3.random.MersenneTwister;
import org.xml.sax.SAXException;

import computation.DnnComputation;
import computation.DnnComputation.Config;
import computation.DnnComputation.HyperParam;
import computation.HmmComputation;
import computation.STFTParam;
import computation.Sequence;
import computation.Sequence.LabelList;
import computation.Sequence.Note;
import cudnn.Cuda;
import cudnn.CudaException;
import cudnn.CudnnException;
import cudnn.layer.ConvLayer;
import errorcomputation.ErrorSaving;
import errorcomputation.Levenshtein;
import errorcomputation.Matching;
import no.uib.cipr.matrix.NotConvergedException;
import utils.DnnUtils;
import utils.Executor;
import utils.Pair;
import utils.SoundUtils;

/**
 * A class that contains an entry point for training and recognition by the LC -&gt; BD & GS arrangement.
 * Before the execution, paths to the necessary files must be set according to the users' computation environment.
 * Users can modify these codes to perform the computation matching their own purposes.
 * Step-by-step descriptions are given by comments in {@link #main(Executor)}
 * @author koumura
 */
public class LcBdGs
{
  public static void main(String... arg)
  {
    int numThread=4;
    Executor executor=new Executor(numThread);
    try
    {
      main(executor);
      Cuda.deviceReset();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
    finally
    {
      executor.get().shutdown();
    }
  }

  public static void main(Executor executor) throws IOException, UnsupportedAudioFileException,
NotConvergedException, CudaException, CudnnException, SAXException, ParserConfigurationException,
InterruptedException, ExecutionException
  {
    /*********************************************************
     * Files.
     * Change them according to your environment.
     *********************************************************/
    //Cuda.
    Path fileCudnnLibrary=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥cudnn64_70.dll");
    Path fileCudaKernel=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥kernel.cu.ptx");

    //Data.
    Path dirWave=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥Wave");
    Path fileAllSequences=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥Annotations.xml");
```

```
    //Outputs.
    Path fileDnnParameter=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥WeightLcBdGs");
    Path fileDnnOutput=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥OutputLcBdGs");
    Path fileOutputSequence=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥OutputSequenceLcBdGs.xml");
    Path fileError=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥ErrorLcBdGs.xml");


    /**********************************************************
     * Hyper parameters.
     * These hyper parameters may be determined using cross-validation within training data.
     *********************************************************/
    //Sequences.
    double trainingSequenceLengthSec=120;

    //Spectrogram.
    STFTParam stftParam=new STFTParam(512, 32);
    double frequencyStartHz=1000, frequencyEndHz=8000;
    int dpssParam=4;

    //DNN.
    double inputDataLengthUpperSec=8;
    int localInputHeight=96;
    int numIter=200;
    int batchSizeUpper=2;
    int randomSeed=0;
    int numConvChannel=16;
    int fullConnectionSize=240;

    //HMM.
    double smoothingConstant=1e-4;
    boolean posteriorToObservationProb=false;


    /**********************************************************
     * Other configurations.
     * Note that results will not be deterministic if ConvLayer.BackwardAlgorithm.FAST_NON_DETERMINISTIC is used.
     *********************************************************/
    boolean verbose=true;
    ConvLayer.BackwardAlgorithm backwardAlgorithm=ConvLayer.BackwardAlgorithm.FAST_NON_DETERMINISTIC;



    /**********************************************************
     * Reading sequences.
     *********************************************************/
    MersenneTwister random=new MersenneTwister(randomSeed);
    ArrayList<Sequence> allSequence=Sequence.readXml(fileAllSequences);
    int
samplingRate=(int)SoundUtils.checkSamplingRate(allSequence.stream().map(s->s.getWaveFileName()).collect(Collec
tors.toList()), dirWave);
    LabelList labelList=Sequence.LabelList.create(allSequence);
    int trainingSequenceLength=(int)(trainingSequenceLengthSec*samplingRate);
    Pair<ArrayList<Sequence>, ArrayList<Sequence>> divided=Sequence.extract(trainingSequenceLength, random,
allSequence);
    ArrayList<Sequence> trainingSequence=divided.get0();
    ArrayList<Sequence> validationSequence=divided.get1();


    /**********************************************************
     * Local classification.
     *********************************************************/
    int freqOffset=(int)(frequencyStartHz/stftParam.unitFrequency(samplingRate));
    int freqLength=(int)(frequencyEndHz/stftParam.unitFrequency(samplingRate))-freqOffset;
    int inputHeightUpper=stftParam.spectrogramLength((int)(inputDataLengthUpperSec*samplingRate));
    int numSubLabel=3;
    IntBinaryOperator silentLabelFunc=(numUpperLabel, numLowreLabel)->numUpperLabel*numLowreLabel;
    IntBinaryOperator softmaxSizeFunc=(numUpperLabel, numLowreLabel)->numUpperLabel*numLowreLabel+1;
    HyperParam dnnHyperParam=new HyperParam(stftParam, dpssParam, localInputHeight, localInputHeight, numSubLabel,
freqOffset, freqLength, inputHeightUpper, batchSizeUpper, numIter, numConvChannel, fullConnectionSize);

    //Computing mean & sd of training spectrogram for input normalization.
    HashMap<Sequence, float[]> spectrogram=SoundUtils.spectrogram(Sequence.wavePositionMap(trainingSequence,
dirWave), stftParam, dpssParam, freqOffset, freqLength);
    double[]
specMeanSd=SoundUtils.spectrogramMeanSd(trainingSequence.stream().map(s->spectrogram.get(s)).collect(Collector
s.toList()));
    Config dnnConfig=new Config(verbose, silentLabelFunc, softmaxSizeFunc, specMeanSd, fileCudaKernel,
fileCudnnLibrary, dirWave, backwardAlgorithm);

    //Training.
    DnnComputation.Param dnnParam=DnnComputation.training(trainingSequence, labelList, random, dnnHyperParam,
dnnConfig);

    //Parameter saving.
    Files.createDirectories(fileDnnParameter.getParent());
    DnnUtils.saveParam(dnnParam.getLayerParam(), fileDnnParameter);

    //Parameter loading.
    //Uncomment to use pre-computed parameters.
//  DnnComputation.Param dnnParam=new DnnComputation.Param(DnnUtils.loadParam(fileDnnParameter));

    //Local recognition in the validation data.
    HashMap<Sequence, float[]> dnnOutput=DnnComputation.recognition(validationSequence, labelList, dnnParam,
dnnHyperParam, dnnConfig);

    //Saving DNN output.
    Files.createDirectories(fileDnnOutput.getParent());
    DnnUtils.saveOutput(dnnOutput, fileDnnOutput);


    /**********************************************************
     * Boundary detection & global sequencing.
     *********************************************************/
    HmmComputation.HyperParam hmmHyperParam=new HmmComputation.HyperParam(smoothingConstant,
posteriorToObservationProb, numSubLabel, stftParam);
    HmmComputation.Config hmmConfig=new HmmComputation.Config(executor,
softmaxSizeFunc.applyAsInt(labelList.size(), numSubLabel));
    HashMap<Sequence, ArrayList<double[]>>
observationProb=HmmComputation.continuousPosteriorToObservationProb(dnnOutput, trainingSequence, labelList,
hmmHyperParam, hmmConfig);

    //Training
    double[][][] transitionProb=HmmComputation.transitionProbability(trainingSequence, labelList,
hmmHyperParam);

    //Global sequencing in the validation data.
    HashMap<Sequence, ArrayList<Note>>
outputSequence=HmmComputation.globalSequencingWithBoundaryDetection(observationProb, transitionProb, labelList,
hmmHyperParam, hmmConfig);

    //Saving output sequences.
    Files.createDirectories(fileOutputSequence.getParent());
    Sequence.writeOutputSequence(outputSequence, fileOutputSequence);


    /**********************************************************
     * Error computation.
     *********************************************************/
    double levenshteinError=Levenshtein.computeError(outputSequence);
    double matchingError=Matching.computeError(outputSequence, stftParam, labelList);
    System.out.printf("Levenshtein error =¥t%.2f%%", (levenshteinError*100));
    System.out.println();
    System.out.printf("Matching error =¥t%.2f%%", (matchingError*100));
    System.out.println();
    Files.createDirectories(fileError.getParent());
    ErrorSaving.writeXml(levenshteinError, matchingError, fileError);
  }
}
```

LcGsBdGs.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
```

```java
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package main;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.concurrent.ExecutionException;
import java.util.function.IntBinaryOperator;
import java.util.stream.Collectors;

import javax.sound.sampled.UnsupportedAudioFileException;
import javax.xml.parsers.ParserConfigurationException;

import org.apache.commons.math3.random.MersenneTwister;
import org.xml.sax.SAXException;

import computation.DnnComputation;
import computation.DnnComputation.Config;
import computation.DnnComputation.HyperParam;
import computation.HmmComputation;
import computation.STFTParam;
import computation.Sequence;
import computation.Sequence.LabelList;
import computation.Sequence.Note;
import cudnn.Cuda;
import cudnn.CudaException;
import cudnn.CudnnException;
import cudnn.layer.ConvLayer;
import errorcomputation.ErrorSaving;
import errorcomputation.Levenshtein;
import errorcomputation.Matching;
import no.uib.cipr.matrix.NotConvergedException;
import utils.DnnUtils;
import utils.Executor;
import utils.Pair;
import utils.SoundUtils;

/**
 * A class that contains an entry point for training and recognition by the LC & GS -&gt; BD & GS arrangement.
 * Before the execution, paths to the necessary files must be set according to the users' computation environment.
 * Users can modify these codes to perform the computation matching their own purposes.
 * Step-by-step descriptions are given by comments in {@link #main(Executor)}
 * @author koumura
 */
public class LcGsBdGs
{
  public static void main(String... arg)
  {
    int numThread=4;
    Executor executor=new Executor(numThread);
    try
    {
      main(executor);
      Cuda.deviceReset();
    }
    catch(Exception e)
    {
      e.printStackTrace();
    }
    finally
    {
      executor.get().shutdown();
    }
  }

  public static void main(Executor executor) throws IOException, UnsupportedAudioFileException,
NotConvergedException, CudaException, CudnnException, SAXException, ParserConfigurationException,
InterruptedException, ExecutionException
  {
    /*********************************************************
     * Files.
     * Change them according to your environment.
     *********************************************************/
    //Cuda.
    Path fileCudnnLibrary=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥cudnn64_70.dll");
    Path fileCudaKernel=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥kernel.cu.ptx");

    //Data.
    Path dirWave=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥Wave");
    Path fileAllSequences=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥AllSequences.xml");

    //Outputs.
    Path fileDnnParameter=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥WeightLcGsBdGs");
    Path fileDnnOutput=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥OutputLcGsBdGs");
    Path fileOutputSequence=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥OutputSequenceLcGsBdGs.xml");
    Path fileError=Paths.get("C:¥¥path¥¥in¥¥your¥¥deivce¥¥Bird0¥¥ErrorLcGsBdGs.xml");


    /*********************************************************
     * Hyper parameters.
     * These hyper parameters may be determined using cross-validation within training data.
     *********************************************************/
    //Sequences.
    double trainingSequenceLengthSec=120;

    //Spectrogram.
    STFTParam stftParam=new STFTParam(512, 32);
    double frequencyStartHz=1000, frequencyEndHz=8000;
    int dpssParam=4;

    //DNN.
    double inputDataLengthUpperSec=8;
    int localInputHeight=96;
    int finalInputHeight=localInputHeight*3;
    int numIter=200;
    int batchSizeUpper=2;
    int randomSeed=0;
    int numConvChannel=16;
    int fullConnectionSize=240;

    //HMM.
    double smoothingConstant=1e-4;
    boolean posteriorToObservationProb=false;


    /*********************************************************
     * Other configurations.
     * Note that results will not be deterministic if ConvLayer.BackwardAlgorithm.FAST_NON_DETERMINISTIC is used.
     *********************************************************/
    boolean verbose=true;
    ConvLayer.BackwardAlgorithm backwardAlgorithm=ConvLayer.BackwardAlgorithm.FAST_NON_DETERMINISTIC;



    /*********************************************************
     * Reading sequences.
     *********************************************************/
    MersenneTwister random=new MersenneTwister(randomSeed);
    ArrayList<Sequence> allSequence=Sequence.readXml(fileAllSequences);
```

```
    int
samplingRate=(int)SoundUtils.checkSamplingRate(allSequence.stream().map(s->s.getWaveFileName()).collect(Collec
tors.toList()), dirWave);
    LabelList labelList=Sequence.LabelList.create(allSequence);
    int trainingSequenceLength=(int)(trainingSequenceLengthSec*samplingRate);
    Pair<ArrayList<Sequence>, ArrayList<Sequence>> divided=Sequence.extract(trainingSequenceLength, random,
allSequence);
    ArrayList<Sequence> trainingSequence=divided.get0();
    ArrayList<Sequence> validationSequence=divided.get1();


    /**********************************************************
     * Local classification & global sequencing.
     **********************************************************/
    int freqOffset=(int)(frequencyStartHz/stftParam.unitFrequency(samplingRate));
    int freqLength=(int)(frequencyEndHz/stftParam.unitFrequency(samplingRate))-freqOffset;
    int inputHeightUpper=stftParam.spectrogramLength((int)(inputDataLengthUpperSec*samplingRate));
    int numSubLabel=3;
    IntBinaryOperator silentLabelFunc=(numUpperLabel, numLowreLabel)->numUpperLabel*numLowreLabel;
    IntBinaryOperator softmaxSizeFunc=(numUpperLabel, numLowreLabel)->numUpperLabel*numLowreLabel+1;
    HyperParam dnnHyperParam=new HyperParam(stftParam, dpssParam, localInputHeight, finalInputHeight, numSubLabel,
freqOffset, freqLength, inputHeightUpper, batchSizeUpper, numIter, numConvChannel, fullConnectionSize);

    //Computing mean & sd of training spectrogram for input normalization.
    HashMap<Sequence, float[]> spectrogram=SoundUtils.spectrogram(Sequence.wavePositionMap(trainingSequence,
dirWave), stftParam, dpssParam, freqOffset, freqLength);
    double[]
specMeanSd=SoundUtils.spectrogramMeanSd(trainingSequence.stream().map(s->spectrogram.get(s)).collect(Collector
s.toList()));
    Config dnnConfig=new Config(verbose, silentLabelFunc, softmaxSizeFunc, specMeanSd, fileCudaKernel,
fileCudnnLibrary, dirWave, backwardAlgorithm);

    //Training.
    DnnComputation.Param dnnParam=DnnComputation.training(trainingSequence, labelList, random, dnnHyperParam,
dnnConfig);

    //Parameter saving.
    Files.createDirectories(fileDnnParameter.getParent());
    DnnUtils.saveParam(dnnParam.getLayerParam(), fileDnnParameter);

    //Parameter loading.
    //Uncomment to use pre-computed parameters.
//  DnnComputation.Param dnnParam=new DnnComputation.Param(DnnUtils.loadParam(fileDnnParameter));

    //Local recognition in the validation data.
    HashMap<Sequence, float[]> dnnOutput=DnnComputation.recognition(validationSequence, labelList, dnnParam,
dnnHyperParam, dnnConfig);

    //Saving DNN output.
    Files.createDirectories(fileDnnOutput.getParent());
    DnnUtils.saveOutput(dnnOutput, fileDnnOutput);


    /**********************************************************
     * Boundary detection & global sequencing.
     **********************************************************/
    HmmComputation.HyperParam hmmHyperParam=new HmmComputation.HyperParam(smoothingConstant,
posteriorToObservationProb, numSubLabel, stftParam);
    HmmComputation.Config hmmConfig=new HmmComputation.Config(executor,
softmaxSizeFunc.applyAsInt(labelList.size(), numSubLabel));
    HashMap<Sequence, ArrayList<double[]>>
observationProb=HmmComputation.continuousPosteriorToObservationProb(dnnOutput, trainingSequence, labelList,
hmmHyperParam, hmmConfig);

    //Training
    double[][][] transitionProb=HmmComputation.transitionProbability(trainingSequence, labelList,
hmmHyperParam);

    //Global sequencing in the validation data.
    HashMap<Sequence, ArrayList<Note>>
outputSequence=HmmComputation.globalSequencingWithBoundaryDetection(observationProb, transitionProb, labelList,
hmmHyperParam, hmmConfig);

    //Saving output sequences.
    Files.createDirectories(fileOutputSequence.getParent());
    Sequence.writeOutputSequence(outputSequence, fileOutputSequence);


    /**********************************************************
     * Error computation.
     **********************************************************/
    double levenshteinError=Levenshtein.computeError(outputSequence);
    double matchingError=Matching.computeError(outputSequence, stftParam, labelList);
    System.out.printf("Levenshtein error =¥t%.2f%%", (levenshteinError*100));
    System.out.println();
    System.out.printf("Matching error =¥t%.2f%%", (matchingError*100));
    System.out.println();
    Files.createDirectories(fileError.getParent());
    ErrorSaving.writeXml(levenshteinError, matchingError, fileError);
  }
}
```

ArrayUtils.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package utils;

import java.util.Arrays;

public class ArrayUtils
{
  public static int[] createSequence(int start, int end)
  {
    int[] array=new int[end-start];
    for(int i=0; i<array.length; ++i) array[i]=i+start;
    return array;
  }

  public static double[] createFilled(int size, double value)
  {
    double[] array=new double[size];
    Arrays.fill(array, value);
    return array;
  }
  public static int[] createFilled(int size, int value)
  {
    int [] array=new int[size];
    Arrays.fill(array, value);
    return array;
  }

  public static int last(int[] array){return array[array.length-1];}

  public static int sum01(int[] array){return array[0]+array[1];}
}
```

CollectionUtils.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package utils;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.stream.Collector;
import java.util.stream.Collectors;

public class CollectionUtils
{
  public static <T> Collector<? super T, ?, ArrayList<T>> arrayListCollector()
  {
    return Collectors.toCollection(ArrayList::new);
  }

  public static ArrayList<int[]> deepCopy(ArrayList<int[]> list)
  {
    return list.stream().map(s->Arrays.copyOf(s, s.length)).collect(arrayListCollector());
  }

  public static <T> T last(ArrayList<T> list)
  {
    return list.get(list.size()-1);
  }
}
```

DnnUtils.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package utils;

import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.ListIterator;

import org.apache.commons.math3.random.MersenneTwister;

import computation.STFTParam;
import computation.Sequence;
import cudnn.ActivationMode;
import cudnn.CudaDriver;
import cudnn.CudaException;
import cudnn.Cudnn;
import cudnn.CudnnException;
import cudnn.FloatType;
import cudnn.IntType;
import cudnn.PoolingMode;
import cudnn.layer.ConvLayer;
import cudnn.layer.DataLayer;
import cudnn.layer.Layer;
import cudnn.layer.ParamLayer;
import cudnn.layer.PoolLayer;
import cudnn.layer.SeqSoftmaxConvLayer;
import cudnn.network.SeqNetwork;

public class DnnUtils
{
  public static enum LayerType
  {
    CONV, POOLING;
  }

  public static int nextSmallestInputSize(List<Pair<LayerType, Integer>> filterSize, int inputSize)
  {
    int scale=1, addition=0;
    ListIterator<Pair<LayerType, Integer>> iterator=filterSize.listIterator(filterSize.size());
    while(iterator.hasPrevious())
    {
      Pair<LayerType, Integer> fs=iterator.previous();
      if(fs.get0()==LayerType.CONV)
      {
        addition+=fs.get1()-1;
      }
      else if(fs.get0()==LayerType.POOLING)
      {
        addition*=fs.get1();
        scale*=fs.get1();
      }
    }
    if(inputSize<=addition) return addition+scale;
    return scale*MathUtils.ceil(inputSize-addition, scale)+addition;
  }

  public static int sumStride(List<Pair<LayerType, Integer>> filterSize)
  {
    int stride=1;
    for(Pair<LayerType, Integer> fs: filterSize)
    {
      if(fs.get0()==LayerType.POOLING) stride*=fs.get1();
    }
    return stride;
  }

  public static ArrayList<Pair<LayerType, Integer>> filterSizeList525242()
  {
    ArrayList<Pair<LayerType, Integer>> filterSizeList=new ArrayList<>();
    filterSizeList.add(new Pair<>(LayerType.CONV, 5));
    filterSizeList.add(new Pair<>(LayerType.POOLING, 2));
    filterSizeList.add(new Pair<>(LayerType.CONV, 5));
    filterSizeList.add(new Pair<>(LayerType.POOLING, 2));
    filterSizeList.add(new Pair<>(LayerType.CONV, 4));
    filterSizeList.add(new Pair<>(LayerType.POOLING, 2));
    return filterSizeList;
```

```
  }

  /**
   * Creates a CDNN used in this program.
   * @param dataLayerHeight Height of the data layer.
   * @param inputWidth Width of the input spectrogram.
   * @param softMaxSize Number of the output labels.
   * @param batchSize
   * @param cudnn
   * @param driver
   * @param localInputHeight Height of the input in local recognition.
   * @param finalInputHeight Height of the input in global sequencing. In BD -gt; LC -gt; GS and LC -gt; BD & GS
arrangements, it should be the same as localInputHeight.
   * @param backwardAlgo Algorithm for back-propagation in convolutional layers.
   * @param numConvChannel Number of channels in convolutional layers.
   * @param fullConnectionSize Dimension of the lower fully-connected layer.
   * @return CDNN.
   * @throws CudnnException
   * @throws CudaException
   */
  public static SeqNetwork createNetwork(int dataLayerHeight, int inputWidth, int softMaxSize, int batchSize, Cudnn
cudnn, CudaDriver driver, int localInputHeight, int finalInputHeight, ConvLayer.BackwardAlgorithm backwardAlgo,
int numConvChannel, int fullConnectionSize) throws CudnnException, CudaException
  {
    DataLayer dataLayer=new DataLayer(1, dataLayerHeight, inputWidth);
    ConvLayer conv1=new ConvLayer(numConvChannel, 5, 5, ActivationMode.RELU, dataLayer, backwardAlgo);
    ConvLayer conv11=new ConvLayer(numConvChannel, 1, 1, ActivationMode.RELU, conv1, backwardAlgo);
    PoolLayer pool1=new PoolLayer(2, 2, PoolingMode.MAX, conv11);
    ConvLayer conv2=new ConvLayer(numConvChannel, 5, 5, ActivationMode.RELU, pool1, backwardAlgo);
    ConvLayer conv21=new ConvLayer(numConvChannel, 1, 1, ActivationMode.RELU, conv2, backwardAlgo);
    PoolLayer pool2=new PoolLayer(2, 2, PoolingMode.MAX, conv21);
    ConvLayer conv3=new ConvLayer(numConvChannel, 4, 4, ActivationMode.RELU, pool2, backwardAlgo);
    ConvLayer conv31=new ConvLayer(numConvChannel, 1, 1, ActivationMode.RELU, conv3, backwardAlgo);
    PoolLayer pool3=new PoolLayer(2, 2, PoolingMode.MAX, conv31);
    int
fullFilterHeight=(((localInputHeight-conv1.getFilterHeight()+1)/pool1.getStride()-conv2.getFilterHeight()+1)/p
ool2.getStride()-conv3.getFilterHeight()+1)/pool3.getStride();
    ConvLayer full=new ConvLayer(fullConnectionSize, fullFilterHeight, 11, ActivationMode.RELU, pool3,
backwardAlgo);
    ConvLayer globalFull=null;
    if(finalInputHeight>localInputHeight)
    {
      int globalFullFilterHeight=new STFTParam(localInputHeight,
pool1.getStride()*pool2.getStride()*pool3.getStride()).spectrogramLength(finalInputHeight);
      globalFull=new ConvLayer(fullConnectionSize, globalFullFilterHeight, 1, ActivationMode.RELU, full,
backwardAlgo);
    }
    SeqSoftmaxConvLayer smLayer=new SeqSoftmaxConvLayer(softMaxSize, 1, 1, 1, 1,
finalInputHeight>localInputHeight?globalFull:full, backwardAlgo);

    ArrayList<Layer> layer=new ArrayList<>();
    layer.add(dataLayer);
    layer.add(conv1);
    layer.add(conv11);
    layer.add(pool1);
    layer.add(conv2);
    layer.add(conv21);
    layer.add(pool2);
    layer.add(conv3);
    layer.add(conv31);
    layer.add(pool3);
    layer.add(full);
    if(finalInputHeight>localInputHeight) layer.add(globalFull);
    layer.add(smLayer);

    SeqNetwork network=new SeqNetwork(FloatType.SINGLE, layer, batchSize, cudnn, IntType.BYTE, -1, 1);
    network.init(driver, cudnn);

    return network;
  }

  /**
   * Initializes network parameters according to http://arxiv.org/abs/1502.01852.
   * @see http://arxiv.org/abs/1502.01852
   * @param layer
   * @param random
   */
  public static void initParam(List<Layer> layer, MersenneTwister random)
  {
    for(Layer la: layer) if(la instanceof ParamLayer)
    {
      int size=0;
      ConvLayer cl=(ConvLayer)la;
      size=cl.getLower().getNumChannel()*cl.getFilterHeight()*cl.getFilterHeight();
      double scale=Math.sqrt(2.0/size);
      for(int i=0; i<cl.getWeightF().length; ++i) cl.getWeightF()[i]=(float)(random.nextGaussian()*scale);
      for(int i=0; i<cl.getBiasF().length; ++i) cl.getBiasF()[i]=0;
    }
  }

  public static ArrayList<float[]> copyParamFromLayer(ArrayList<Layer> layer)
  {
    ArrayList<float[]> param=new ArrayList<float[]>();
    for(Layer la: layer) if(la instanceof ParamLayer)
    {
      ParamLayer pl=(ParamLayer)la;
      param.add(Arrays.copyOf(pl.getWeightF(), pl.getWeightSize()));
      param.add(Arrays.copyOf(pl.getBiasF(), pl.getBiasSize()));
    }
    return param;
  }

  public static void copyParamToLayer(ArrayList<float[]> src, ArrayList<Layer> dst)
  {
    {
      if((int)dst.stream().filter(la->la instanceof ParamLayer).count()*2!=src.size())
      {
        System.err.println("Parameters were not copied because of the inconsistent network structure.");
        return;
      }
      int paramIndex=0;
      for(Layer la: dst) if(la instanceof ParamLayer)
      {
        ParamLayer pl=(ParamLayer)la;
        if(pl.getWeightSize()!=src.get(paramIndex++).length||pl.getBiasSize()!=src.get(paramIndex++).length)
        {
          System.err.println("Parameters were not copied because of the inconsistent network structure.");
          return;
        }
      }
    }
    int paramIndex=0;
    for(Layer la: dst) if(la instanceof ParamLayer)
    {
      ParamLayer pl=(ParamLayer)la;
      System.arraycopy(src.get(paramIndex++), 0, pl.getWeightF(), 0, pl.getWeightSize());
      System.arraycopy(src.get(paramIndex++), 0, pl.getBiasF(), 0, pl.getBiasSize());
    }
  }

  public static void saveParam(ArrayList<float[]> param, Path file) throws IOException
  {
    int numFloat=param.stream().mapToInt(array->array.length).sum();
    ByteBuffer buf=ByteBuffer.allocate((param.size()+1)*Integer.BYTES+numFloat*Float.BYTES);
    buf.putInt(param.size());
    for(float[] p: param)
    {
      buf.putInt(p.length);
      for(float value: p) buf.putFloat(value);
    }
    Files.write(file, buf.array());
  }

  public static ArrayList<float[]> loadParam(Path file) throws IOException
  {
    ByteBuffer buf=ByteBuffer.wrap(Files.readAllBytes(file));
    int size=buf.getInt();
    ArrayList<float[]> param=new ArrayList<float[]>(size);
```

```java
    for(int i=0; i<size; ++i)
    {
      int len=buf.getInt();
      float[] array=new float[len];
      param.add(array);
      for(int a=0; a<len; ++a) array[a]=buf.getFloat();
    }
    return param;
  }

  public static void saveOutput(HashMap<Sequence, float[]> output, Path file) throws IOException
  {
    int byteSize=Integer.BYTES;
    for(Sequence seq: output.keySet())
    {
      byteSize+=seq.byteSize();
      byteSize+=Integer.BYTES+output.get(seq).length*Float.BYTES;
    }
    ByteBuffer buf=ByteBuffer.allocate(byteSize);
    buf.putInt(output.size());
    for(Sequence seq: output.keySet())
    {
      seq.serialize(buf);
      buf.putInt(output.get(seq).length);
      for(float value: output.get(seq)) buf.putFloat(value);
    }
    Files.write(file, buf.array());
  }

  public static HashMap<Sequence, float[]> loadOutput(Path file, Collection<Sequence> sequence) throws IOException
  {
    HashMap<Sequence, Sequence> sequenceMap=new HashMap<>(sequence.size()*4/3);
    for(Sequence s: sequence) sequenceMap.put(s, s);

    ByteBuffer buf=ByteBuffer.wrap(Files.readAllBytes(file));
    int size=buf.getInt();
    HashMap<Sequence, float[]> output=new HashMap<Sequence, float[]>(size*4/3);
    for(int i=0; i<size; ++i)
    {
      Sequence seq=Sequence.deserialize(buf, sequenceMap);
      int len=buf.getInt();
      float[] value=new float[len];
      for(int v=0; v<len; ++v) value[v]=buf.getFloat();
      output.put(seq, value);
    }
    return output;
  }
}
```

Executor.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package utils;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

/**
 * {@link ExecutorService} and the number of thread are handled in the same class for convenience.
 * @author koumura
 *
 */
public class Executor
{
  private ExecutorService executor;
  private int numThread;

  public Executor(int numThread)
  {
    this.numThread = numThread;
    this.executor = Executors.newFixedThreadPool(numThread);
  }

  public ExecutorService get() {
    return executor;
  }

  public int getNumThread() {
    return numThread;
  }
}
```

MathUtils.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package utils;

import no.uib.cipr.matrix.NotConvergedException;
import no.uib.cipr.matrix.SymmPackEVD;
import no.uib.cipr.matrix.UpperSymmPackMatrix;

public class MathUtils
{
  public static int ceil(int numerator, int denominator)
  {
    if(numerator%denominator==0) return numerator/denominator;
    return numerator/denominator+1;
  }
  public static long ceil(long numerator, int denominator)
  {
    if(numerator%denominator==0) return numerator/denominator;
    return numerator/denominator+1;
  }

  private static double square(double x){return x*x;}

  /**
   * Computes 0th order discrete prolate spheroidal sequences.
   * @param fftLength
   * @param nw Parameter for the DPSS.
```

```
   * @return
   * @throws NotConvergedException
   */
  public static double[] dpss(int fftLength, int nw) throws NotConvergedException
  {
    double w=(double)nw/fftLength;
    UpperSymmPackMatrix matrix=new UpperSymmPackMatrix(fftLength);
    for(int i=0; i<fftLength; ++i)
    {
      double value=square((double)(fftLength-1-2*i)/2)*Math.cos(2*Math.PI*w);
      matrix.set(i, i, value);
    }
    for(int i=1; i<fftLength; ++i)
    {
      double value=(double)i*(fftLength-i)/2;
      matrix.set(i, i-1, value);
      matrix.set(i-1, i, value);
    }

    int dimension=matrix.numRows();
    SymmPackEVD evd=SymmPackEVD.factorize(matrix);
    int rowIndex=0;
    for(int i=0; i<dimension; ++i) if(evd.getEigenvalues()[i]>evd.getEigenvalues()[rowIndex]) rowIndex=i;
    double[] eigenVector=new double[fftLength];
    for(int j=0; j<dimension; ++j) eigenVector[j]=evd.getEigenvectors().get(j, rowIndex);
    return eigenVector;
  }

  public static int subIntervalBegin(int size, int numSub, int subIndex)
  {
    return subIndex*size/numSub;
  }
  public static long subIntervalBegin(long size, int numSub, int subIndex)
  {
    return subIndex*size/numSub;
  }

  public static double sum(double... value)
  {
    double sum=0;
    for(double v: value) sum+=v;
    return sum;
  }
  public static int sum(int... value)
  {
    int sum=0;
    for(int v: value) sum+=v;
    return sum;
  }

  public static void divideBySum(double[] value)
  {
    double sum=sum(value);
    for(int i=0; i<value.length; ++i) value[i]/=sum;
  }
  public static double[] divideBySum(int[] value)
  {
    int sum=sum(value);
    double[] freq=new double[value.length];
    for(int i=0; i<value.length; ++i) freq[i]=(double)value[i]/sum;
    return freq;
  }
}
```

Pair.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package utils;

import java.util.Comparator;

public class Pair<T0, T1>
{
  private T0 value0;
  private T1 value1;

  public Pair(T0 value0, T1 value1) {
    this.value0 = value0;
    this.value1 = value1;
  }

  public T0 get0() {
    return value0;
  }

  public void set0(T0 value0) {
    this.value0 = value0;
  }

  public T1 get1() {
    return value1;
  }

  public void set1(T1 value1) {
    this.value1 = value1;
  }

  public static <T0 extends Comparable<T0>, T1> Comparator<Pair<T0, T1>> comparator0()
  {
    return (o1, o2)->o1.get0().compareTo(o2.get0());
  }

  public static <T0, T1 extends Comparable<T1>> Comparator<Pair<T0, T1>> comparator1()
  {
    return (o1, o2)->o1.get1().compareTo(o2.get1());
  }
}
```

RandomUtils.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
```

```
 */
package utils;

import java.util.ArrayList;

import org.apache.commons.math3.random.MersenneTwister;

public class RandomUtils
{
  public static int[] permutation(int size, MersenneTwister random)
  {
    int[] index=new int[size];
    ArrayList<Integer> remaining=new ArrayList<>();
    for(int i=0; i<size; ++i) remaining.add(i);
    for(int i=0; i<size; ++i)
    {
      int next=random.nextInt(remaining.size());
      index[i]=remaining.get(next);
      remaining.remove(next);
    }
    return index;
  }
  public static <T> ArrayList<T> permutation(ArrayList<T> list, MersenneTwister random)
  {
    int[] index=permutation(list.size(), random);
    ArrayList<T> permutation=new ArrayList<T>(list.size());
    for(int i: index) permutation.add(list.get(i));
    return permutation;
  }
}
```

SoundUtils.java

```
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package utils;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;

import javax.sound.sampled.AudioFormat;
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.UnsupportedAudioFileException;

import org.apache.commons.math3.complex.Complex;
import org.apache.commons.math3.stat.descriptive.SummaryStatistics;
import org.apache.commons.math3.transform.DftNormalization;
import org.apache.commons.math3.transform.FastFourierTransformer;
import org.apache.commons.math3.transform.TransformType;

import computation.STFTParam;
import computation.Sequence;
import computation.Sequence.WavePosition;
import no.uib.cipr.matrix.NotConvergedException;

public class SoundUtils
{
  /**
   * Reads wave data from 16 bit linear PCM.
   * @param file
   * @return
   * @throws UnsupportedAudioFileException
   * @throws IOException
   */
  public static short[] readWaveShort(Path file) throws UnsupportedAudioFileException, IOException
  {
    AudioInputStream stream=AudioSystem.getAudioInputStream(file.toFile());
    AudioFormat format=stream.getFormat();
    int numChannel=format.getChannels();
    int bitDepth=format.getSampleSizeInBits();
    if(numChannel!=1) throw new UnsupportedAudioFileException("Number of channel must be 1.");
    if(bitDepth!=16) throw new UnsupportedAudioFileException("Bit depth must be 16.");
    long frameLength=stream.getFrameLength();
    if(frameLength>Integer.MAX_VALUE) throw new UnsupportedAudioFileException("Too long audio.");
    int frameLengthInt=(int)frameLength;
    float samplingRate=format.getSampleRate();
    byte[] data=new byte[frameLengthInt*bitDepth/8];
    int readLen=0;
    int len;
    while((len=stream.read(data, readLen, data.length-readLen))>=0)
    {
      readLen+=len;
    }
    stream.close();
    ByteBuffer buf=ByteBuffer.wrap(data);
    if(!format.isBigEndian()) buf=buf.order(ByteOrder.LITTLE_ENDIAN);
    short[] dataShort=new short[data.length/2];
    for(int i=0; i<dataShort.length; ++i) dataShort[i]=buf.getShort();
    return dataShort;
  }

  public static AudioFormat readAudioFormat(Path file) throws UnsupportedAudioFileException, IOException
  {
    AudioInputStream stream=AudioSystem.getAudioInputStream(file.toFile());
    AudioFormat format=stream.getFormat();
    stream.close();
    return format;
  }

 public static float[] spectrogram(short[] wave, STFTParam stftParam, double[] taper, int freqOffset, int freqLength,
int wavePosition, int waveLength, double[] realData)
  {
    if(realData==null||realData.length!=stftParam.getFftLength()) realData=new
double[stftParam.getFftLength()];
    int length=stftParam.spectrogramLength(waveLength);
    float[] spec=new float[length*freqLength];
    FastFourierTransformer fft=new FastFourierTransformer(DftNormalization.STANDARD);
    for(int i=0; i<length; ++i)
    {
      if(i<length-1)
      {
        for(int t=0; t<stftParam.getFftLength(); ++t)
realData[t]=wave[wavePosition+i*stftParam.getShiftLength()+t]*taper[t];
      }
      else
      {
        int len=waveLength-(length-1)*stftParam.getShiftLength();
        for(int t=0; t<len; ++t) realData[t]=wave[wavePosition+i*stftParam.getShiftLength()+t]*taper[t];
        for(int t=len; t<stftParam.getFftLength(); ++t) realData[t]=0;
```

```java
      }
      Complex[] complex=fft.transform(realData, TransformType.FORWARD);
      for(int f=0; f<freqLength; ++f) spec[i*freqLength+f]=(float)Math.log10(complex[f+freqOffset].abs());
    }
    return spec;
  }

  public static <K> HashMap<K, float[]> spectrogram(HashMap<K, WavePosition> waveFilePositionLength, STFTParam
stftParam, int dpssParam, int freqOffset, int freqLength) throws UnsupportedAudioFileException, IOException,
NotConvergedException
  {
    HashMap<Path, LinkedList<K>> fileMap=new HashMap<>();
    for(K key: waveFilePositionLength.keySet())
    {
      Path path=waveFilePositionLength.get(key).getWaveFile();
      fileMap.putIfAbsent(path, new LinkedList<>());
      fileMap.get(path).add(key);
    }
    HashMap<K, float[]> spectrogram=new HashMap<>(waveFilePositionLength.size()*4/3);
    double[] window=MathUtils.dpss(stftParam.getFftLength(), dpssParam);
    double[] realData=new double[stftParam.getFftLength()];
    for(Path file: fileMap.keySet())
    {
      short[] wave=readWaveShort(file);
      for(K seq: fileMap.get(file))
      {
        int pos=waveFilePositionLength.get(seq).getPosition();
        int len=waveFilePositionLength.get(seq).getLength();
        float[] spec=spectrogram(wave, stftParam, window, freqOffset, freqLength, pos, len, realData);
        spectrogram.put(seq, spec);
      }
    }
    return spectrogram;
  }

  public static float[] spectrogram(Sequence sequence, Path dirWave, STFTParam stftParam, int dpssParam, int
frequencyOffset, int frequencyLength) throws UnsupportedAudioFileException, IOException, NotConvergedException
  {
    ArrayList<Sequence> sequenceList=new ArrayList<>(1);
    HashMap<Sequence, WavePosition> waveFilePosition=Sequence.wavePositionMap(sequenceList, dirWave);
    return spectrogram(waveFilePosition, stftParam, dpssParam, frequencyOffset, frequencyLength).get(sequence);
  }

  public static void whiteSpectrogram(Collection<float[]> spectrogram, double mean, double sd) throws IOException
  {
    for(float[] spec: spectrogram) for(int i=0; i<spec.length; ++i) spec[i]=(float)((spec[i]-mean)/sd);
  }

  public static double[] spectrogramMeanSd(List<float[]> spectrogram)
  {
    SummaryStatistics stat=new SummaryStatistics();
    for(float[] spec: spectrogram) for(float v: spec) stat.addValue(v);
    return new double[]{stat.getMean(), stat.getStandardDeviation()};
  }

  /**
   * Sampling rate must be same in the all wave files. Perhaps this constraint can be loosened by modifying other
codes.
   * @param filename
   * @param dirWaveFile
   * @return
   * @throws UnsupportedAudioFileException
   * @throws IOException
   */
  public static float checkSamplingRate(Collection<String> filename, Path dirWaveFile) throws
UnsupportedAudioFileException, IOException
  {
    float samplingRate=Float.NaN;
    for(String fn: filename)
    {
      Path file=dirWaveFile.resolve(fn);
      AudioFormat format=readAudioFormat(file);
      if(Float.isNaN(samplingRate)) samplingRate=format.getSampleRate();
      if(samplingRate!=format.getSampleRate())
      {
        System.err.println("Sampling rate must be same across sequences.");
        return Float.NaN;
      }
    }
    return samplingRate;
  }

  public static BufferedImage spectrogramImage(float[] spectrogram, float blackValue, float whiteValue, int
frequencyLength, int bottomMargin)
  {
    BufferedImage image=new BufferedImage(spectrogram.length/frequencyLength, frequencyLength+bottomMargin,
BufferedImage.TYPE_INT_ARGB);
    Graphics2D graphics=(Graphics2D)image.getGraphics();
    for(int i=0; i<spectrogram.length; ++i)
    {
      double rgbd=(spectrogram[i]-blackValue)/(whiteValue-blackValue)*255;
      int rgbi;
      if(rgbd>255) rgbi=255;
      else if(rgbd<0) rgbi=0;
      else rgbi=(int)rgbd;
      Color color=new Color(rgbi, rgbi, rgbi);
      graphics.setPaint(color);
      int y=frequencyLength-1-i%frequencyLength;
      int x=i/frequencyLength;
      graphics.fillRect(x, y, 1, 1);
    }
    return image;
  }
}
```

XmlUtils.java

```java
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */
package utils;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.LinkedList;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.bootstrap.DOMImplementationRegistry;
import org.w3c.dom.ls.DOMImplementationLS;
import org.w3c.dom.ls.LSInput;
```

```java
import org.w3c.dom.ls.LSOutput;
import org.w3c.dom.ls.LSParser;
import org.w3c.dom.ls.LSSerializer;

public class XmlUtils
{
  private static final DocumentBuilder DOCUMENT_BUILDER;
  private static final DOMImplementationRegistry REGISTRY;

  static
  {
    DocumentBuilder documentBuilder=null;
    try
    {
      documentBuilder=DocumentBuilderFactory.newInstance().newDocumentBuilder();
    }
    catch (ParserConfigurationException e) {e.printStackTrace();}
    DOCUMENT_BUILDER=documentBuilder;

    DOMImplementationRegistry registry = null;
    try
    {
      registry = DOMImplementationRegistry.newInstance();
    }
    catch (ClassNotFoundException | InstantiationException | IllegalAccessException | ClassCastException e)
{e.printStackTrace();}
    REGISTRY=registry;
  }

  public static Element rootElement(String nodeName)
  {
    Document document = DOCUMENT_BUILDER.newDocument();
    Element root= document.createElement(nodeName);
    document.appendChild(root);
    return root;
  }

  public static void write(Document document, OutputStream stream)
  {
    DOMImplementationLS impl = (DOMImplementationLS)REGISTRY.getDOMImplementation("LS");
    LSSerializer writer = impl.createLSSerializer();
    LSOutput output=impl.createLSOutput();
    output.setByteStream(stream);
    writer.write(document, output);
  }
  public static void write(Document document, Path file) throws IOException
  {
    BufferedOutputStream stream=new BufferedOutputStream(Files.newOutputStream(file));
    write(document, stream);
    stream.close();
  }

  public static void removeBlankText(Node node, LinkedList<Pair<Node, Node>> removed)
  {
    for(int c=0; c<node.getChildNodes().getLength(); ++c)
    {
      Node child=node.getChildNodes().item(c);
      if(child.getNodeType()==Node.TEXT_NODE&&child.getTextContent().trim().length()==0) removed.add(new
Pair<>(node, child));
      else removeBlankText(child, removed);
    }
  }

  public static Document parse(InputStream stream)
  {
    DOMImplementationLS impl = (DOMImplementationLS)REGISTRY.getDOMImplementation("LS");
    LSParser builder = impl.createLSParser(DOMImplementationLS.MODE_SYNCHRONOUS, null);
    LSInput input=impl.createLSInput();
    input.setByteStream(stream);
    Document document = builder.parse(input);

    LinkedList<Pair<Node, Node>> removed=new LinkedList<>();
    removeBlankText(document, removed);
    for(Pair<Node, Node> r: removed) r.get0().removeChild(r.get1());
```



```java
    return document;
  }
  public static Document parse(Path file) throws IOException
  {
    BufferedInputStream stream=new BufferedInputStream(Files.newInputStream(file));
    Document document=parse(stream);
    stream.close();
    return document;
  }

  public static <T> Element addChild(Node parent, String name, T content)
  {
    Element child=parent.getOwnerDocument().createElement(name);
    if(content!=null) child.setTextContent(content.toString());
    parent.appendChild(child);
    return child;
  }

  public static boolean nodeBoolean(Node node){return
Boolean.parseBoolean(node/*.getFirstChild()*/.getTextContent());}
  public static byte nodeByte(Node node){return Byte.parseByte(node/*.getFirstChild()*/.getTextContent());}
  public static int nodeInt(Node node){return Integer.parseInt(node/*.getFirstChild()*/.getTextContent());}
  public static short nodeShort(Node node){return Short.parseShort(node/*.getFirstChild()*/.getTextContent());}
  public static float nodeFloat(Node node){return Float.parseFloat(node/*.getFirstChild()*/.getTextContent());}
  public static double nodeDouble(Node node){return
Double.parseDouble(node/*.getFirstChild()*/.getTextContent());}
  public static String nodeText(Node node){return node/*.getFirstChild()*/.getTextContent();}
}
```

Kernel.cu

```cuda
/*
 * Copyright (C) 2016 Takuya KOUMURA
 * https://github.com/takuya-koumura/birdsong-recognition
 *
 * This file is part of Birdsong Recognition.
 *
 * Birdsong Recognition is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Birdsong Recognition is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Birdsong Recognition.  If not, see <http://www.gnu.org/licenses/>.
 */

#include <device_launch_parameters.h>
#include <cmath>

extern "C" __global__ void SeqSoftmaxConvBackwardDoubleChar(const char *label, double *diff, const int *size, int
labelShiftY, int labelShiftX)
{
  int batchSize = size[0];
  int numChannel= size[1];
  int height = size[2];
  int width = size[3];
  int singleSize= size[4];
  int singleHeight= size[5];
  int singleWidth= size[6];
  int labelHeight = size[7];
  int labelWidth = size[8];
  int labelShiftUpperH = size[9];
  int labelShiftUpperW = size[10];

  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx >= batchSize*height*width*singleHeight*singleWidth) return;
```

```
  int x = idx%width;
  int i = (idx - x) / width;
  int y = i%height;
  i = (idx - y) / height;
  int sx = i%singleWidth;
  i = (i - sx) / singleWidth;
  int sy = i%singleHeight;
  int b = i / singleHeight;

  char la = label[b*labelWidth*labelHeight + (y*labelShiftUpperH + labelShiftY + sy)*labelWidth + x*labelShiftUpperW
+ labelShiftX + sx];

  if (la < 0)
  {
    for (int li = 0; li < singleSize; ++li)
    {
      int index = ((b*numChannel + (li*singleHeight + sy)*singleWidth + sx)*height + y)*width + x;
      diff[index] = 0;
    }
    return;
  }
  int index = ((b*numChannel + (la*singleHeight + sy)*singleWidth + sx)*height + y)*width + x;
  diff[index] -= 1;
}

extern "C" __global__ void SeqSoftmaxConvBackwardFloatChar(const char *label, float *diff, const int *size, int
labelShiftY, int labelShiftX)
{
  int batchSize = size[0];
  int numChannel = size[1];
  int height = size[2];
  int width = size[3];
  int singleSize = size[4];
  int singleHeight = size[5];
  int singleWidth = size[6];
  int labelHeight = size[7];
  int labelWidth = size[8];
  int labelShiftUpperH = size[9];
  int labelShiftUpperW = size[10];

  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx >= batchSize*height*width*singleHeight*singleWidth) return;

  int x = idx%width;
  int i = (idx - x) / width;
  int y = i%height;
  i = (idx - y) / height;
  int sx = i%singleWidth;
  i = (i - sx) / singleWidth;
  int sy = i%singleHeight;
  int b = i / singleHeight;

  char la = label[b*labelWidth*labelHeight + (y*labelShiftUpperH + labelShiftY + sy)*labelWidth + x*labelShiftUpperW
+ labelShiftX + sx];

  if (la < 0)
  {
    for (int li = 0; li < singleSize; ++li)
    {
      int index = ((b*numChannel + (li*singleHeight + sy)*singleWidth + sx)*height + y)*width + x;
      diff[index] = 0;
    }
    return;
  }
  int index = ((b*numChannel + (la*singleHeight + sy)*singleWidth + sx)*height + y)*width + x;
  diff[index] -= 1;
}


extern "C" __global__ void SeqSoftmaxConvErrorDoubleChar(double *error, const double *output, const char *label,
const int *size, int labelShiftY, int labelShiftX, double outputLowerForError)
{
  int batchSize = size[0];
  int numChannel = size[1];
  int height = size[2];
  int width = size[3];
  int singleSize = size[4];
  int singleHeight = size[5];
  int singleWidth = size[6];
  int labelHeight = size[7];
  int labelWidth = size[8];
  int labelShiftUpperH = size[9];
  int labelShiftUpperW = size[10];

  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx >= batchSize*height*width*singleHeight*singleWidth) return;

  int x = idx%width;
  int i = (idx - x) / width;
  int y = i%height;
  i = (idx - y) / height;
  int sx = i%singleWidth;
  i = (i - sx) / singleWidth;
  int sy = i%singleHeight;
  int b = i / singleHeight;

  int labelIndex = b*labelWidth*labelHeight + (y*labelShiftUpperH + labelShiftY + sy)*labelWidth +
x*labelShiftUpperW + labelShiftX + sx;
  char la = label[labelIndex];

  if (la < 0)
  {
    error[idx] = 0;
    return;
  }

  int index = ((b*numChannel + (la*singleHeight + sy)*singleWidth + sx)*height + y)*width + x;
  double o = output[index];
  if (isnan(o)) error[idx] = o;
  else
  {
    if (o < outputLowerForError) o = outputLowerForError;
    error[idx] = -log(o);
  }
}


extern "C" __global__ void SeqSoftmaxConvErrorFloatChar(float *error, const float *output, const char *label, const
int *size, int labelShiftY, int labelShiftX, float outputLowerForError)
{
  int batchSize = size[0];
  int numChannel = size[1];
  int height = size[2];
  int width = size[3];
  int singleSize = size[4];
  int singleHeight = size[5];
  int singleWidth = size[6];
  int labelHeight = size[7];
  int labelWidth = size[8];
  int labelShiftUpperH = size[9];
  int labelShiftUpperW = size[10];

  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx >= batchSize*height*width*singleHeight*singleWidth) return;

  int x = idx%width;
  int i = (idx - x) / width;
  int y = i%height;
  i = (idx - y) / height;
  int sx = i%singleWidth;
  i = (i - sx) / singleWidth;
  int sy = i%singleHeight;
  int b = i / singleHeight;

  int labelIndex = b*labelWidth*labelHeight + (y*labelShiftUpperH + labelShiftY + sy)*labelWidth +
x*labelShiftUpperW + labelShiftX + sx;
  char la = label[labelIndex];
```

```
  if (la < 0)
  {
    error[idx] = 0;
    return;
  }

  int index = ((b*numChannel + (la*singleHeight + sy)*singleWidth + sx)*height + y)*width + x;
  float o = output[index];
  if (isnan(o)) error[idx] = o;
  else
  {
    if (o < outputLowerForError) o = outputLowerForError;
    error[idx] = -log(o);
  }
}


extern "C" __global__ void FillFloat(float *vector, float value, int size)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx >= size) return;

  vector[idx] = value;
}

extern "C" __global__ void FillDouble(double *vector, double value, int size)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx >= size) return;

  vector[idx] = value;
}


extern "C" __global__ void AdamFloat(float *param, const float* grad, float* moment, float* moment2, const float*
hyperParam, int size)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx >= size) return;

  float alpha = hyperParam[0];
  float beta1= hyperParam[1];
  float beta2 = hyperParam[2];
  float epsilon = hyperParam[3];
  float beta1T = hyperParam[4];
  float beta2T = hyperParam[5];

  moment[idx] = beta1*moment[idx] + (1 - beta1)*grad[idx];
  moment2[idx] = beta2*moment2[idx] + (1 - beta2)*grad[idx] * grad[idx];
  float alphaT = alpha*sqrt(1 - beta2T) / (1 - beta1T);
  float delta = alphaT*moment[idx] / (sqrt(moment2[idx]) + epsilon);
  param[idx] -= delta;
}

extern "C" __global__ void AdamDouble(double *param, const double* grad, double* moment, double* moment2, const
double* hyperParam, int size)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx >= size) return;

  double alpha = hyperParam[0];
  double beta1 = hyperParam[1];
  double beta2 = hyperParam[2];
  double epsilon = hyperParam[3];
  double beta1T = hyperParam[4];
  double beta2T = hyperParam[5];

  moment[idx] = beta1*moment[idx] + (1 - beta1)*grad[idx];
  moment2[idx] = beta2*moment2[idx] + (1 - beta2)*grad[idx] * grad[idx];
  double alphaT = alpha*sqrt(1 - beta2T) / (1 - beta1T);
  double delta = alphaT*moment[idx] / (sqrt(moment2[idx]) + epsilon);
  param[idx] -= delta;
}
```

付録 E　確率接尾辞木の構成手順

アルゴリズム：確率接尾辞木の構成。

ハイパーパラメータ：節の候補として採用する部分系列の最小出現確率P_{min}、ある部分系列をその最大長接尾辞の下位の節として採用するための、その部分系列が与えられたときの条件付き確率の最小値C_{min}、その際の条件付き確率の変化の割合の最小値r、節として採用する部分系列の最大長L、条件付き確率の平滑化定数γ。

(1) Tを長さ 0 の記号列を付与した根のみからなる木とする。Sを$\{\sigma | \sigma \in \Sigma$ かつ $P(\sigma) \geq P_{min}\}$とする。

ただし要素分類をσ、全要素分類の集合をΣ、要素分類σの出現頻度を$P(\sigma)$とする。

(2) Sが空でない限り、Sから$s \in S$を選び、次を繰り返す。

(a) Sからsを除く。

(b) $P(\sigma|s) \geq C_{min}$かつ$\frac{P(\sigma|s)}{P(\sigma|suf(s))}\begin{cases} \geq r \\ \text{または} \\ \leq 1/r \end{cases}$ を満たす$\sigma \in \Sigma$が存在すれば、sを付与した節と、sの全ての接尾辞を付与した節を、Tに追加する。

ただし、部分系列sの最大長接尾辞を$suf(s)$とする。例えば、s =ABC のとき、$suf(s)$ =BC となる。

(c) $|s| < L$なら、$\{\sigma' s | \sigma' \in \Sigma$ かつ $P(\sigma' s) \geq P_{min}\}$を$S$に追加する。

(3) Tの全ての節に、平滑化した条件付き確率

$$\frac{P(\sigma|s) + \gamma}{\sum_{\sigma'}(P(\sigma'|s) + \gamma)}$$

を付与する。