修 士 論 文

Self and Mutual Occlusion Aware Multi-Person Skeleton Estimation using Multi-Task Deep Learning Network

（自己および相互オクルージョンを考慮したマルチタスク深層学習による人物スケルトン推定）

指導教員　　上條俊介　准教授

東京大学大学院
情報理工学系研究科
電子情報学専攻

学籍番号・氏名 48-176441　張　慧揚

提出日　２０１９年１月31日

Abstract

For years, behavior understanding has been a hot topic in the field of computer vision. As an important part of human behavior understanding, skeleton estimation has attracted lots of interests. Recently, deep learning methods, such as Mask R-CNN, have achieved much better performance for computer vision tasks than that of traditional approaches, as deep neural network can find representative features efficiently. For skeleton estimation, most of the deep learning approaches mainly focus on the joint feature. However, this feature is not sufficient, especially when the pose is occluded or not intact. In fact, many features other than joint can also contribute to skeleton estimation, such as body boundary, body orientation and visibility condition. By adopting multi-task strategy, these features can be efficiently combined inside deep learning model for skeleton estimation.

In this thesis, we present a multi-task skeleton estimation approach to deal with human behavior understanding combining different human information as new features. Our deep learning model is based on Mask R-CNN and extended to a multi-task network with three branches including five tasks. We first build our parallel proposed multi-task network with each task separately. Then we improve our proposed method into two serial multi-task models by connecting different branch in the training. Our proposed model is trained on the public dataset MS COCO, which is further augmented by ground truths of orientation mask and mutual-occlusion mask. Experiments show the learning accuracy of the proposed method. Comparisons further illustrate the performance improvement after combining more features by multi-task strategy.

Contents

List of Figures

List of Tables

Chapter 1.

Introduction

In this thesis, we studied how to build a self and mutual occlusion aware multi-person skeleton estimation model using multi-task deep learning network

1.1. Background

As the needs of big data analysis increases, behavior analysis and scene understanding have been highly-concerned topics in computer vision field, which can provide valuable information for other purpose such as marketing or security. For example, customer behavior information could be useful to improve customer experience, reduce daily operational costs, optimize shopping performance, and finally increase profit of store. In the related research of these topics, human behavior analysis has an irreplaceable position. In order to achieve this goal, information hidden in human behavior need to be extracted, especially the pose (the skeleton structure) and orientation information. To extract these information, we need to estimate human pose and orientation using image processing tasks.

In order to estimate the human pose, there are many approaches using different kinds of information. One of the approaches is to apply human silhouette to estimate the pose or action. From silhouette, model-based methods are commonly employed for pose estimation [1]. There are also methods which analyze silhouettes from multiple views to generate 3D human pose [2].

Another way for pose estimation is to cooperate the temporal information in image sequence. One kind of these methods is based on detection and tracking, which detects a rough pose in the initial frame and track it in every continuous frame [3]. Another way is to apply spatio-temporal Markov Random Field (MRF) models for pose estimation in image sequence [4]. These models are benefited from the position constraints of both intra-frame joints and inter-frame joints. Based on the spatio-temporal MRF, Cherian et al. [5] split the pose into limbs and then recomposed them to pose after optimization.

A more basic but more challenging topic is human pose estimation from a single image. It requires to detect human from variant backgrounds and estimate the pose in a large number of degrees of freedom. General techniques are using Pictoral Structures

[6] or Deformable Part Models (DPM) [7]. As another approach, Pishchulin et al. [8] proposed a more flexible Poselet model which is based on Pictorial Structure. In recent years, the Deep Neural Networks (DNN) [9] achieve high performance in different tasks of computer vision. New network structures like Alexnet [10], VGG [11], GoogLeNet [12] and ResNet [13] has achieved higher accuracy in image classification task. In pose estimation task, Toshev et al. [14] proposed the DeepPose to cascade DNN structure to estimate human pose from coarse to fine. However, there are some limitations of their method. Firstly, if the initial estimation is result is far from the true position, the system will not be able to correct the estimation in the cascade structure. Secondly, this method only gives one prediction per image, which means there is no candidate to improve the prediction. To address the first problem, Carreira et al. [15] and Haque et al. [16] applied feedback structures to optimize the pose estimation in 2D and depth image respectively. Their methods feed the estimated pose to the input end and gradually refine the result in iteration. And for the second problem, methods using probabilistic heatmaps were proposed as a solution. These heatmaps generated by Convolutional Neural Networks (CNN) turn the joint estimation problem in to a pixel-wise classification problem.

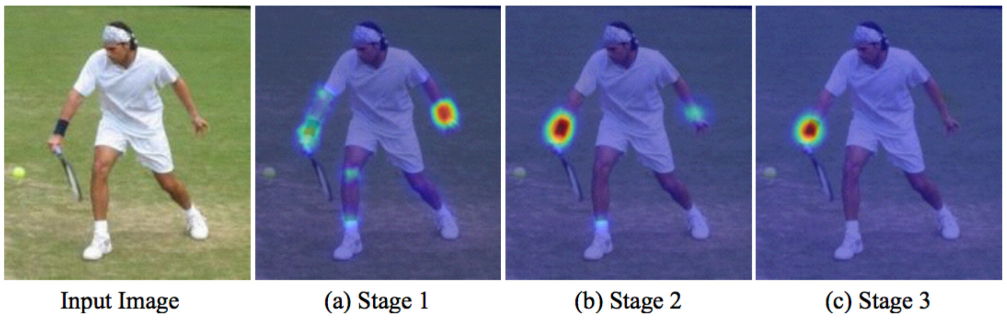


Figure 1.1 An example of input and output from multi-stage CNN pose estimation method [17]

As the CNN structure is widely used in image processing tasks, more and more researchers noticed that the dense feature maps extracted by CNN can be very helpful to describe human pose information in pose estimation task. Wei et al. [17] build a Convolutional Pose Machine network for single person pose estimation. Each joint from the person of input image will be predicted at each stage and be printed out as confidence maps (heat maps) and intermediate supervision is introduced to solve the vanishing gradient problem. Figure 1.1 shows how multi-stage CNN corrects the output of joint prediction. Newell et al. [18] presents an hourglass module to capture information at every scale in order to combine features from different stage better.

However, when it comes to multi-person pose estimation, the problem becomes more complex. Occlusion caused by overlapping between two people makes the detection much more difficult. In order to deal with multi-person pose estimation, the solutions are divided into two types: Top-down approaches and Bottom-up approaches.

In bottom-up approaches, firstly body part detection would be done, and then by person clustering and joint labeling we could make an accurate prediction of the number of people and their poses in the input image. Pishchulin et al. [19] propose a DeepCut method where Fast R-CNN[20] is used as body part detector. Then the goal of task becomes subset partitioning from a graph of all connections among each joint detected from the image, which turns the task into an Integer Linear Programming (ILP) problem (illustrated in Figure 1.2). They also developed this method into DeeperCut[21] which improves DeepCut by using deeper ResNet architectures[13] to enhance body part detectors and propose an image-conditioned pairwise terms that assemble the proposals into a variable number of consistent body part configurations. Furthermore, in order to achieve the goal of real-time pose estimation, Cao et al. present a new method using Part Affinity Fields (PAFs) [22], which is also known as OpenPose later. Part Affinity Fields is a set of 2D vector fields that represents both locations and orientations of limbs over the image domain. They build a deep CNN architecture which is similar to their former contribution [17], but in this paper they use two parallel branches in order to generate both confidence maps and PAFs at the same stage, which is shown in Figure 1.3.

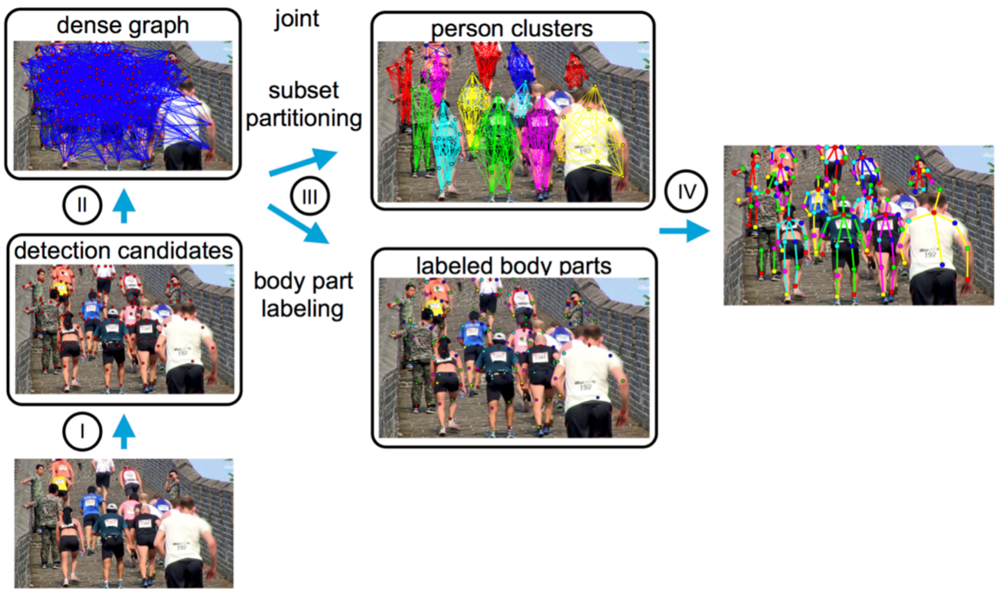


Figure 1.2 Visualization of Deep Cut [19]

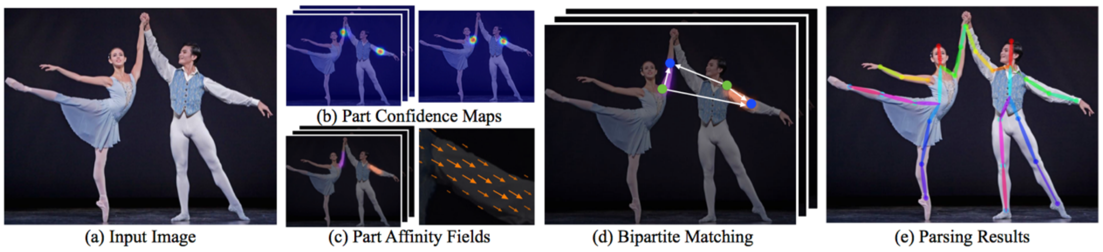


Figure 1.3 Pipeline of OpenPose [11]

Although bottom-up approaches show general advantage in runtime analysis, it's still a tough task when people in the image are in small scale where detecting body parts before detecting person itself becomes even difficult. At this case, top-down approaches could have better results by first do the person detection and then proceed single-person pose estimation in each bounding box. Papandreou et al. [23] propose a method using a two-stage network which first uses a person box detection system [24] as bounding box detector, then predict the pose of each single person in each bounding box. He et al. [25] also develop a multi-task method based on their former contribution [26] which can be implemented for pose estimation task.

Furthermore, in order to handle the self-occlusion in pose estimation, Azizpour et al. [27] and Ghiasi et al. [28] learned templates for occluded versions of each body part. Rafi et al. [29] incorporated the context information of occluding objects to predict the locations of occluded joints. Haque et al. [30] implicitly learned the occlusion through a deep neural network and gave the output of visibility mask of joints.

On the other hand, the orientation estimation is studied in many different scenes. Gandhi et al. [31] used Histogram of Oriented Gradients (HOG) [32] and Support Vector Machine (SVM) to recognize the body orientations of pedestrians. Goffredo et al. [33] proposed to recognize the gaits with different orientation by using in multi-camera surveillance.

1.2. Objective

In our previous work [34], we build a system for single person customer pose estimation combined with orientation prediction. However, in habitual shopping scenes multi-person case is exceedingly common, where the occlusion between different customers become more intractable for the pose estimation task. In order to deal with this problem and build a more general model for multi-person pose estimation, we propose a multi-task skeleton estimation system using Deep Neural Network, which combines body orientation prediction task with a top-down skeleton estimation network to keep both relative estimation high accuracy. In fact, there is much more useful information that can contribute to pose estimation task such as body boundary, body part segmentation, and visibility condition. By adopting multi-task strategy, these features can be efficiently combined inside deep learning model for pose estimation. Therefore, we add body segmentation and mutual-occlusion as new information into our model to enhance the results of both pose estimation and orientation prediction. For the purpose to train orientation and occlusion recognition tasks, we build a subset with images from COCO keypoints challenge dataset by adding orientation and occlusion mask as new annotations.

1.3. Thesis Structure

The Structure of this paper is listed as follows:

In Chapter 2, basic concepts of the Deep Neural Network (DNN) are introduced.

In Chapter 3, we will introduce several previous works about multi-task pose (skeleton) estimation.

In Chapter 4, we will introduce our proposed multi-task skeleton estimation model and show the experiment results.

In Chapter 5, we give the conclusions of this work and discuss the future work.

Chapter 2.

Basic Concepts of Deep Neural Network

In this chapter, we will explain some basic concepts about Deep Neural Network (DNN). DNN is a kind of multi-layer feed-forward network specialize for extracting features from input data. Section 2.1 first introduces some biological concepts of DNN. In Section 2.2, we introduce some basic layers used of DNN. Then a detailed introduction of non-linear activation functions will be given in Section 2.3. In Section 2.4, we introduce several most used and important architectures, which are LeNet[42], AlexNet [10], VGGNet [11], GoogleNet [12], and ResNet [13], in chronological order. Section 2.5 gives a general introduction about how to train the neural networks.

2.1. Introduction of Deep Neural Network

Inspired by biological nervous systems, DNN aim at reaching their versatility through learning. DNN is commonly employed in artificial intelligence, machine learning, and pattern recognition. There has been substantial research into how the human brain's structure achieves such a high level of versatility. This research has provided some important insights, however, the conclusions are far from completely explaining the complex functioning of the brain. Even though we have not been able to replicate the brain so far, the field of artificial intelligence offers very effective solutions to many problems by simulating the observations of biological research of various nervous systems.

It is estimated, that the average human brain contains 86 billion neurons [35]. Together they form a huge network. Even if we knew the detailed inner structure of the human brain, we would still not be able to simulate it with current technology because of its robustness. Our efforts are therefore rather different. We want to build a neural network with a good ratio between its size and its effectiveness.

Generally, DNN consists of a set of artificial neurons. Formally, an artificial neural has n inputs represented as a vector $\vec{x} \in \mathbb{R}^n$. Inputs in an artificial neuron correspond to the dendrites in a biological neuron, while a single output of an artificial neuron corresponds to the axon in a biological neuron, which is depicted in Figure 2.1. Each input i, $1 \leq i \leq n$, has an assigned weight w_1, w_2, ..., w_n, and bias b_1, b_2, ..., b_n.

Weighted input values are combined and followed by a non-linear activation (see in Section 2.3), as shown in Figure 2.2.

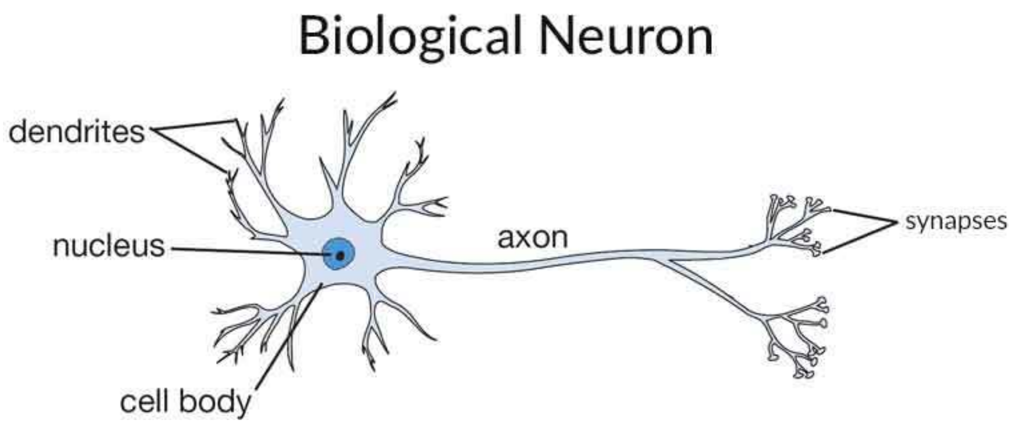


Figure 2.1 Model of biological neuron [36]

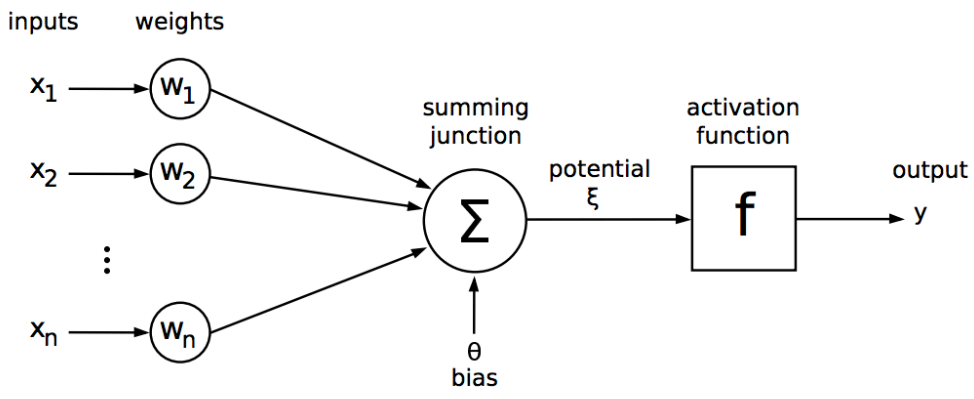


Figure 2.2 Model of an artificial neuron

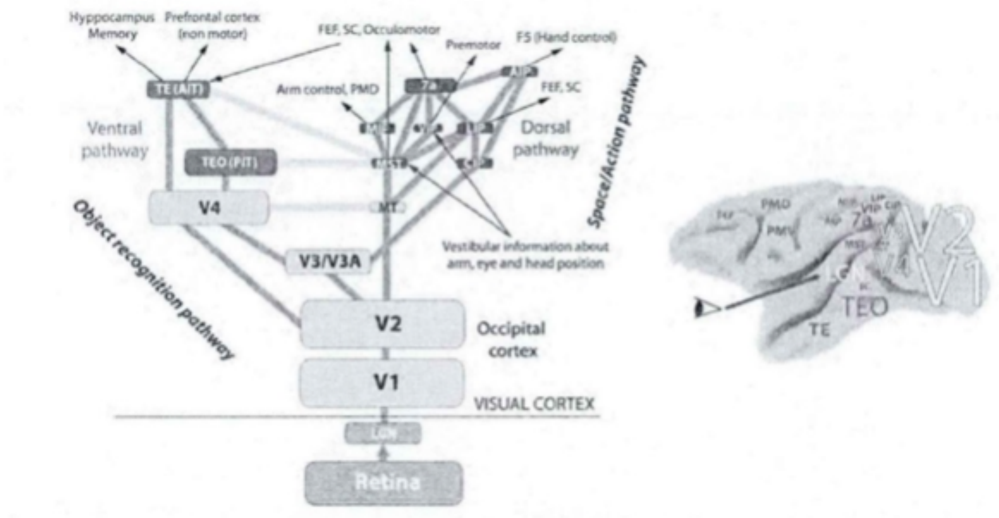


Figure 2.3 Brain structure explaining how human's visual system works [37]

DNN represents images using a multi-layered hierarchy of features and are inspired by the structure and functionality of the visual pathway of the human brain. Figure 2.3 [37] shows human's visual system. Visual information is collected by our eyes, and project to the retina. Then the information will be transmitted in a stream way (layers by layers. Actually, different layers in our brain servers as different functionality, in Brain Sciences, we divide it into V1, V2, V3, V4 regions. DNN also mimics the information flow in layers by layers way. It works by feeding the data into the input neurons. The data flow in the direction of oriented edges and ends when the output neurons are hit. The result is interpreted from the values obtained in the output neurons. For the input neurons, they together represent an instance of the problem to be solved by DNN. All output neurons have exactly one output, and all outputs together represent a possible solution to the problem. Between input neurons and output neurons, there exist a layers by layers structure. Figure 2.4 shows a Multilayer perceptron [38] network consists of three layers. A set of hidden neurons consists of the neurons which are not input, nor output neurons. Their number and organization into layers may vary even for the same problem but is a key feature of the network vastly influencing its performance. The Multilayer perceptron (MLP) is a feed-forward neural network consisting of multiple mutually interconnected layers of neurons. The layers are stacked one onto each other. Every neuron in one layer is connected to every neuron in the following layer. The motivation behind designing multilayer networks is to be able to solve more complex tasks. The layers in MLP network is built of fully-connected layers defined in Section 2.2.

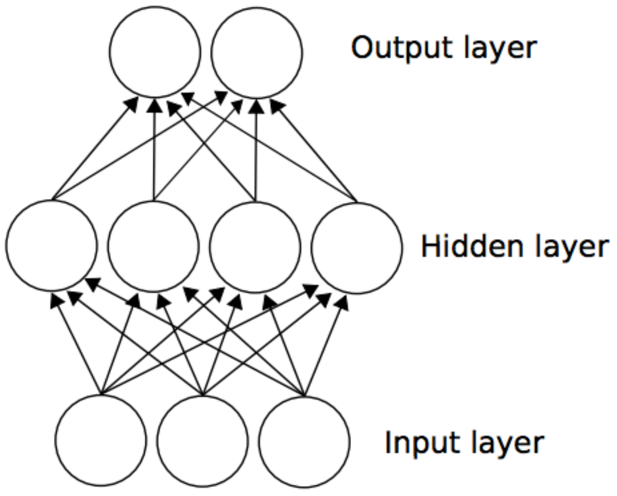


Figure 2.4 A multilayer perceptron network with three layers [10]

DNN uses the stacked layers structure to solve various problems in real world. For example, image classification, speech recognition, language transformation and so on. Based on my understanding, DNN can be thought as a huge black box for high-dimensional approximator. The implementation of DNNs can be easily described by linear transformation followed by non-linear activation. By stacking this procedure again and again, DNNs are competent for a variety of tasks.

2.2. Layers

Even though there are many different architectures for DNN in the literature, the majority of them can be built by stacking four main type of layers in different combinations. Namely, the fully connected layer, the convolutional layer, pooling layer and batch normalization layer. In this section, we will explain those layers.

2.2.1 Fully Connected Layer

Mathematically, we can treat linear layers as functions which applies linear transformation on vector inputs of dimension I and output vectors of dimension O. Usually the layer has a bias parameter, and it can be expressed like:

$$y = W \cdot x + b \tag{2.1}$$

Linear layer is motivated by the basic computational unit of the brain: neuron. Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately 10^{14}- 10^{15} synapses [35]. Each neuron receives input signals from its dendrites and produces output signal along its axon. The linear layer is a simplification of a group of neuron having their dendrites connected to the same inputs. Usually an activation function, such as sigmoid, is used to mimic the $1-0$ impulse carried away from the cell body and also to add non-linearity (shown in Figure 2.5).

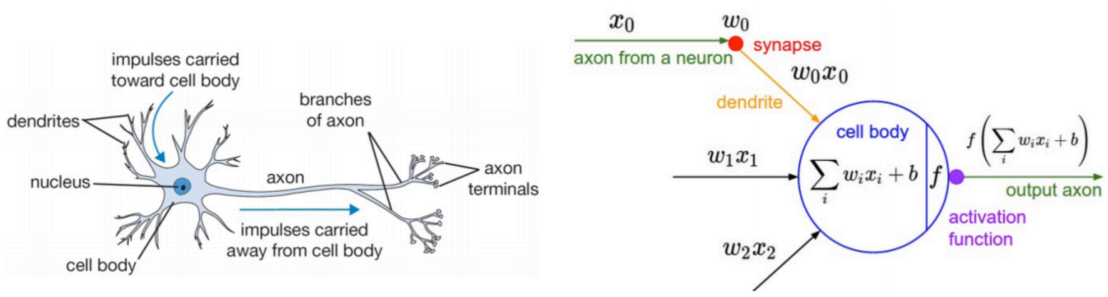


Figure 2.5 An illustration of biological neuron (left), and its mathematical model(right) [11]

2.2.2 Convolutional Layer

The convolutional layer is the most important layer in a Dense CNN(DCNN). The purpose of convolutional layers is to extract dense features from the input images. It consists of multiple feature maps, each feature map can recognize certain specific feature. It contains a set of convolutional filters, whose parameters need to learn in the training. The filtering is essentially done through weight adjustments. These filters are generally with a small size, but have the same channels number as the input. It means each pixel of input data share the same filter parameters.

There are two reasons for sharing parameters. One is that, when the input has large channel number (e.g. the RGB image has 3), it is too time-consuming to calculate the connections to all of the neurons. The fully connected architecture cannot analyze the local spatial structure of the data. The second reason relies on an assumption that, if a local feature is important to calculate at certain positions, it should be also important at any other positions. Therefore, the convolutional layer enforces the local spatially correlation by limiting connection size and sharing the parameters. This small region can be also thought as the receptive field in this layer. This design makes sure that the learnable convolutional filters can generate the strong activation in any position for specific input pattern.

Regular Neural Networks which are only made of linear and activation layers do not scale well when dealing with full images. However, convolution layers take advantage of the fact that their input exhibits many spatial relationships. In fact, neighboring pixels should not be affected by their location within the image. Thus, a convolutional layer learns a set of N_k filters $F = f_1, f_2, ..., f_{N_k}$, which are convolved spatially with input image x, to produce a set of N_k 2D features maps z:

$$z_k = f_k \otimes x \tag{2.2}$$

where $\otimes$ is the convolution operator. When the filter correlates well with a region of the input image, the response in the corresponding feature map location is strong. Unlike conventional linear layer, weights are shared over the entire image reducing the number of parameters per response and equivariance is learned ($i.e.$ an object shifted in the input image will simply shift the corresponding responses in a similar way). Figure 2.6 shows 2×2 convolution.

Also, a fully connected layer can be seen as a convolutional layer with a filter of sizes $1\times1\times m$, where m is the input size. In addition, in CNNs, each filter f_i is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a feature map.

There are 3 hyper-parameters which determine the size of the output data. They are the number of filters, stride of sliding window and the size of zero-padding. The channel number of output data is determined by the filter number of this layer. These filters learn to give strong response for some certain patterns in the input. For example, a raw image is given into a convolutional layer. Then different filters may generate in different types of oriented edges or circles.

The stride determines the sliding length of the filter. Because the filter size is generally much smaller than the input data, the filters need to scan the input area in order to calculate the convolution across the input data. When the stride is 1 then the filters move 1 pixel per step.

The zero-padding size determines the how many zeros will be padded on the border of the input in convolutions. Padding provides a method to adjust the size of output data. Specifically, the zero-padding can be applied to generate the exact same size of input data in the output.

Therefore, the size of the output data can be determined by the above 3 hyper-parameters. Assume that the input data has size L, the filter size convolutional layer is F, the stride of filters is S, and the number of zero-padding is Z. Then the output size can be calculated as $(L - F + 2Z)/S + 1$. The output size may be not integer when the stride and the zero-padding number are not set correctly. Thus the filters cannot tightly fit the input data. Particularly, letting the number zero-padding $Z = (F - 1)/2$ ensures that the output data have the same spatially size with the input data when $S = 1$. Although it is generally not a requirement to keep the size of the previous layer, for example, one can use just a portion of padding.

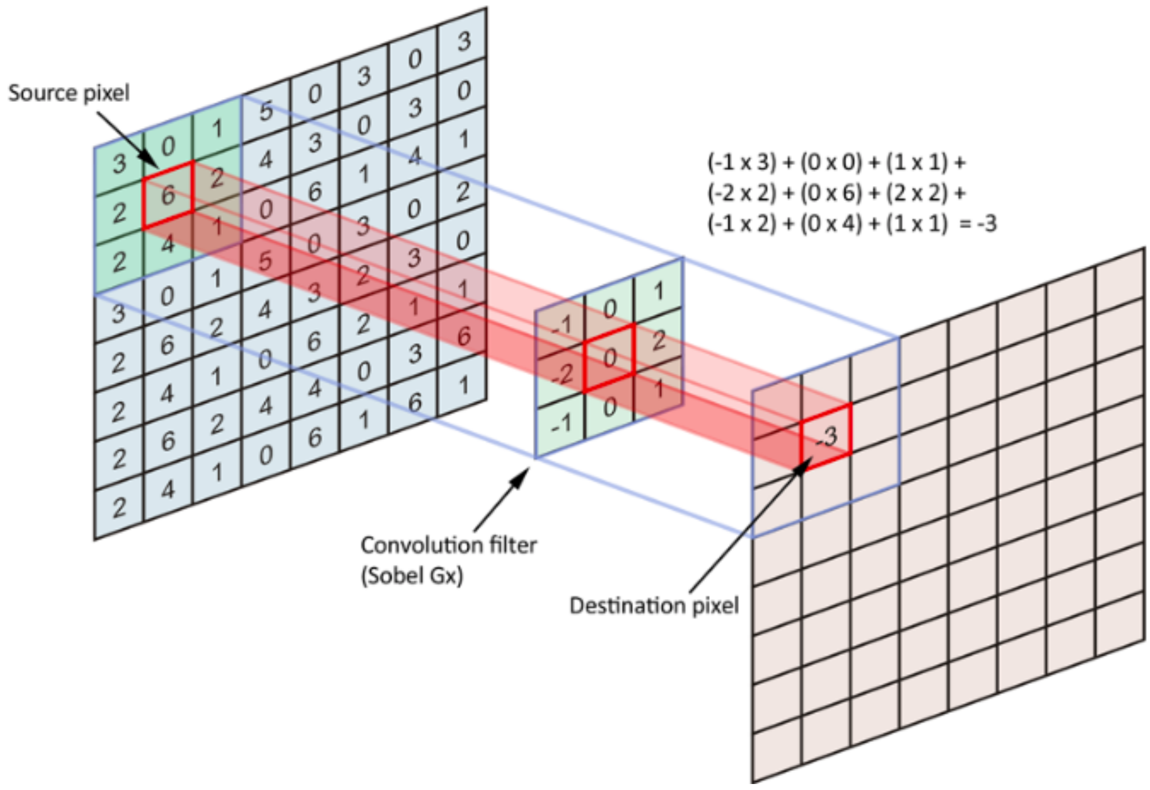

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

Figure 2.6 Convolution with 3×3 filter

2.2.3 Pooling Layer

Pooling layer is another important layer of CNN. It was also named subsampling layer. In CNNs, a pooling layer is typically present to provide invariance to slightly different input images and to reduce the dimension of the feature maps:

$$P_R = P_{i \in R}(z_i) \tag{2.3}$$

where P is a pooling function over the region of pixels R. The pooling layer conducts a non-linear down sampling to the input data. There are serval non-linear functions to conduct the pooling. In details, the input data is scanned by a window, whose stride must be equal to the window size. Then for each window position, the maximum will be output in the max pooling. The essential thought of pooling layer is that the local feature in a specific location is less important than the spatial relationship with other features. Usually, there are multiple pooling layers in the DCNN to gradually make the size of the data smaller and smaller. It means the calculation time is also reduced. Moreover, this reduction can also control overfitting, because the pooling operation can be considered as another kind of geometry invariance. Generally, pooling layers are periodically inserted between two successive convolutional layers in a CNN architecture.

There are several non-linear functions to implement pooling among which max pooling is the most common. Max pooling is preferred as it avoids cancellation of negative elements and prevents blurring of the activations and gradients throughout the network since the gradient is placed in a single location during backpropagation. Figure 2.7 shows an example of max pooling. In addition to max pooling, the pooling units can use other functions, such as average pooling.

The pooling layer operates independently on every channel of the input and down samples it spatially. The most common pooling size is 2×2. It means a 2×2 window scans with a stride of 2 along both width and height at every channel and down samples the input data. This pooling generates an output with 1/4 size of the input. The number of channels does not change through the pooling layer. Therefore, the pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters and amount of computation in the network, and hence to control overfitting. It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture. The spatial pooling layer is defined by its aggregation function, the high and width dimensions of the area where it is applied, and the properties of the convolution (such as padding or stride).

Because of the rapid decrease of the data size, the convolution layers after the pooling layer actually involve a lagers area in the raw image scale. This allows the succeeding convolution layers to learn more complex pattern from a large spatial scale, even their filters sizes do not increase. On the other hand, the pooling layer may result in the loss the detail information in the raw image. For example, if the target of network is to detect some small objects in the image, the pooling layers should avoid being used too much in the network.

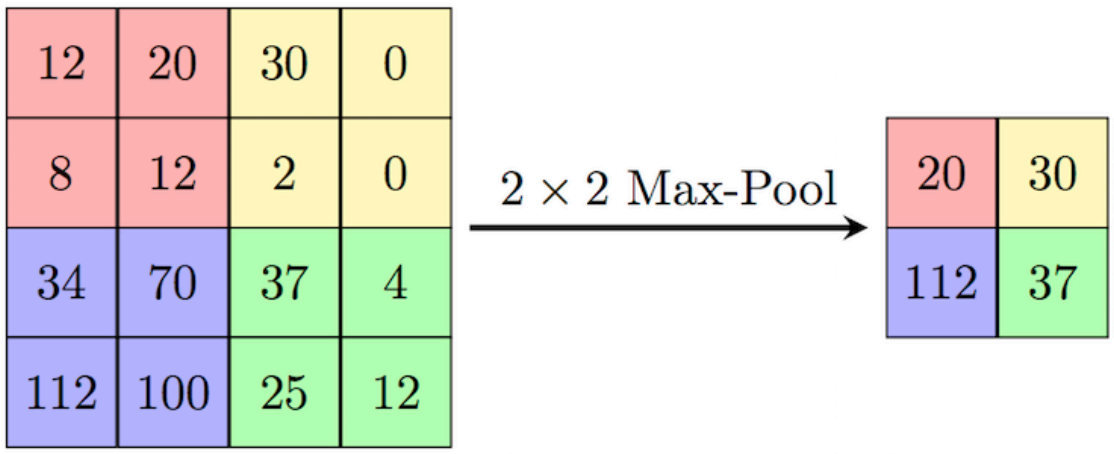


Figure 2.7 Max pooling with 2×2 filter

2.2.4 Batch Normalization

The batch normalization layer shifts the input data to a certain mean and variance. It is proposed by Ioffe and Szegedy [39]. The essential thought behind the batch normalization is to reduce the internal covariate shift.

The normalization operation is often used as the pre-processing in the traditional machine learning, such as SIFT and HOG. As in the DCNN, the values are automatically adjusted in the network. However, sometimes it makes the data too big or too small again. This condition is called an internal covariate shift in [39]. By normalizing the data in each training mini-batch, this problem can be significantly avoided.

The Batch Normalization (BN) layer quickly became very popular mostly because it helps to converge faster. It adds a normalization step (shifting inputs to zero-mean and unit variance) to make the inputs of each trainable layers comparable across features. By introducing the batch normalization layer, a higher learning rate can be applied to accelerate the training process. Generally, the learning rate for DCNN is set to a small value, so that only a small portion of gradients corrects the weights. The reason is that one does not want the outlier data to affect the whole network. By batch normalization, these outlier data are reduced. Therefore a higher learning rate can be adopted to improve the learning speeding.

2.2.5 Dropout Layer

The dropout layer randomly sets a proportion of the input data to zero. The aim of dropout layer is to reduce overfitting. The DCNN is prone to overfitting when layers are too many or the number of parameters is too large. At each training batch, the dropout layers "drop out" the net with probability 1 - *p*, where the *p* is the dropout rate. The connections between the input data and output data are also deleted. Only the remained network is updated in one training mini-batch. Then the weights of deleted nodes are reset to their values. In every training mini-batch, the dropout nodes are randomly selected. In the prediction stage, the dropout layer does not dropout anything or can be directly removed.

In the training stages, the drop rate is generally set as 0.5. However, for dropout layers which near to the first input layer, the drop rate may adjust to a lower value, in order to avoid losing too much information at the beginning.

The dropout layers avoid overfitting in the training. They also significantly reduce the training time. Although the dropout layers seem to reduce node connections, they actually force the network to obtain more robust parameters that produce better results in new data.

2.2.6 Loss Layer

Loss layer determines the approach to measure the difference between the predictions and ground truths. It is commonly the output layer in the training step and could be removed in the trained network. There are many loss functions can be used for different tasks. For example, softmax loss is used for predicting a single class from *K* mutually exclusive classes. Sigmoid cross-entropy loss is used for predicting *K* independent probabilistic values from 0 to 1. Euclidean loss is used for regressing to real value labels $(-\infty,+\infty)$.

2.3. Activation Functions

The capacity of the neural networks to approximate any functions, especially nonconvex, is directly the result of the non-linear activation functions. Every kind of activation function takes a vector and performs a certain fixed point-wise operation on it. Mainly, all the activation functions can be divided into two groups, which are free-parametric activation function and parametric activation functions. In this section, we will introduce various kinds of non-linear activation function, and how they work.

2.3.1 Parameterized Activation Functions

In this section, we will introduce some basic activation functions with no parameters needed to learn, which are Step function, Sigmoid function, Tanh function, ReLu function, and LeakyReLu function [40].

Step Function

$$y = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{2.4}$$

Sigmoid Function

$$y = sigmoid(x) = \frac{1}{1+e^{-x}} \tag{2.5}$$

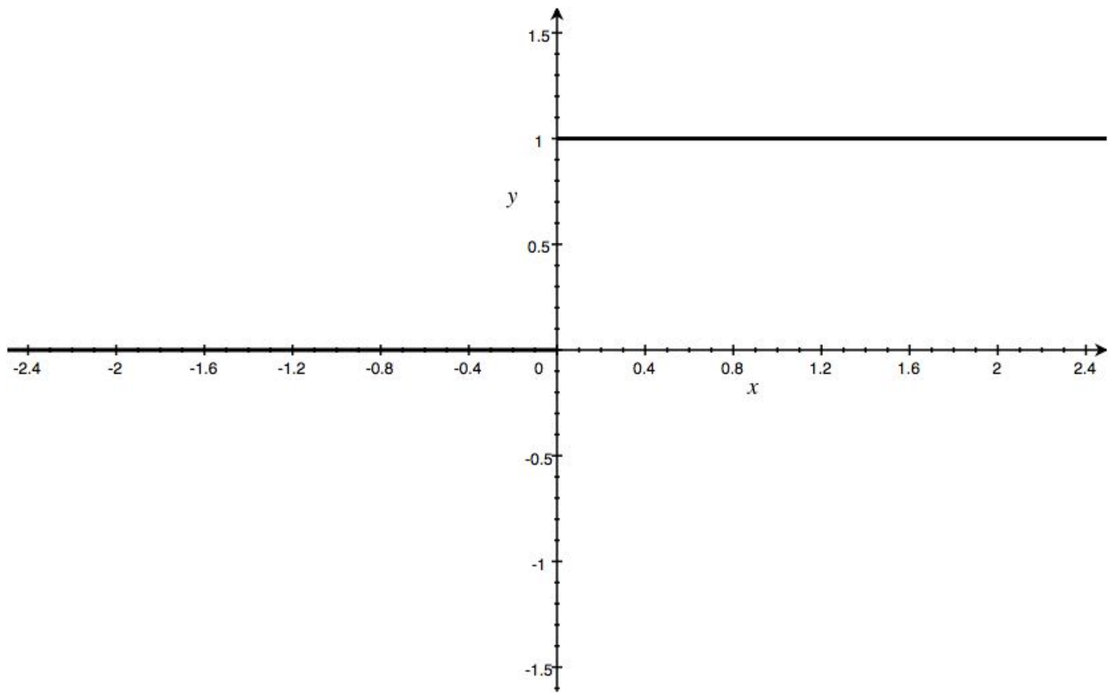


Figure 2.8 Step function

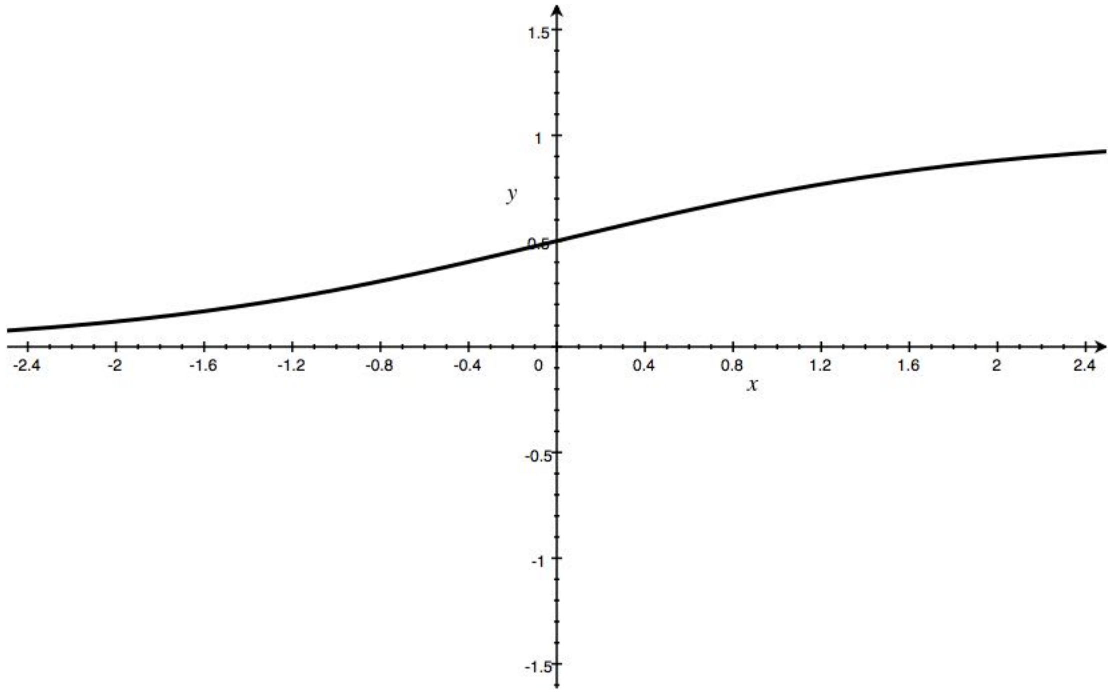


Figure 2.9 Sigmoid function

Tanh Function

$$y = tanh(x) = \frac{2}{1+e^{-2x}} - 1 = 2sigmoid(2x) - 1 \tag{2.6}$$

ReLu Function

$$y = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{2.7}$$

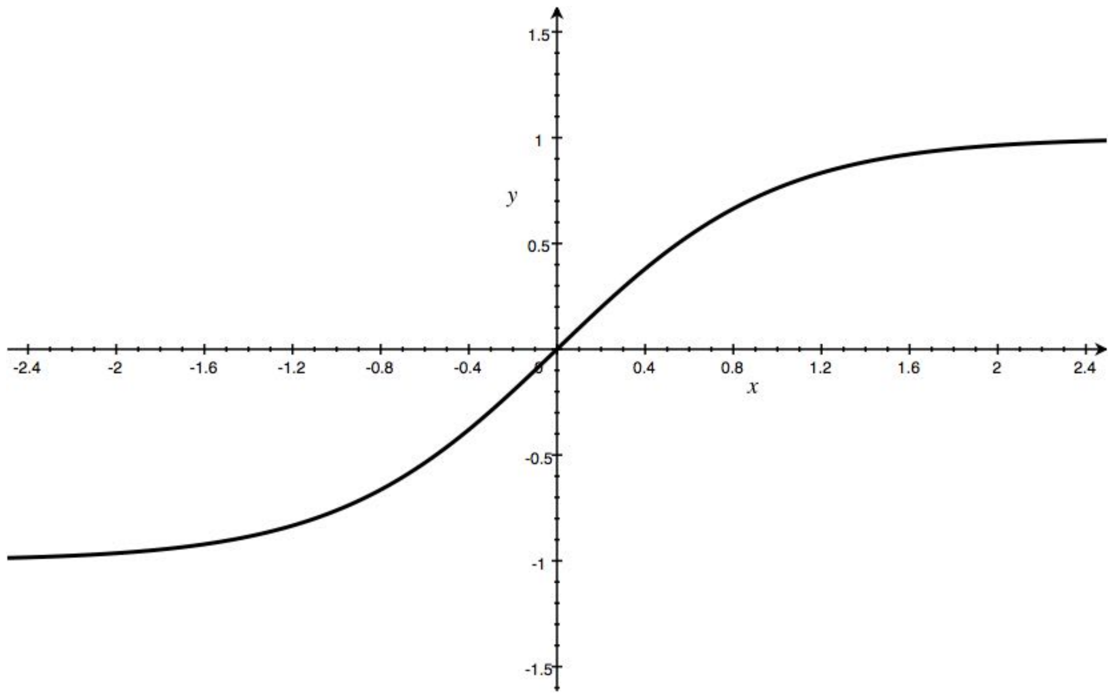


Figure 2.10 Tanh function

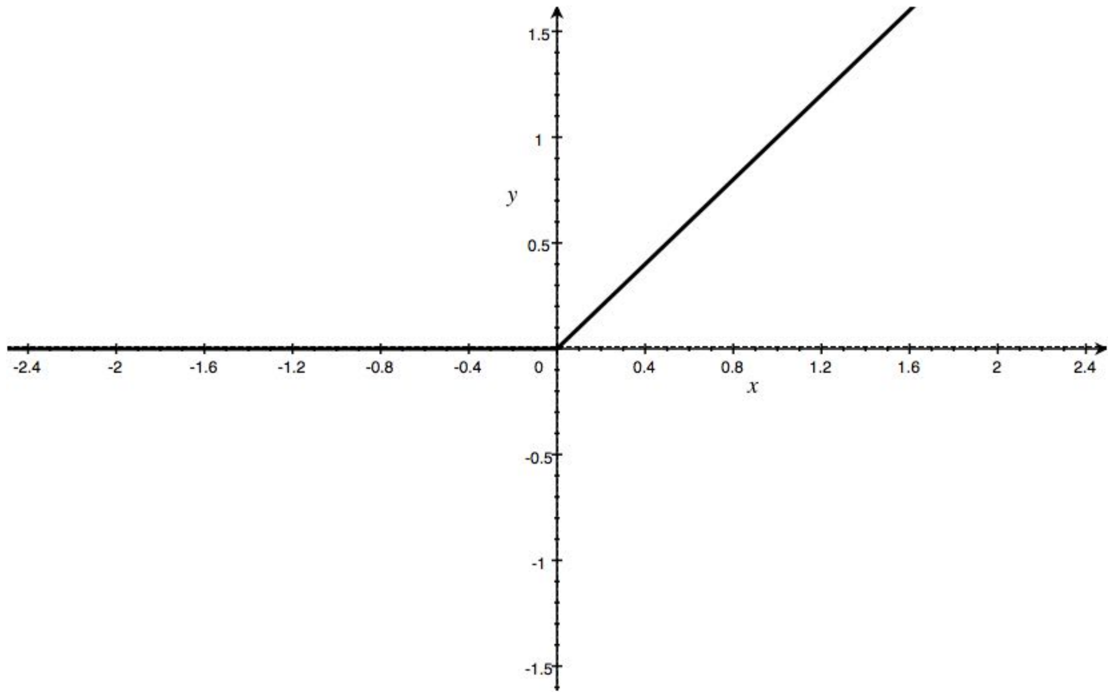


Figure 2.11 ReLu function

LeakyReLu Function

$$y = \begin{cases} x & x > 0 \\ \lambda x & x \leq 0 \end{cases} \tag{2.8}$$

where λ is a small value which is usually set to 0.2.

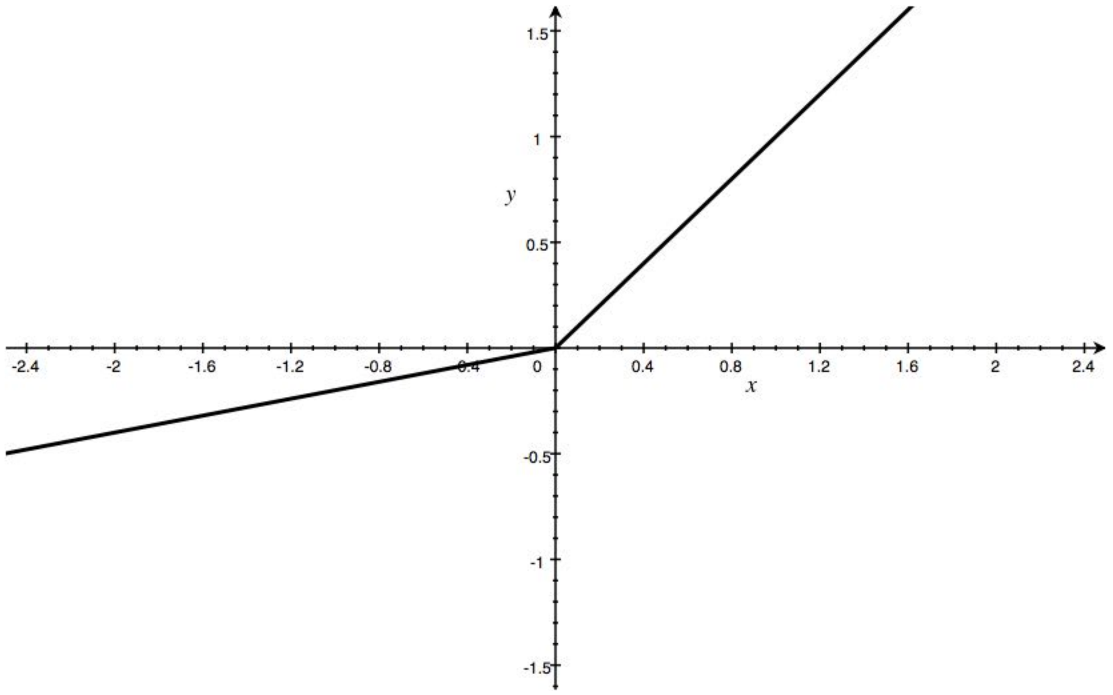


Figure 2.12 LeakyReLu function

2.3.2 Non-parametric Activation Function

In this section, we will introduce another type of non-linear activation function: parametric activation function. PReLu [41] (Parametric Rectified Linear Unit).

PReLu Function

$$y_i = \begin{cases} x_i & x_i > 0 \\ \lambda_i x_i & x_i \leq 0 \end{cases} \tag{2.9}$$

As its name suggests, λ_i are parameter need to be learn. If we set λ_i to 0, then PReLu becomes ReLu. If we set λ_i to a small fixed value, it becomes LeakyReLu.

2.4. Network Architectures

In this section, we will introduce several important modern DNNs architectures. LeNet [42] the first successful application of Convolutional neuron networks (CNNs) to digit recognition, developed by Yann LeCun in 1990. It consists of a sequence of Convolutional, Max Pooling layers followed by a Fully Connected layer. AlexNet [10] popularized CNNs in computer vision, did really well on the ImageNet ILSVRC [43] challenge in 2012, showing significant gains in performance. The network has similar architecture to LeNet but is deeper and bigger and features convolutional layers stacked on top of each other. DeConvNet [44] demonstrates how to visualize convolutional layer's learning results. VGG16 [11] demonstrated the importance of depth as a critical component to good performance, it was a runner-up in ILSVRC 2014. The architecture consists of a stacked convolutional and max pooling layers, with increasing depth and it uses a large number of parameters due to the final fully connected layers. GoogLeNet [12] proposed from Google won the ILSVRC 2014 challenge. The architecture consists of inception modules that dramatically reduce the number of parameters, it uses multi-scale 3×3, 5×5 convolutional filters including 1×1 convolutions for dimensionality reduction. ResNet [13] residual network was the winner of ILSVRC 2015, it introduces skip connections for easier training that enable very deep architectures and makes use of batch normalization. Each of those famous modern structure will be introduced in the following subsections.

2.4.1 LeNet

The LeNet architecture is an excellent "first architecture" for CNN (especially when trained on the MNIST dataset, an image dataset for handwritten digit recognition).

LeNet is small and easy to understand, but the parameters it consists are enough to provide interesting results. Furthermore, the combination of LeNet + MNIST is able to run on the CPU, making it easy for beginners to take their first step in Deep Learning and Convolutional Neural Networks. Figure 2.13 shows the architecture of LeNet. The LeNet architecture consists of the following layers which are convolutional layer, pooling layer, and fully-connected layer.

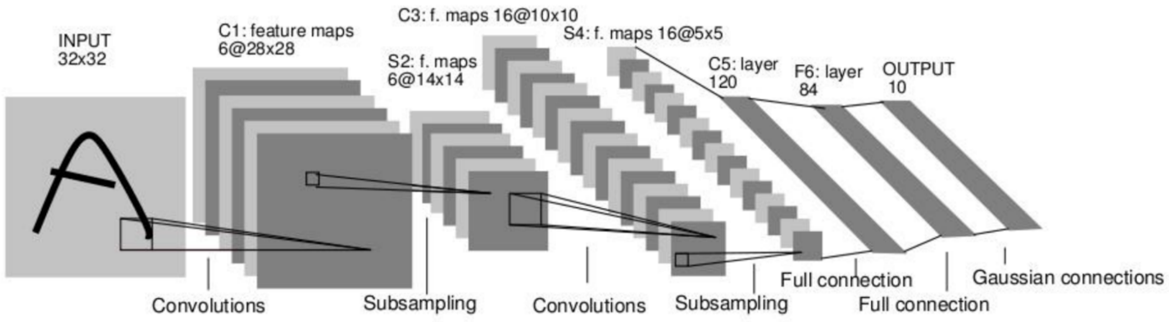


Figure 2.13 Architecture of LeNet

2.4.2 AlexNet

The one that started it all (Though some may say that Yann LeCun's paper in 1998 was the real pioneering publication). This paper, titled "ImageNet Classification with Deep Convolutional Networks" [10], has been cited a total of 17,955 times and is widely regarded as one of the most influential publications in the field. Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton created a "large, deep convolutional neural network" that was used to win the 2012 ILSVRC (ImageNet Large-Scale Visual Recognition Challenge). For those that aren't familiar, this competition can be thought of as the annual Olympics of computer vision, where teams from across the world compete to see who has the best computer vision model for tasks such as classification, localization, detection, and more. 2012 marked the first year where a CNN was used to achieve a top 5 test error rate of 15.4% (Top 5 error is the rate at which, given an image, the model does not output the correct label with its top 5 predictions). The next best entry achieved an error of 26.2%, which was an astounding improvement that pretty much shocked the computer vision community. Safe to say, CNNs became household names in the competition from then on out.

In the paper, the group discussed the architecture of the network (which was called AlexNet). They used a relatively simple layout, compared to modern architectures. The network was made up of 5 conv layers, max-pooling layers, dropout layers, and 3 fully connected layers. The network they designed was used for classification with 1000 possible categories. Figure 2.14 shows the architecture of AlexNet. The Main points of AlexNet includes: (1) Trained the network on ImageNet data, which contained over 15 million annotated images from a total of over 22,000 categories. (2) Used data augmentation techniques that consisted of image translations, horizontal reflections, and patch extractions. (3) Implemented dropout layers in order to combat the problem of overfitting to the training data. (4) Trained the model using batch stochastic gradient descent, with specific values for momentum and weight decay.

The neural network developed by Krizhevsky, Sutskever, and Hinton in 2012 was the coming out party for CNNs in the computer vision community. This was the first time a model performed so well on a historically difficult ImageNet dataset. Utilizing techniques that are still used today, such as data augmentation and dropout, this paper really illustrated the benefits of CNNs and backed them up with record breaking performance in the competition.

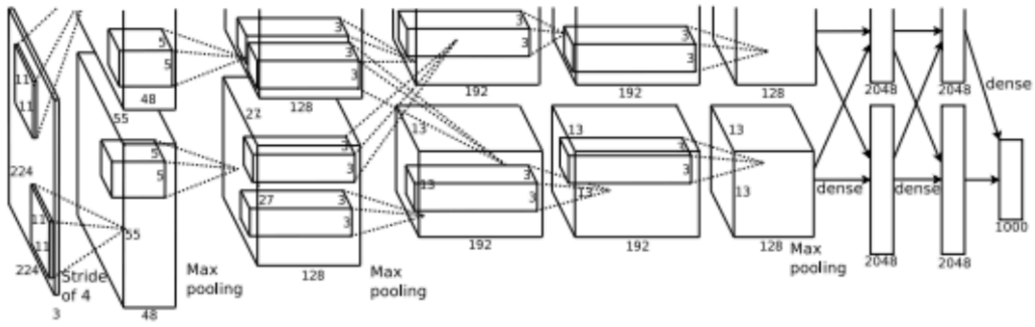


Figure 2.14 Architecture of AlexNet [10]

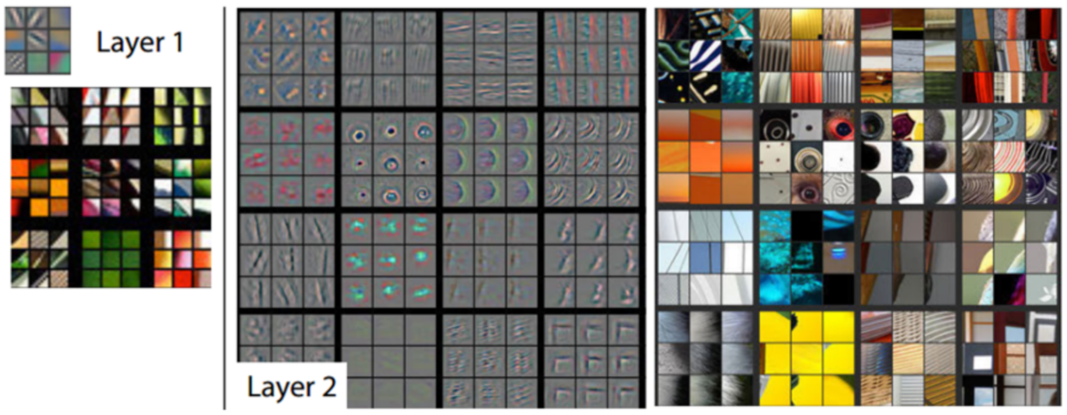


Figure 2.15 Visualizations of layer 1 and layer 2. Each layer illustrated 2 pictures, one which shows the filters themselves and that shows what part of the image are most strongly activated by the given filter. For example, in the space labeled layer 2, we have representation of the 16 different filters (on the left) [45]

2.4.3 DeConvNet

The basic idea behind how this works is that at every layer of the trained CNN, you attach a "DeConvNet" which has a path back to the image pixels. An input image is fed into the CNN and activations are computed at each level. This is the forward pass. Now, let's say we want to examine the activations of a certain feature in the 4th conv layer.

We would store the activations of this one feature map, but set all of the other activations in the layer to 0, and then pass this feature map as the input into the deconvnet. This deconvnet has the same filters as the original CNN. This input then goes through a series of unpooling (reverse maxpooling), rectify, and filter operations for each preceding layer until the input space is reached. The reasoning behind this whole process is that we want to examine what type of structures excite a given feature map. Figure 2.15 shows the visualizations of the first and second layers.

The first layer of the ConvNet is always a low level feature detector that will detect simple edges or colors in this particular case. Figure 2.16 gives the more visualization on 3,4,5 layers. These layers show a lot more of the higher level features such as dogs' faces or flowers. This means the proceeding layer try to learn more high-level features.

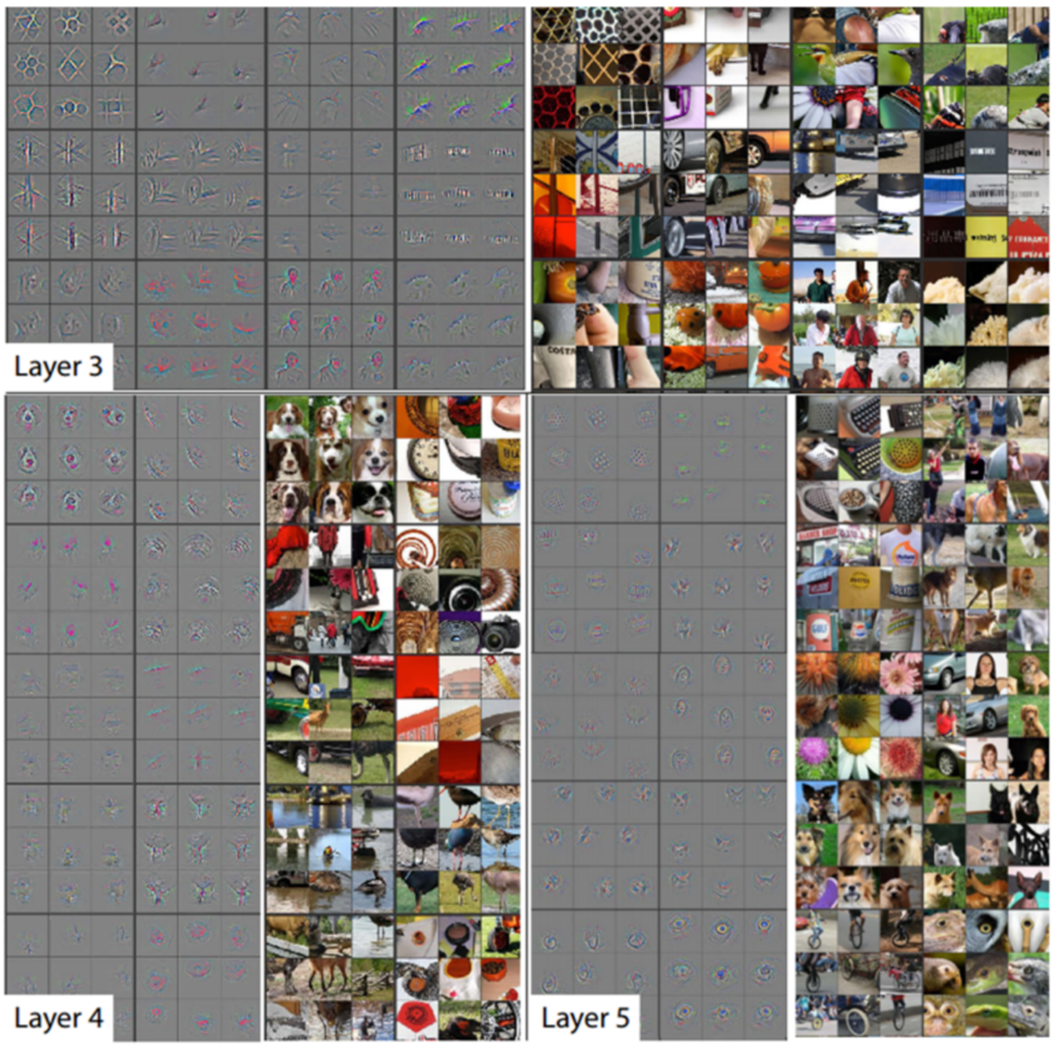
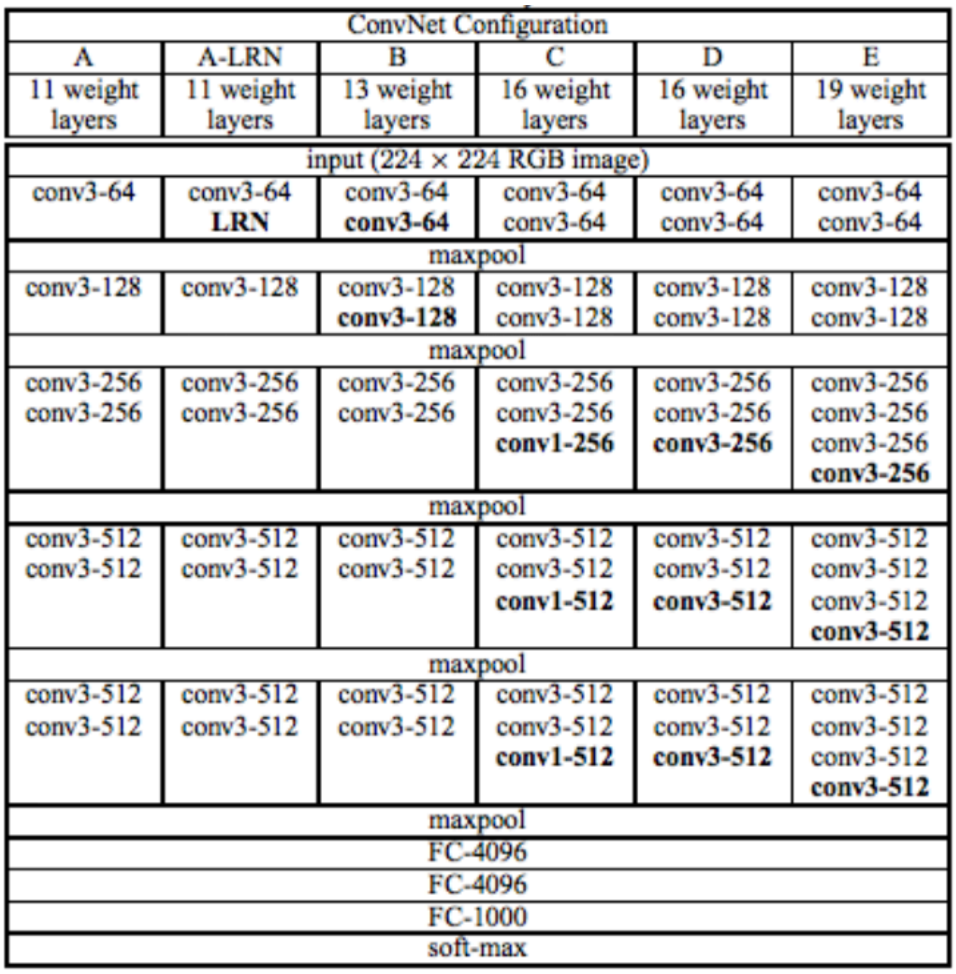


Figure 2.16 More visualization on layer 3,4,5 [45]

2.4.4 VGGNet

Figure 2.17 shows 6 different architectures of VGGNet. The keypoints in VGGNet includes: (1) The use of only 3x3 sized filters is quite different from AlexNet’s 11x11 filters in the first layer and ZF Net’s 7x7 filters. The authors’ reasoning is that the combination of two 3x3 conv layers has an effective receptive field of 5x5. This in turn simulates a larger filter while keeping the benefits of smaller filter sizes. One of the benefits is a decrease in the number of parameters. Also, with two conv layers, we’re able to use two ReLU layers instead of one. (2) 3 conv layers back to back have an effective receptive field of 7x7. (3) As the spatial size of the input volumes at each layer decrease (result of the conv and pool layers), the depth of the volumes increases due to the increased number of filters as you go down the network. (4) Interesting to notice that the number of filters doubles after each max-pooling layer. This reinforces the idea of shrinking spatial dimensions, but growing depth.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.17 The 6 different architectures of VGGNet in original paper [11]

VGGNet is one of the most influential papers in my mind because it reinforced the notion that convolutional neural networks have to have a deep network of layers in order for this hierarchical representation of visual data to work. Keep it deep. Keep it simple.

2.4.5 GoogLeNet

The Inception module has been proposed by GoogLeNet [12]. Figure 2.18 shows the architecture of a full Inception. GoogLeNet is a 22-layer CNN and was the winner of ILSVRC 2014 with a top 5 error rate of 6.7%. To my knowledge, this was one of the first CNN architectures that really strayed from the general approach of simply stacking convolution and pooling layers on top of each other in a sequential structure. The authors of the paper emphasized that this new model places notable consideration on memory and power usage.

In Figure 2.18, the bottom green box is our input and the top one is the output of the model (Turning this picture right 90 degrees would let you visualize the model in relation to the last picture which shows the full network). Basically, at each layer of a traditional ConvNet, you have to make a choice of whether to have a pooling operation or a conv operation (there is also the choice of filter size). What an Inception module allows you to do is perform all of these operations in parallel. In an Inception module, we have a medium sized filter convolution, a large-sized filter convolution, and a pooling operation. The network in network convolution is able to extract information about the very fine grain details in the volume, while the 5x5 filter is able to cover a large receptive field of the input, and thus able to extract its information as well. It also has a pooling operation that helps to reduce spatial sizes and combat overfitting. On top of all of that, it has a ReLu layer after each conv layer, which helps improve the nonlinearity of the network. Basically, the network is able to perform the functions of these different operations while still remaining computationally considerate.

2.4.6 ResNet

Imagine a deep CNN architecture. Take that, double the number of layers, add a couple more, and it still probably isn’t as deep as the ResNet architecture that Microsoft Research Asia came up with in late 2015. ResNet is a new 152-layer (the deepest version) network architecture that set new records in classification, detection, and localization through one incredible architecture. Aside from the new record in terms of the number of layers, ResNet won ILSVRC 2015 with an incredible error rate of 3.6% (Depending on their skill and expertise, humans generally hover around a 5-10% error rate).

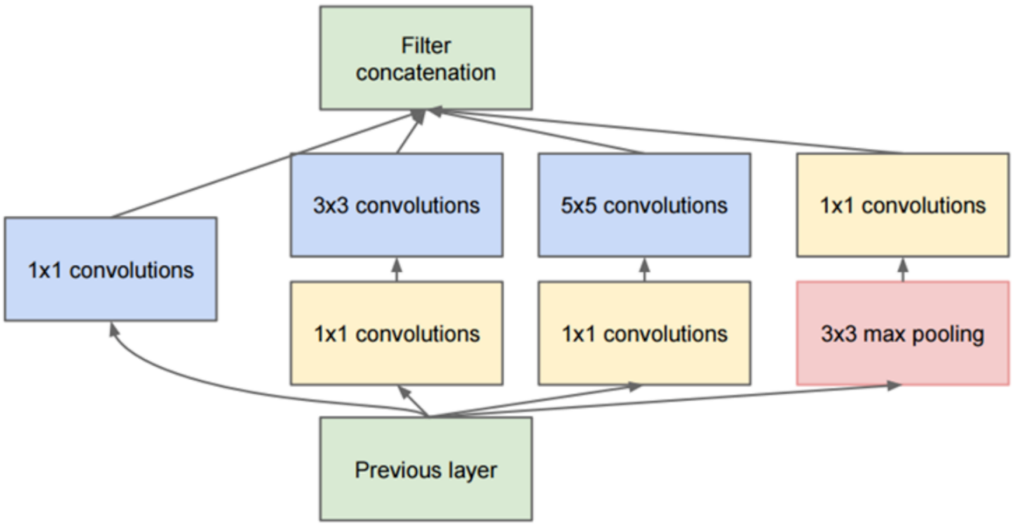


Figure 2.18 Full inception module architecture [12]

Except its depth, ResNet introduce a new module called Residual Block [13] see in Figure 2.19. The idea behind a Residual Block is that you have your input x go through conv-relu-conv series. This will give you some $F(x)$. That result is then added to the original input x. Let's call that $H(x) = F(x) + x$. In traditional CNNs, your $H(x)$ would just be equal to $F(x)$. So, instead of just computing that transformation (straight from x to $F(x)$, we're computing the term that you have to add $F(x)$ to your input, x. Basically, the mini module shown below is computing a "delta" or a slight change to the original input x to get a slightly altered representation (When we think of traditional CNNs, we go from x. to $F(x)$. which is a completely new representation that doesn't keep any information about the original x). The authors believe that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping [13].

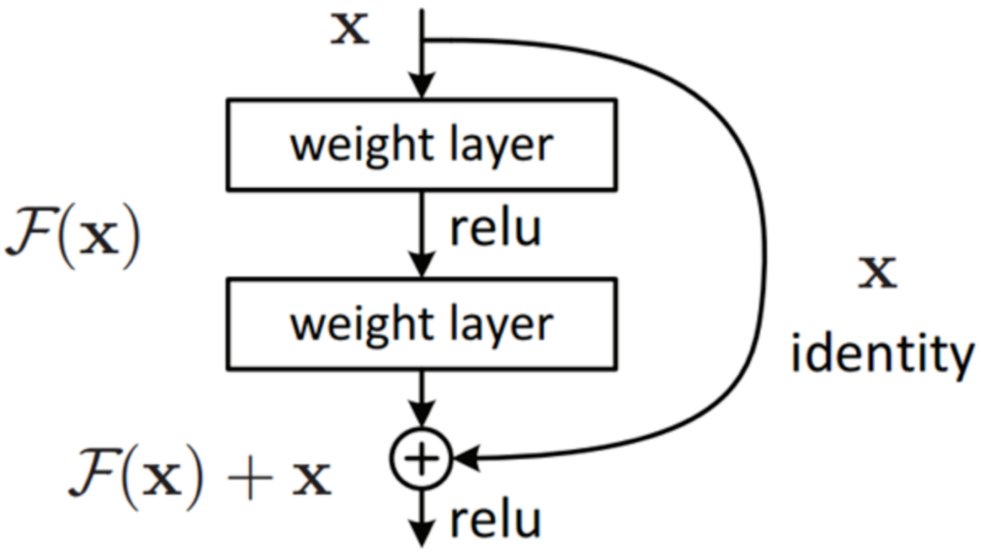

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.19 Illustration of a Residual Block [13]

2.5. Training

Training is an important concept of neural networks. The purpose of training process is to find the most optimal parameters of the network for solving the certain task. Before the training process starts, network parameters need to be initialized. Initial values are often chosen randomly, however using some heuristics may lead to a faster parameter adjustment towards the optimal values. By feeding the training data through the network, neural networks can learn feature information hidden in the training set. This is an iterative process, where the outputs produced on each input from the training set are analyzed and the network is repeatedly being adjusted to produce better results. The network is considered to be well trained after reaching the target performance on the training data.

The training of neural networks is based on gradient back propagation [46]. Namely, by computing the difference between the output of a neural network and the expected output, the error between these two outputs will be propagated back through the whole network, which can guide the parameters updating. Section 2.5.1 will give more details on back propagation algorithm. The loss functions that define the differences between network outputs and ground truths will be introduced in Section 2.5.2. Also, there are various strategies used for updating network parameters, which will be introduced s In Section 2.5.3.

2.5.1 Back Propagation

Back propagation is a method used in artificial neural networks to calculate the error contribution of each neuron after a batch of data (in image recognition, multiple images) is processed. It is a special case of an older and more general technique called automatic differentiation. In the context of learning, back propagation is commonly used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function. This technique is also sometimes called backward propagation of error, because the error is calculated at the output and distributed back through the network layers.

The back propagation algorithm has been repeatedly rediscovered and is equivalent to automatic differentiation in reverse accumulation mode. Back propagation requires an expected output for each input value. It is therefore considered to be a supervised training strategy. Back propagation is also a generalization of the delta rule to multi-layered feedforward networks, which becomes possible by using the chain rule to iteratively compute gradients for each layer. It is closely related to the Gauss-Newton algorithm, and also a part of continuing research in neural backpropagation.

To understand the mathematical derivation of the backpropagation algorithm, it helps to first develop some intuition about the relationship between the actual output of a neuron and the correct output for a particular training example. Consider a simple neural network with two input units, one output unit and no hidden units. Each neuron uses a linear output that is the weighted sum of its input.

Initially, before training, the weights will be set randomly. Then the neuron learns from training examples, which in this case consist of a sets of tuples (x_1, x_2, G) where x_1 and x_2 are the inputs to the network and G is the correct output (the output the network should produce given those inputs, when it has been trained). The initial network which is given x_1 and x_2 will compute an output y that likely differs from G (given random weights). A common method for measuring the discrepancy between the expected output t and the actual output y is the squared error measure:

$$E = (G - y)^2 \tag{2.10}$$

where E is the discrepancy or error.

As an example, consider the network on a single training case: (1,1,0), thus the input $_1$ and x_2 are 1 and 1 respectively and the correct output G is 0. Now if the actual output y is plotted on the horizontal axis against the error E on the vertical axis, the result is a parabola. The minimum of the parabola corresponds to the output y which minimizes the error E. For a single training case, the minimum also touches the horizontal axis, which means the error will be zero and the network can produce an output y that exactly matches the expected output G. Therefore, the problem of mapping inputs to outputs can be reduced to an optimization problem of finding a function that will produce the minimal error.

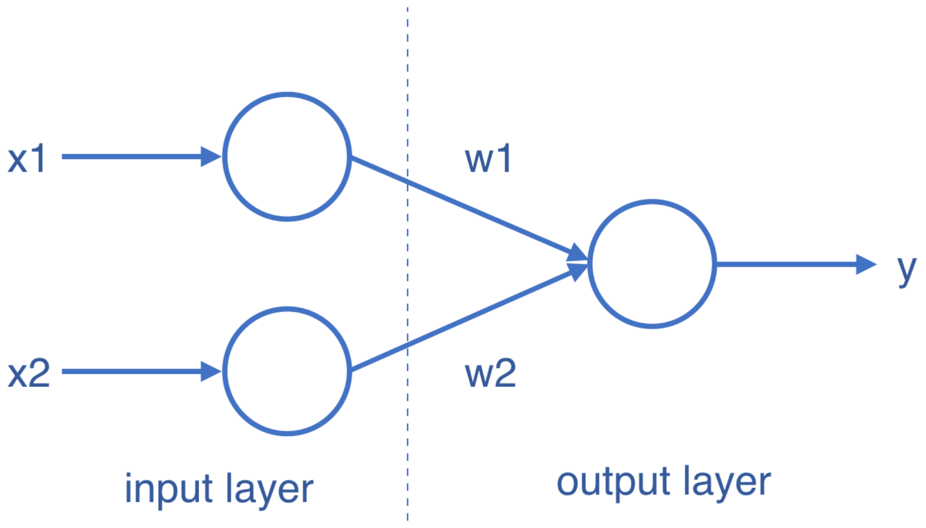


Figure 2.20 A simple neural network with two input units and one output unit

However, the output of a neuron depends on the weighted sum of all its inputs (shown in Figure 2.20):

$$y = x_1 w_1 + x_2 w_2 \tag{2.11}$$

where w_1 and w_2 are the weights on the connection from the input units to the output unit. Therefore, the error also depends on the incoming weights to the neuron, which is ultimately what needs to be changed in the network to enable learning. If each weight is plotted on a separate horizontal axis and the error on the vertical axis, the result is a parabolic bowl. For a neuron with k weights, the same plot would require an elliptic paraboloid of $k + 1$ dimensions.

One commonly used algorithm to find the set of weights that minimizes the error is gradient descent. Backpropagation is then used to calculate the steepest descent direction. The gradient descent method involves calculating the derivative of the squared error function with respect to the weights of the network. This is normally done using backpropagation. Assuming one output neuron, the squared error function is:

$$E = \frac{1}{2}(G - y)^2 \tag{2.12}$$

where E is the squared error, G is the target output for a training sample, and y is the actual output of the output neuron.

The factor of $1/2$ is included to cancel the exponent when differentiating. Later, the expression will be multiplied with an arbitrary learning rate.

For each neuron j, its output O_j is defined as (shown in Figure 2.21):

$$O_j = \varphi(N_j) = \varphi\left(N_j \textstyle\sum_{k=1}^{n} w_{kj} O_k\right) \tag{2.13}$$

where input N_j to a neuron is the weighted sum of outputs O_k of previous neurons. If the neuron is in the first layer after the input layer, the O_k of the input layer are simply the inputs x_k to the network. The number of input units to the neuron is n. The variable w_{kj} denotes the weight between neurons k and j.

The activation function φ is non-linear and differentiable. A commonly used activation function is the logistic function:

$$\varphi(z) = \frac{1}{1+e^{-z}} \tag{2.14}$$

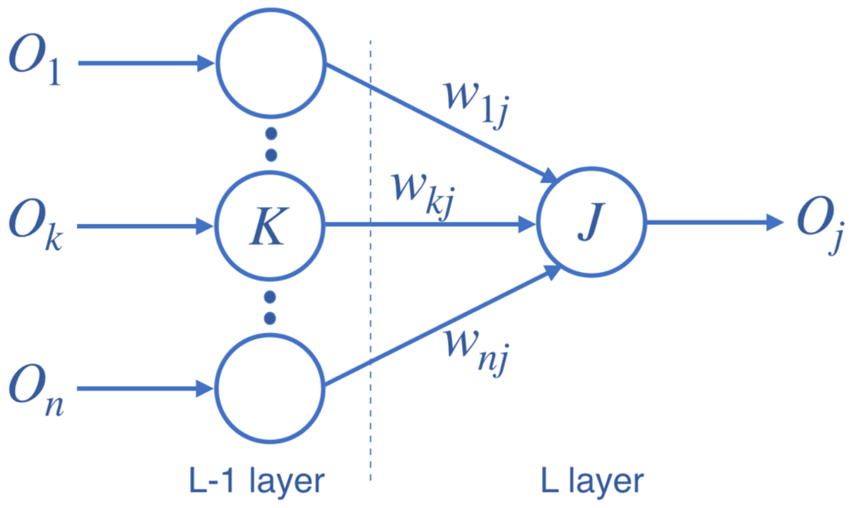


Figure 2.21 Calculation between two layers

which has a convenient derivative of:

$$\frac{d\varphi}{dz}(z) = \varphi(z)(1 - \varphi(z)) \tag{2.15}$$

Calculating the partial derivative of the error with respect to a weight w_{ij} is done using the chain rule twice:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial O_j}\frac{\partial O_j}{\partial N_j}\frac{\partial N_j}{\partial w_{ij}} \tag{2.16}$$

In the last factor of the right-hand side of the above, only one term in the sum N_j depends on w_{ij}, so that

$$\frac{\partial N_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}\left(\sum_{k=1}^{n} w_{kj}O_k\right) = \frac{\partial}{\partial w_{ij}} w_{ij}O_i \tag{2.17}$$

If the neuron is in the first layer after the input layer, O_i is just x_i.

The derivative of the output of neuron j with respect to its input is simply the partial derivative of the activation function (assuming here that the logistic function is used):

$$\frac{\partial O_j}{\partial N_j} = \frac{\partial}{\partial N_j}\varphi(N_j) = \varphi(N_j)(1 - \varphi(N_j)) \tag{2.18}$$

The first factor is straightforward to evaluate if the neuron is in the output layer, because then $O_j = y$ and

$$\frac{\partial E}{\partial O_j} = \frac{\partial E}{\partial y} = \frac{\partial}{\partial y}\frac{1}{2}(y - G)^2 = y - G \tag{2.19}$$

However, if j is in an arbitrary inner layer of the network, finding the derivative E with respect to O_j is less obvious. Considering E as a function of the inputs of all neurons $L = u, v, \ldots, w$ receiving input from neuron j,

$$\frac{\partial E(O_j)}{\partial O_j} = \sum_{\ell\in L}\left(\frac{\partial E}{\partial N_\ell}\frac{\partial N_\ell}{\partial O_j}\right) = \sum_{\ell\in L}\left(\frac{\partial E}{\partial O_\ell}\frac{\partial O_\ell}{\partial N_j}w_{j\ell}\right) \tag{2.20}$$

Therefore, the derivative with respect to O_j can be calculated if all the derivatives with respect to the outputs O_ℓ of the next layer are known. Putting it all together we will get:

$$\frac{\partial E}{\partial w_{ij}} = \delta_j O_j \tag{2.21}$$

Finally, we can summarize the parameter adjustment applicable to all the layers in a given network:

$$\Delta w_{ij} = -\eta\frac{\partial E}{\partial w_{ij}} = -\eta\delta_j O_j \tag{2.22}$$

The above explanation is based on a fully-connected layer with no bias added. It is the same case in the other types of layers. The idea is that we compute the error between the output of network and our expected value. Then, the gradient is back propagated from behind to the head of the neuron network for updating the weights.

2.5.2 Loss Function

Loss function is an important part when training DNNs, which is used for measuring the inconsistency between predicted value $\hat{y}$ and actual label y. It is a non-negative value, where the robustness of model increases along with the decrease of the value of loss function.

Mean Squared Error

Mean Squared Error (MSE), or quadratic, loss function is widely used in linear regression as the performance measure, and the method of minimizing MSE is called Ordinary Least Squares (OSL), the basic principle of OSL is that the optimized fitting line should be a line which minimizes the sum of distance of each point to the regression line, i.e., minimizes the quadratic sum. The standard form of MSE loss function is defined as:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(y^i - \hat{y}^i)^2 \tag{2.23}$$

where $(y^i - \hat{y}^i)$ is named as residual, and the target of MSE loss function is to minimize the residual sum of squares. It is worthing to mention that if using sigmoid function as the non-linear activation function, the quadratic loss function would suffer the problem of slow convergence (learning speed), for other activation functions, it would not have such problem.

Mean Squared Logarithmic Error

Based on our understanding, Mean Squared Logarithmic Error (MSLE) loss function is a variant of MSE, which is defined as:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} n(\log{(y^i + 1)} - \log{(\hat{y}^i + 1)})^2 \tag{2.24}$$

MSLE is also used to measure the difference between actual and predicted. By taking the log of the predictions and actual values, what changes is the variance that you are measuring. It is usually used when you do not want to penalize huge differences in the predicted and the actual values when both predicted and true values are huge numbers. Another thing is that MSLE penalizes under-estimates more than over-estimates.

L2 norm

L2 norm loss function is the square of the L2 norm of the difference between actual value and predicted value. It is mathematically similar to MSE, only do not have division by n, it is computed by the following:

$$\mathcal{L} = \sum_{i=1}^{n}(y^i - \hat{y}^i)^2 \tag{2.25}$$

Mean Absolute Error

Mean Absolute Error (MAE) is a quantity used to measure how close forecasts or predictions are to the eventual outcomes, which is computed by the following:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}\left|y^i - \hat{y}^i\right| \tag{2.26}$$

where $|\cdot|$ denotes the absolute value. Albeit, both MSE and MAE are used in predictive modeling, there are several differences between them. MSE has nice mathematical properties which make it easier to compute the gradient. However, MAE requires more complicated tools such as linear programming to compute the gradient. Because of the square, large errors have a relatively greater influence on MSE than do the smaller error. Therefore, MAE is more robust to outliers since it does not make use of square. On the other hand, MSE is more useful if concerning about large errors whose consequences are much bigger than equivalent smaller ones. MSE also corresponds to maximizing the likelihood of Gaussian random variables.

L1 norm

The L1 loss function is sum of absolute errors of the difference between actual value and the predicted value. Similar to the relation between MSE and L2 norm, L1 norm is mathematically similar to MAE, only do not have division by n, and it is defined as the following:

$$\mathcal{L} = \sum_{i=1}^{n} \left| y^i - \hat{y}^i \right| \tag{2.26}$$

Cross Entropy

Cross Entropy is commonly-used in binary classification (labels are assumed to take values 0 or 1) as a loss function (For multi-classification, use Multi-class Cross Entropy), which is computed by the following:

$$\mathcal{L} = -\frac{1}{n} \sum_{i=1}^{n} [y^i \log(\hat{y}^i) + (1 - y^i) \log(1 - \hat{y}^i)] \tag{2.27}$$

Cross entropy measures the divergence between two probability distribution, if the cross entropy is large, which means that the difference between two distribution is large, while if the cross entropy is small, which means that two distribution is similar to each other. As we have mentioned in MSE that it suffers slow divergence when using sigmoid as activation function, here the cross entropy does not have such problem. Similarly, $\hat{y}^i = \sigma(z^i) = \sigma(\theta^T x^i)$, and we only consider one training sample, by using sigmoid, we have $\mathcal{L} = y \log(\sigma(z)) + (1 - y) \log(1 - \sigma(z))$, and compute it derivative as the following:

$$\frac{\partial \mathcal{L}}{\partial \theta} = (y - \sigma(z)) \cdot x \tag{2.28}$$

compare to the derivative in MSE, it eliminates the term $\sigma(z)'$, where the learning speed is only controlled by $(y - \sigma(z))$. In this case, when the difference between predicted value and actual value is large, the convergence is fast. Generally, comparing to quadratic cost function, cross entropy cost function has the advantages that fast convergence and is more likely to reach the global optimization.

Hinge

Hinge Loss, also known as max-margin objective, is a loss function used for training classifiers. The hinge loss is used for "maximum-margin" classification, most notably for support vector machines (SVMs) [47]. For an intended output $y(i) = \pm 1$, i.e., binary classification and a classifier score $\hat{y}^i$, the hinge loss of the prediction y is defined as:

$$\mathcal{L} = \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y(i), \hat{y}^i) \tag{2.29}$$

Note that $\hat{y}^i$ should be the "raw" output of the classifier's decision function, not the predicted class label. It can be seen that when yi and y^i have the same sign (meaning $\hat{y}^i$ predicts the right class) and $|\hat{y}^i| > 1$, the hinge loss equals to zero, but when they have opposite sign, hinge loss increases linearly with $\hat{y}^i$ (one-sided error). However, there is a more general expression as the following:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \max{(0, m - y(i), \hat{y}^i)} \tag{2.30}$$

where m margin is a customized value. More details about extending to multi-classification.

2.5.3 Optimization Algorithms

In this section, we will introduce some basic algorithms for finding the optimal weights when training DNNs. This section focuses on one particular case of optimization: finding the parameters of a neural network that significantly reduce a cost function $J(\theta)$, which typically includes a performance measure evaluated on the entire training set. We will start with introducing the most fundamental algorithm Gradient Decent, then some more advanced algorithm will be introduced.

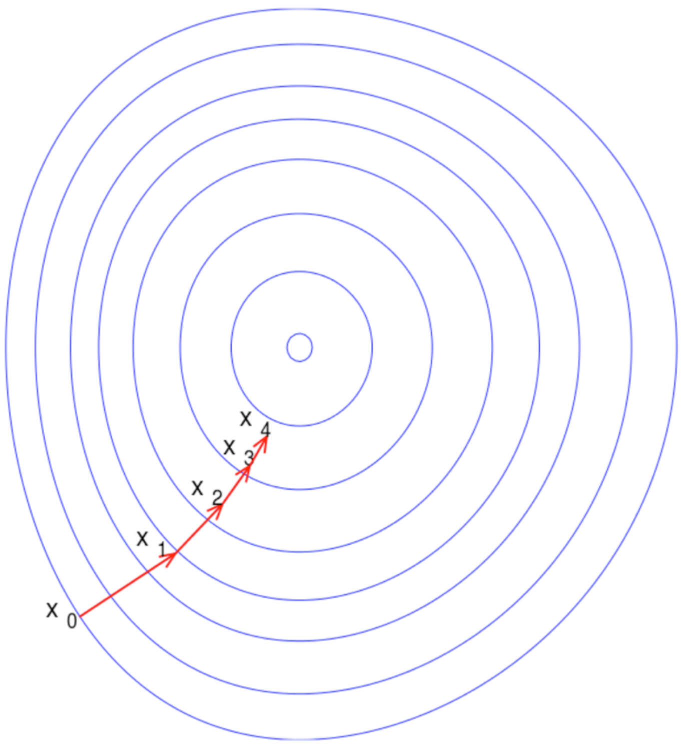


Figure 2.22 Illustration of gradient descent algorithm on a series of level sets

Gradient Descent

Gradient Descent (GD) is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. At the same time, every state-of-the-art Deep Learning library contains implementations of various algorithms to optimize gradient descent (e.g. tensorflow's [48], [49], and caffe's [49] documentation). Figure 2.22 illustrates the GD algorithm on a series of level sets.

Gradient descent is based on the observation that if the multi-variable function $F(x)$ is defined and differentiable in a neighborhood of a point $\mathbf{a}$, then $F(x)$ decreases fastest if one goes from a in the direction of the negative gradient of F at $(\mathbf{a}, -\nabla F(x))$. It follows that, if:

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n) \tag{2.31}$$

for γ small enough, then $F(\mathbf{a}_n) \geq F(\mathbf{a}_{n+1})$. In other words, the term $\gamma \nabla F(\mathbf{a})$ is subtracted from $\mathbf{a}$ because we want to move against the gradient, toward the minimum. With this observation in mind, one starts with a guess x0 for a local minimum of F, and considers the sequence $\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}, \ldots$, such that:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n)\,,\ n \geq 0 \tag{2.32}$$

we have:

$$F(\mathbf{x_0}) \geq F(\mathbf{x_1}) \geq F(\mathbf{x_2}) \geq \cdots \tag{2.33}$$

So hopefully the sequence $(\mathbf{x}_n)$ converges to the desired local minimum. Note that the value of the step size γ is allowed to change at every iteration. With certain assumptions on the function F and particular choices of γ. The following term:

$$\gamma_n = \frac{(\mathbf{x}_n - \mathbf{x}_{n-1})^T [\nabla F(\mathbf{x}_n) - \nabla F(\mathbf{x}_{n-1})]}{||\nabla F(\mathbf{x}_n) - \nabla F(\mathbf{x}_{n-1})]||^2} \tag{2.34}$$

convergence to a local minimum can be guaranteed. When the function F is convex, all local minima are also global minima, so in this case, gradient descent can converge to the global solution.

This process is illustrated in the adjacent picture (Figure 2.22). Here F is assumed to be defined on the plane, and that its graph has a bowl shape. The blue curves are the contour lines, that is, the regions on which the value of F is constant. A red arrow originating at a point shows the direction of the negative gradient at that point. Note that the (negative) gradient at a point is orthogonal to the contour line going through that point. We see that gradient descent leads us to the bottom of the bowl, that is, to the point where the value of the function F is minimal.

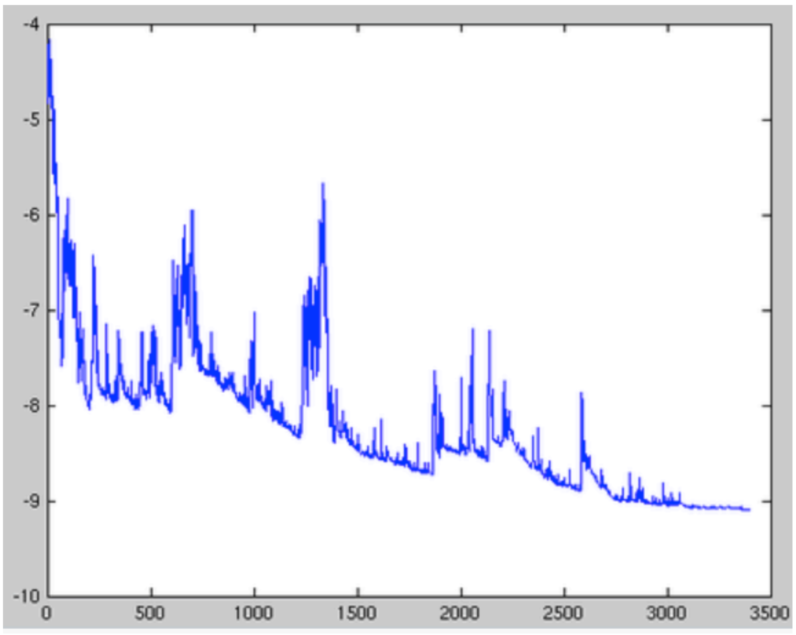


Figure 2.23 Illustration of SGD fluctuation

Stochastic Gradient Descent

Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example x_i and label y_i:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x_i; y_i) \tag{2.35}$$

batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online. SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily as in Figure 2.23. While batch gradient descent converges to the minimum of the basin the parameters are placed in, SGD's fluctuation, on the one hand, enables it to jump to new and potentially better local minima. On the other hand, this ultimately complicates convergence to the exact minimum, as SGD will keep overshooting. However, it has been shown that when we slowly decrease the learning rate, SGD shows the same convergence behavior as batch gradient descent (use a batch of samples in one iteration), almost certainly converging to a local or the global minimum for non-convex and convex optimization respectively.

Momentum

Further proposals include the momentum method, which appeared in Rumelhart, Hinton and Williams' seminal paper on backpropagation learning [46]. Stochastic gradient descent with momentum remembers the update Δw at each iteration, and determines the next update as a linear combination of the gradient and the previous update [50]:

$$\Delta w :=\alpha \Delta w - \eta \nabla Q_i(w) w :=w + \Delta w w :=w + \Delta w \tag{2.36}$$

which leads to:

$$w :=w - \eta \nabla Q_i(w) + \alpha \Delta w \tag{2.37}$$

where the parameter w which minimizes $Q(w)$ is to be estimated, and η is a step size (sometimes called the learning rate in machine learning).

The name momentum stems from an analogy to momentum in physics: the weight vec- tor w, thought of as a particle traveling through parameter space [19], incurs acceleration from the gradient of the loss ("force"). Unlike in classical stochastic gradient descent, it tends to keep traveling in the same direction, preventing oscillations. Momentum has been used successfully in various cases. Figure 3.23 describe the training process between SGD and SGD with momentum.

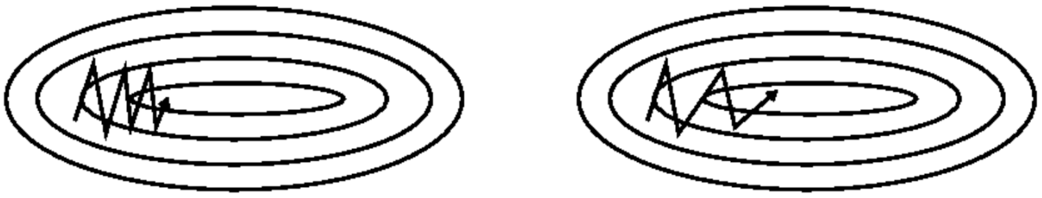


Figure 2.24 Trainging process between SGD (left), and SGD with momentum (right) [46]

SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily as in Figure 2.23. While batch gradient descent converges to the minimum of the basin the parameters are placed in, SGD's fluctuation, on the one hand, enables it to jump to new and potentially better local minima. On the other hand, this ultimately complicates convergence to the exact minimum, as SGD will keep overshooting. However, it has been shown that when we slowly decrease the learning rate, SGD shows the same convergence behavior as batch gradient descent (use a batch of samples in one iteration), almost certainly converging to a local or the global minimum for non-convex and convex optimization respectively.

AdaGrad

AdaGrad (for adaptive gradient algorithm) is a modified stochastic gradient descent with per-parameter learning rate, first published in 2011 [51]. Informally, this increases the learning rate for more sparse parameters and decreases the learning rate for less sparse ones. This strategy often improves convergence performance over standard stochastic gradient descent in settings where data is sparse and sparse parameters are more informative. Examples of such applications include natural language processing and image recognition. It still has a base learning rate η, but this is multiplied with the elements of a vector $G_{j,j}$ which is the diagonal of the outer product matrix.

$$G = \sum_{\tau=1}^{t} g_\tau g_\tau^T \tag{2.38}$$

where $g_\tau = \nabla Q_i(w)$. g_τ is the gradient at iteration τ. The diagonal is given by the following:

$$G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2 \tag{2.39}$$

This vector is updated after every iteration. The formula for an update is now become:

$$w: = w - \eta \operatorname{diag}(G)^{-\frac{1}{2}} \circ g \tag{2.40}$$

or, can be written as per-parameter updates:

$$w_j: = w_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j \tag{2.41}$$

Each $G_{i,i}$ gives rise to a scaling factor for the learning rate that applies to a single parameter w_i. Since the denominator in this factor, $\sqrt{G_i} = \sqrt{\sum_{\tau=1}^{t} g_\tau^2}$ is the L2 norm of previous derivatives, extreme parameter updates get dampened, while parameters that get few or small updates receive higher learning rates [52].

RMSProp

RMSProp (for Root Mean Square Propagation) is also a method in which the learning rate is adapted for each of the parameters. The idea is to divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight [53]. So, first the running average is calculated in terms of means square :

$$v(w,t) = \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2 \qquad (2.42)$$

where, γ is the forgetting factor. And the parameters are updated as the following:

$$w: = w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w) \qquad (2.43)$$

RMSProp has shown excellent adaptation of learning rate in different applications.

Adam

Adam (short for Adaptive Moment Estimation) [54] is an improved RMSProp optimizer. In this optimization algorithm, running averages of both the gradients and the second moments of the gradients are used. Given parameters w(t) and a loss function L(t), where t indexes the current training iteration (indexed at 1).

Chapter 3. Pose Estimation using Multi-Task Deep Learning Network

In this chapter, we introduce several pose (skeleton) estimation models using multi-task deep learning network. Section 3.1 first introduces our previous work on single-person image using in-store cameras [34]. In Section 3.2, we introduce Mask R-CNN [24], the basic idea of our proposed model, and the former method used for object detection and classification [25].

3.1. Single-Person Pose Estimation using In-Store Cameras

In our previous work, we built a pose estimation network for customer behavior analysis using in-store cameras. The customer pose, which includes the positions of joints, can provide rich information about the customer interest level to the merchandise. For example, the hand position could tell whether the customer is taking an item. The foot position can reveal the distance between the customer and merchandise shelf. Some special pose configurations can also represent the squatting and bending over behaviors. These are the reason of that this dissertation treats the customer pose estimation is an indispensable part of customer interest level assessment system.

However, the 2D human pose estimation is not an easy task in computer vision. One of the difficulties is the occlusion problem, including both self-occlusion and mutual-occlusion by other objects. The main reason of self-occlusion is the body orientation. For example, if a standing person is facing the right in the camera, his or her left body is probably occluded. On the other hand, the mutual-occlusions may happen due to arbitrary objects. In retail store environment, the mutual-occlusions are typically caused by the shopping baskets. Figure 3.1 shows the examples of the self-occlusion and the mutual-occlusion by shopping basket. Another difficulty of pose estimation is the left-right similarity problem because of the symmetry of human body. For example, the left shoulder of a person in the back view is very similar to the right shoulder in front view. Therefore, in order to solve these problems, this dissertation proposes to incorporate the pose estimation with body orientation, joint connections and joint visibility mask. These elements are tightly related no matter in physics or in the image.

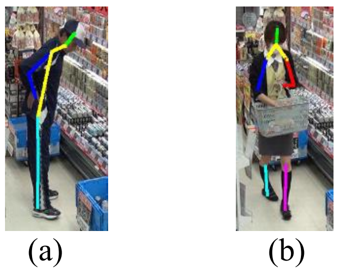


Figure 3.1 The examples of occlusions in the retail store. (a) The left body part is self-occluded. (b) The right wrist, left hip and right hip are occluded by the shopping basket

3.1.1 Body Orientation

The body orientation is a kind of global information of pose configuration. The body orientation is defined in 8 directions as Figure 3.2. The body orientation is useful for solving both self-occlusion problem and left-right similarity problem. For example, if one knows a person is facing to right, the body orientation indicates the occlusion of his or her left body. Similarly, if a person is facing the camera, his or her right shoulder is probably at the left side of image. Note that the orientation defined in Figure 2.3 is only applicable for some simple pose in the store, such as standing, walking, bending over or squatting down. For more complex poses, it is difficult to give a body orientation in the definition of 8 directions, such as the dancing, park over or doing gymnastics.

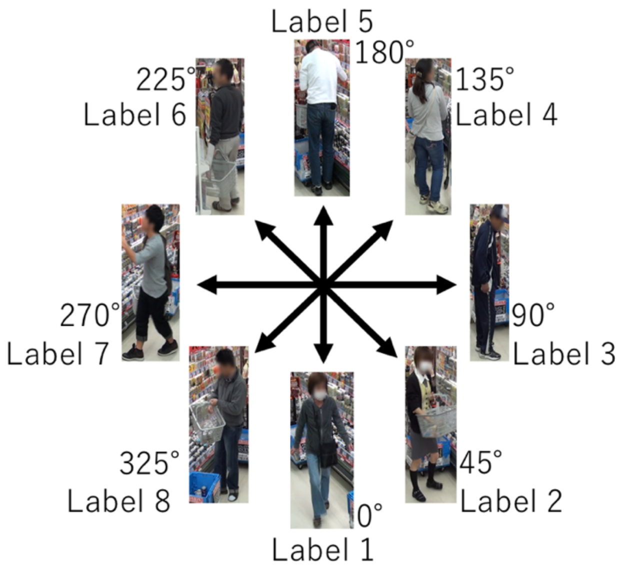


Figure 3.2 The representations of body orientations

3.1.2 Visibility Mask

The visibility mask is a binary vector where its element indicates the visibility of each joint. The term of visibility mask is introduced by Haque et al [55]. However, they only applied the visibility mask in the top view images for self-occlusions. This thesis extends the application of visibility mask for more flexible view angles and for both self-occlusion and the mutual-occlusion by objects. Other approaches also used a binary vector to represent the occlusion. Although these approaches modeled the occlusions in different ways, they did not consider the relationship between the joint occlusion and body orientation. This thesis argues that the body orientation is an indispensable part of the occlusion model. Additionally, we combines the occluding object detection in the system. It is an apparent clue of mutual-occlusion for visibility mask prediction.

3.1.3 Deep Joint Position Learning

In recent year, some outstanding deep neural networks are proposed for classification tasks. For pose estimation, a trivial method is applying these networks to directly produce the output of joint positions and visibility mask. The framework of this method is shown as Figure 3.3. The ResNet is modified to adapt the output of joint positions and visibility mask. Firstly, the original softmax layer at the output end of ResNet is removed. Secondly, the last FC layer is modified to output a vector with length 56. This vector contains two parts of information. One is the x, y coordinates of 14 joints, whose length is 14 × 2 = 28. Another is the [visibility, invisibility] scores of each joint, whose length is also 14 × 2 = 28. The [visibility, invisibility] scores are given into an additional softmax layer to produce the probabilities of visibility and invisibility. For the ground truth visibility mask, the pair of [visibility, invisibility] is [1, 0] for visible joints and [0, 1] for occluded joints. Therefore, the filter size of last FC layer is modified t from 1 × 1 × 2048 × 1000 to 1 × 1 × 2048 × 56.

After constructing the network, the inputs and ground truth can be defined. Assume the input image is I and the pose vector is $h = (\ldots, h_i, \ldots)$i ∈ {1,...,14}, where h_i contains the x and y coordinates (x_i, y_i) of the ith joints. Because the ResNet requires the input image should be a square with a smallest size of size s = 224, the input image and pose are resized as follows, where I_r is the resized image, p' is the normalized the pose vector, b_w and b_h are the width and height of original input image, $b_c \in \mathbb{R}^2$ is the center position of original input image, p' is the label in learning process.

$$I_r = resize(I, s) \tag{3.1}$$

$$p' = \begin{pmatrix} s/b_w & 1 \\ 1 & s/b_h \end{pmatrix} (h - b_c) \tag{3.2}$$

Assuming p is the predicted pose of network, the final predicted pose in original image size is calculated as below.

$$p_{final} = \begin{pmatrix} b_w/s & 1 \\ 1 & b_h/s \end{pmatrix} (p + b_c) \tag{3.3}$$

Then the loss of prediction can be calculated. The pose loss and gradient are calculated by Mean Square Error (MSE) between predicted pose p and the normalized true pose p'. The loss and its gradient are shown as below, where L_p is loss of pose and e_p is the gradient of loss, N is the number of keypoints (14 in this method).

$$L_p = \frac{1}{2N}\sum_{i=1}^{N}\|p_i - p_i'\|_2^2 \tag{3.4}$$
$$e = p - p' \tag{3.5}$$

The loss of visibility mask is calculated by the Cross Entropy. The loss and its gradient are shown as below, where L_v and e_v is visibility loss and its gradient, v and v' is the predicted and true visibility vector respectively.

$$L_v = -\sum_{i=1}^{2N} v_i \log v_i' \tag{3.6}$$
$$e_v = v - v' \tag{3.7}$$

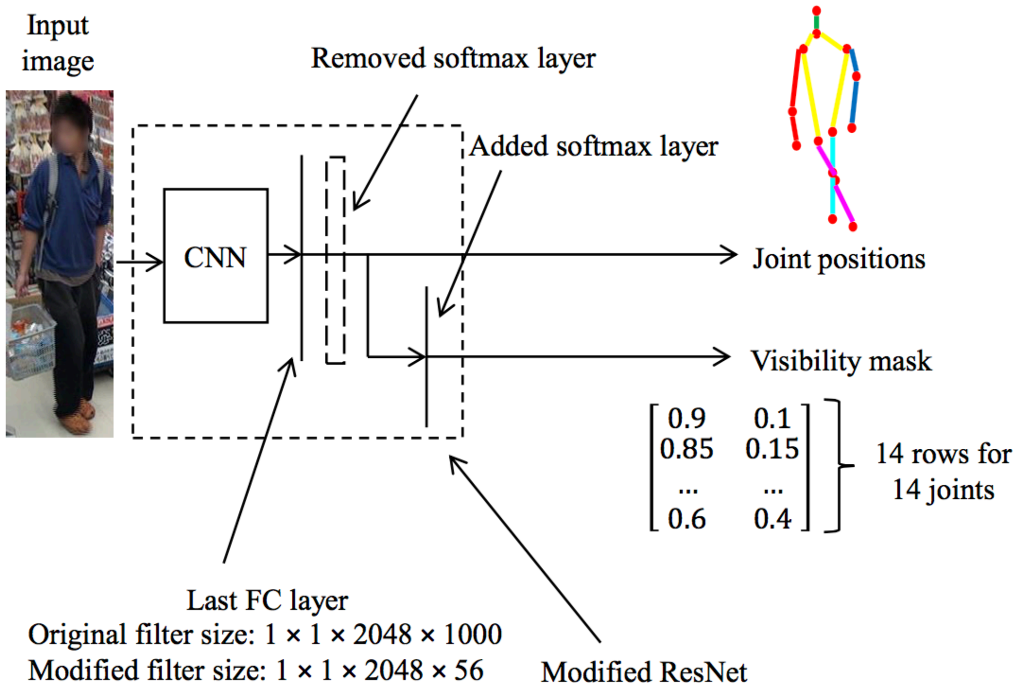


Figure 3.3 The framework of joint position estimation based on ResNet

This network can be also improved by introducing the body orientation as Figure 3.4. In Figure 3.4, the last FC layer not only receives the output of previous layer, but also the body orientation vector. The filter size of the last FC layer also becomes 1 × 1 × 2048 × 56 × 8.

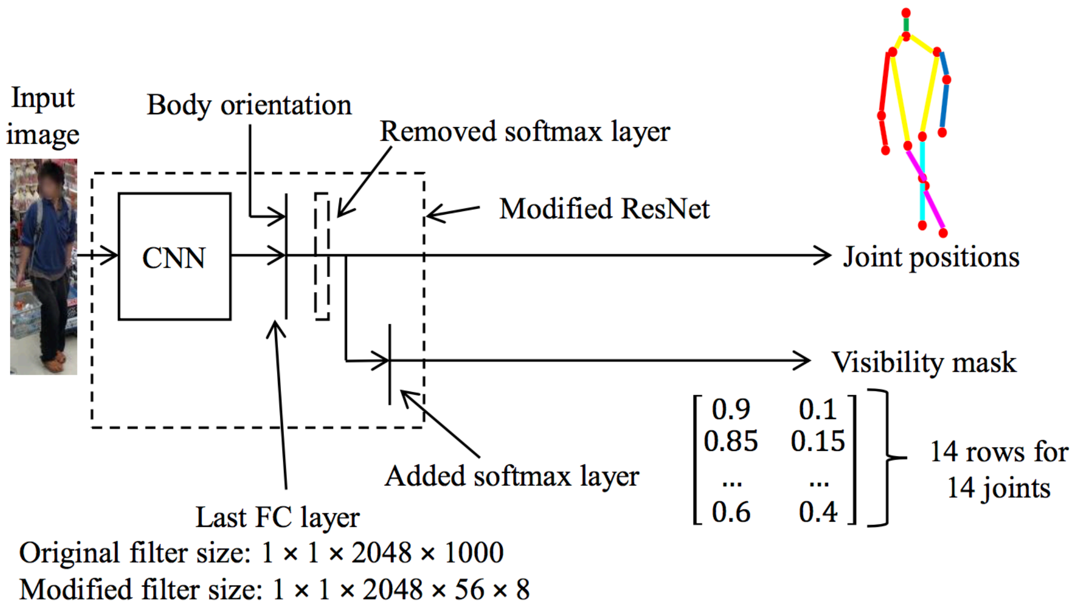


Figure 3.4 The framework of joint position estimation based on ResNet combining body orientation

3.1.4 Deep Joint Heatmap Learning

In Section 3.1.3, we introduce a method directly predicts joint position. However, there is a limitation in these trivial deep learning method: they only give one pose prediction per image. Even though the elbow positions are relatively accurate, it is difficult to correct the hand positions based on elbow positions, because the single prediction without any confidence information or candidates could not provide any extra information. One solution is to generate joint probabilistic heatmaps. Therefore, this Section will propose a joint heatmap estimation using deep learning.

The main idea behind the joint heatmap is to increase the prediction candidates. Based on the heatmap, many flexible correction methods could be applied. For example, a set of message-passing layer can correct the joint heatmap by modeling the relationship between neighboring joints. There are many methods to generate the heatmaps. One kind of methods is to trained the CNN with local image patches and tested by sliding window to generate the joint heatmaps [56] [57]. That means these methods convert the joint estimation problem to a pixel-wise classification problem. In these pixel-wise classification methods, the training and testing are time consuming. Besides, the negative patches are far more than the positive patches in the image, the number of negative patches usually need to be manually controlled. To cover this shortage, another kind of methods is to extend the popular CNN, such as GoogLeNet or VGG, to a Full Convolutional Network (FCN). The mainly different between FCN and conventional CNN is that FCN replaces the Fully Connected (FC) layer in CNN to a 1 × 1 size convolution layer. Therefore, FCN can receive the whole image as input, rather than the local image patch, to directly generate a feature map. According to different applications, the FCN could be trained to generate different semantic heatmaps.

Considering the good performance of FCN-8s in [58], this chapter will propose the joint heatmap learning based on FCN-8s. The overall framework is shown as Figure 3.5.

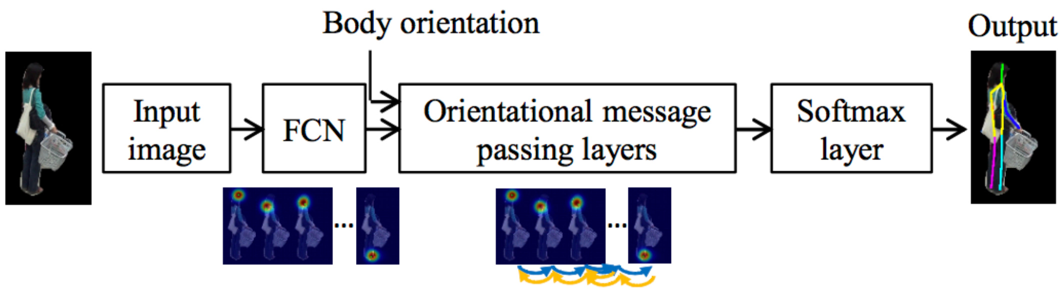


Figure 3.5 The framework of deep joint heatmap learning

The customer image is given into a Fully Convolutional Networks (FCN) to generate initial joint heatmaps, which is shown in Figure 3.6. Based on these heatmaps and body orientation input, a set of Orientational Message Passing (OMP) layers are constructed. These layers model the global pose configuration from orientation and the local neighboring joint connections from the generated heatmaps. Finally, a softmax layer is used to normalize the heatmaps to probabilistic distribution. The output joint position is the centroid of each joint heatmap.

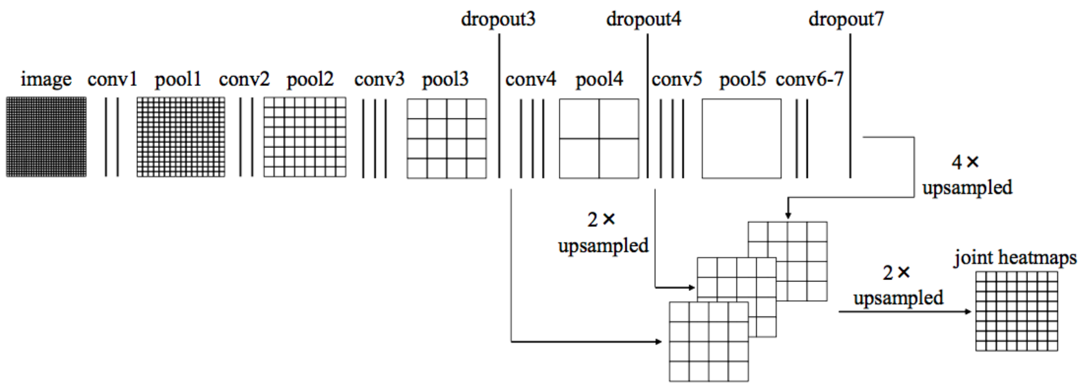


Figure 3.6 The structure of modified FCN-8s for pose estimation [58]

3.2. Mask R-CNN

In this Section, we will introduce the details of Mask R-CNN, which is the basic method of our proposed method. Mask R-CNN is a famous multi-task deep learning network improved on the Faster R-CNN. Section 3.2.1 first introduces the concepts of Faster R-CNN. Then the improvement of Mask R-CNN will be discussed in Section 3.2.2.

3.2.1 Faster R-CNN

Faster R-CNN starts with R-CNN [59] proposed by R.Girshick et al. for the puropose to bypass the problem of selecting a huge number of regions in object detection task. Fig 3.7 shows the object detection process of R-CNN. They first used selective search algorithm [60] to extract around 2000 bottom-up region proposals from each input image. Then each region proposal is warped into the same size and pass a deep CNN. And then using CNN fully connected layer output as the features to classify each region. They used linear SVMs as the classifier. In the end, using bounding-box regression on classified regions to get a better bounding-box.

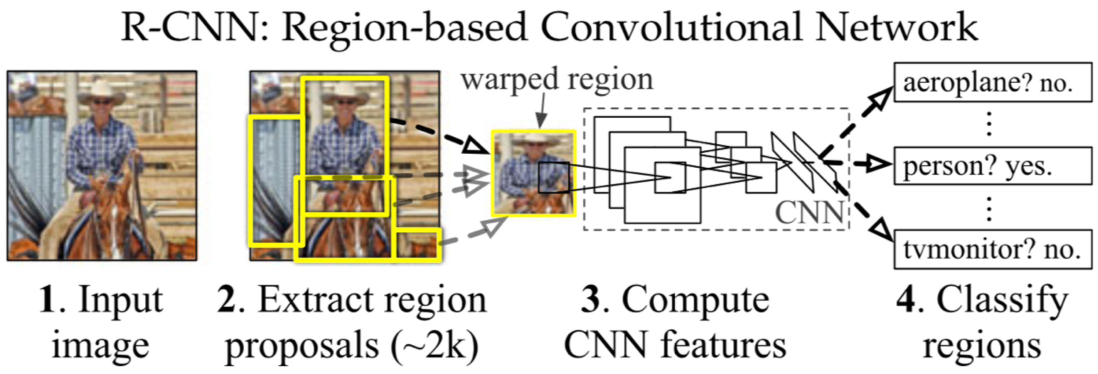


Figure 3.7 R-CNN object detection system overview [59]

R-CNN achieves a mean average precision (mAP) of 66.0 percent in PASCAL VOC object detection [61], it improves mAP by more than 50 percent relative to the previous best result. However, there are many problems with the R-CNN framework: 1. Training of R-CNN is divided into multiple stages, training process is very tedious. 2. The training process is time-consuming and need a lot of processing memory. 3. The speed object detection is not satisfying.

The main reason why R-CNN is time-consuming is that it uses a large CNN network to process every proposed region, without sharing computation. It means when there are 2000 region proposals, it has to repeat 2000 times forward pass, which wastes lots of time. In fact, many region proposals overlap each other, and those overlapped parts are fed into the CNN for many times. In order to solve this problem by sharing the computation between proposals, Fast R-CNN [20] was proposed. Fig 3.8 shows the illustration of Fast R-CNN.

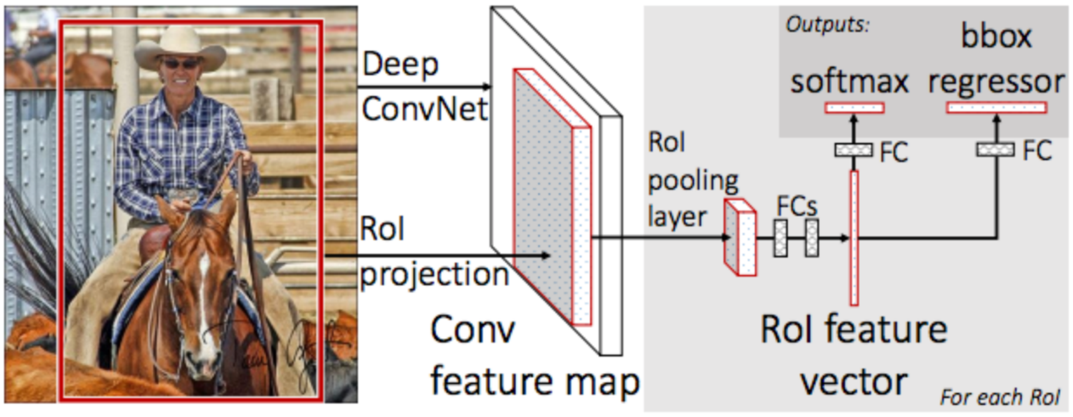


Figure 3.8 The architecture of Fast R-CNN [20]

In Fast R-CNN, instead of feeding the region proposals to the CNN, they feed the input images to the CNN to generate a set of convolutional feature maps. From the convolutional feature maps, they identify the region of proposals and warp them into squares and by using an RoI pooling layer that reshapes them into a fixed size so that they can be fed into a fully connected layer. From the RoI feature vector, they use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box. The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets.

Fast R-CNN achieves the top result on VOC2012 with a mAP of 65.7% (and 68.4% with extra data). And it is much faster than R-CNN, when using VGG16 as backbone feature extraction network, Fast-RCNN can process an image in 3 seconds.

However, Fast R-CNN is still not fast enough for the real-time application. Both of the above algorithms (R-CNN and Fast R-CNN) use selective search to find out the region proposals, where selective search is a slow and time-consuming process affecting the performance of the network. It takes up 90% of the processing time. Therefore, Ren et al. proposed Faster-RCNN, an object detection algorithm that eliminates the selective search algorithm and lets the network learn the region proposals. In Faster R-CNN, they replace selective search algorithm with a Region Proposal Network (RPN) to proposes regions. As shown in Fig. 3.9, the entire system is a single, unified network for object detection.

Faster R-CNN is composed of two modules. The first module is RPN, which is a deep fully convolutional network for region proposal, and the second module is the Fast R-CNN detector that uses the proposed regions. Using the recently popular terminology of neural networks with attention [62] mechanisms, the RPN module tells the Fast R-CNN module where to look. As shown in Fig 3.10, an RPN takes an image feature map as input and outputs a set of rectangular object proposals (coordinates), each with an object score (objects or background). k denotes the number of anchor boxes. So the regression layer has $4k$ outputs encoding the coordinates of k boxes, and the cls layer outputs $2k$ scores that estimate the probability of object or not object for each proposal.

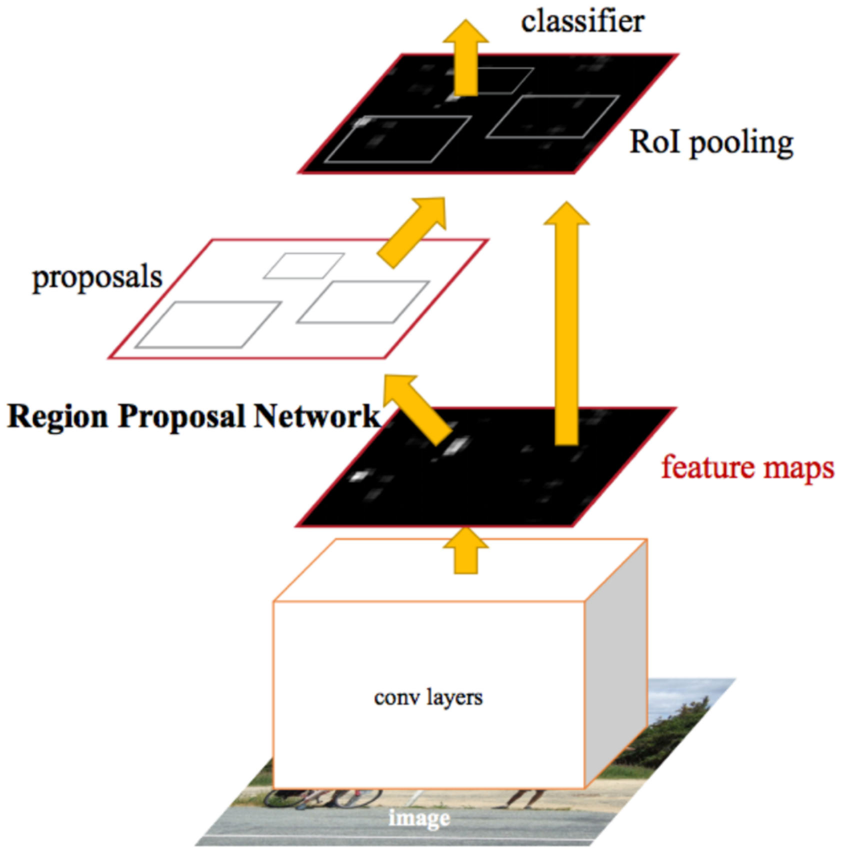


Figure 3.9 An illustration of the concepts in Faster R-CNN

For training RPN, they used an objective function following the multi-task loss in Fast R-CNN. The loss function for an image is defined as:

$$\mathcal{L}(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i \mathcal{L}_{cls}(p_i,p_i^*) + \lambda \frac{1}{N_{reg}}\sum_i p_i^* \mathcal{L}_{reg}(t_i,t_i^*) \qquad (3.8)$$

where i is the index of an anchor in a mini-batch and p_i is the predicted probability of anchor i being an object. p_i^* denotes the ground-truth label. p_i^* is 1 when the anchor is positive, otherwise it is 0. t_i denotes the the 4 parameterized coordinates of the predicted bounding box, and t_i^* is the ground-truth box associated with a positive anchor. $\mathcal{L}_{cls}$ is classification loss over two classes(object and background). For the regression loss $\mathcal{L}_{reg}$, they used $\mathcal{L}_{reg}(t_i,t_i^*) = R(t_i - t_i^*)$ where R is the robust loss function (smooth $\mathcal{L}_1$). The term $p_i^* \mathcal{L}_{reg}$ means the regression loss is activated only for positive anchors $(p_i^* = 1)$ and is disabled otherwise.

Faster-RCNN proposed RPN for efficient and accurate region proposal generation. Using this method, region proposal step became nearly cost-free. At the same time, the learned RPN also improves region proposal quality and thus the overall object detection accuracy.

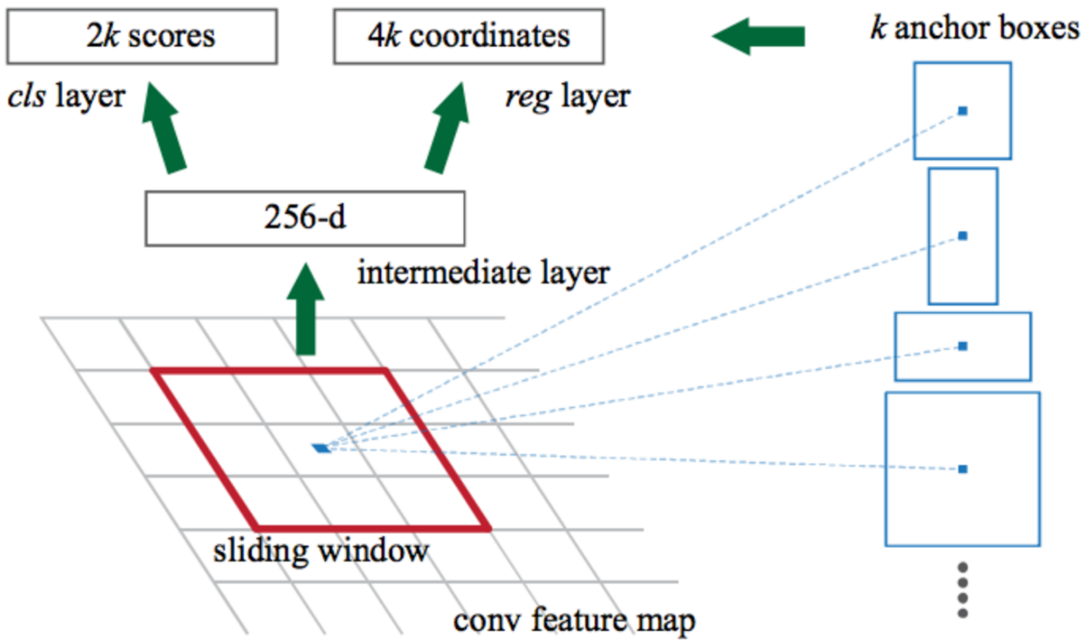


Figure 3.10 A illustration shows how Region Proposal Network (RPN) generate anchors

3.2.2 RoI Align Layer in Mask R-CNN

As we discussed in Section 3.2.2, Faster R-CNN achieves faster objection detection and classification by changing selective search algorithm into Region Proposal Network (RPN). This method was later developed into Mask R-CNN, which is a multi-task network used for semantic segmentation (shown in Figure 3.12).

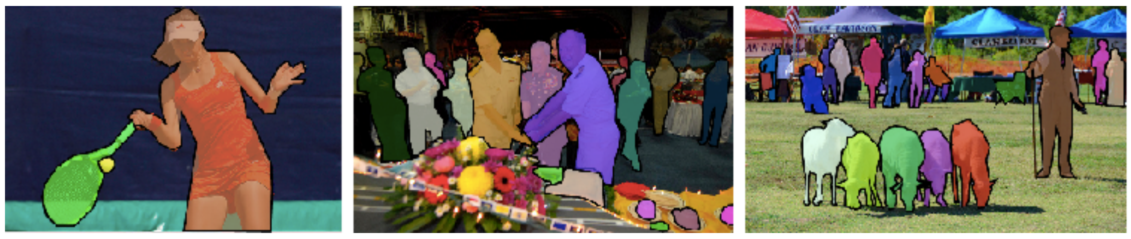


Figure 3.11 Examples of instance level segmentation in COCO dataset [63]

In Section 3.2.2, we also mentioned that Faster R-CNN uses an RoI pooling layer to reshape the features extracted from backbone network into a fixed size so that they can be fed into a fully connected layer. In this reshape transformation, the feature maps are scaling into a smaller size, which means some information in original scale may be collapsed during the pooling. This did not become a problem when dealing with objection detection and classification tasks because the fully connected layer can be seen as an encoder, where the input will be further compressed to output vector prediction. However, when it comes to segmentation task, the situation becomes different. In order to generate segmentation results from small size feature maps, an up-sampling layer that recovers the features into larger size is necessary. This process can be seen as a decoder. If we use the features reshaped by the RoI pooling layer, because the information in original scale is partly cropped, the segmentation output will be very rough. This mis-alignment problem might decrease the segmentation accuracy.

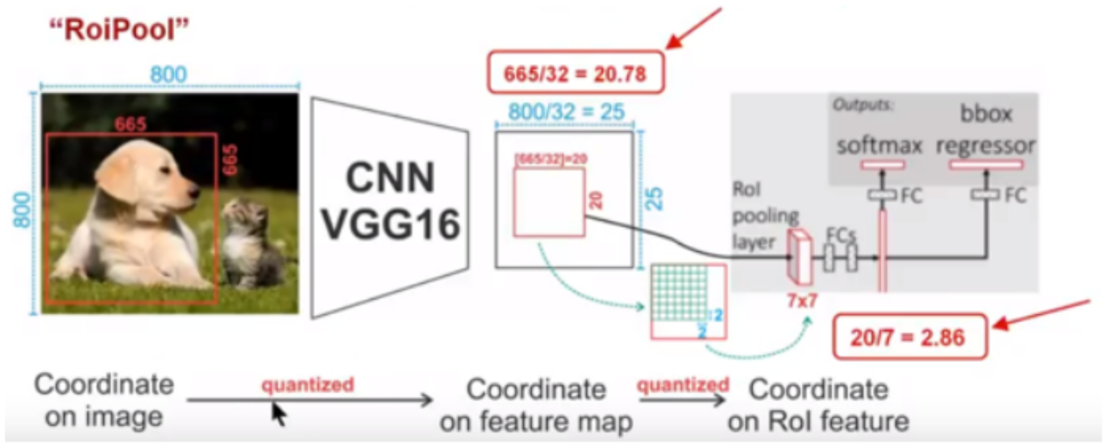


Figure 3.12 An illustration of the misalignment problem of RoI pooling

Figure 3.13 shows an example of the misalignment problem in RoI pooling layer. In this Faster R-CNN detection framework, the input is an 800*800 image with a 665*665 bounding box of dog. After passing the feature extraction backbone network, the feature maps are cropped with a stride of 32. As a result, the size of input image and bounding box are reshaped into the 1/32 of original size. In this example, the input image size 800 can be exact divided by 32 into 25, while the bounding box size 665 turns into 20.78 when divided by 32, which will be rounded down to 20. Then the feature maps are input into the RoI pooling layer, where they are further reshaped into size 7*7. This means the bounding box will be divided averagely into 49 square areas. Then each square area has a size of 2.86, which will be rounded down to 2. At this step, the error of candidate area has been very obvious (shown with the green part in Figure 3.13). More importantly, the pixel error in feature map scale will be zoomed out 32 times in original scale, which means the 0.86 pixel error in this case will lead to a 30 pixel error in the original image.

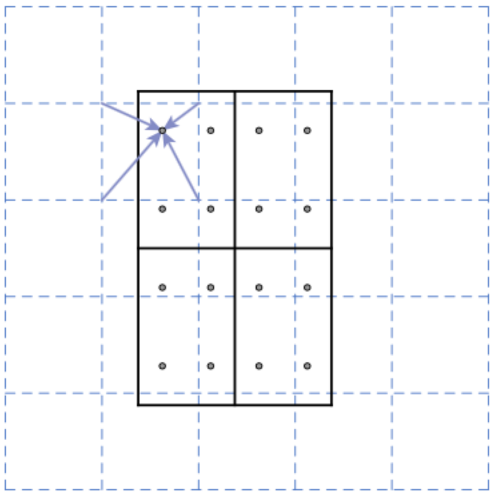


Figure 3.13 Description of RoI Align layer using bilinear interpolation [25]

In order to make accuracy pixel-wise segmentation, the RoI pooling layer need to be changed. In Mask R-CNN, they propose a new architecture called RoI Align layer to solve this problem. RoI Align layer canceled the quantization in RoI pooling layer and uses bilinear interpolation instead of maxpooling. Figure 3.14 roughly describes the main concept of RoI Align layer, the dashed grid represents a feature map, the solid lines an RoI (with 2×2 bins in this example), and the dots the 4 sampling points in each bin. RoIAlign computes the value of each sampling point by bilinear interpolation from the nearby grid points on the feature map. No quantization is performed on any coordinates involved in the RoI, its bins, or the sampling points.

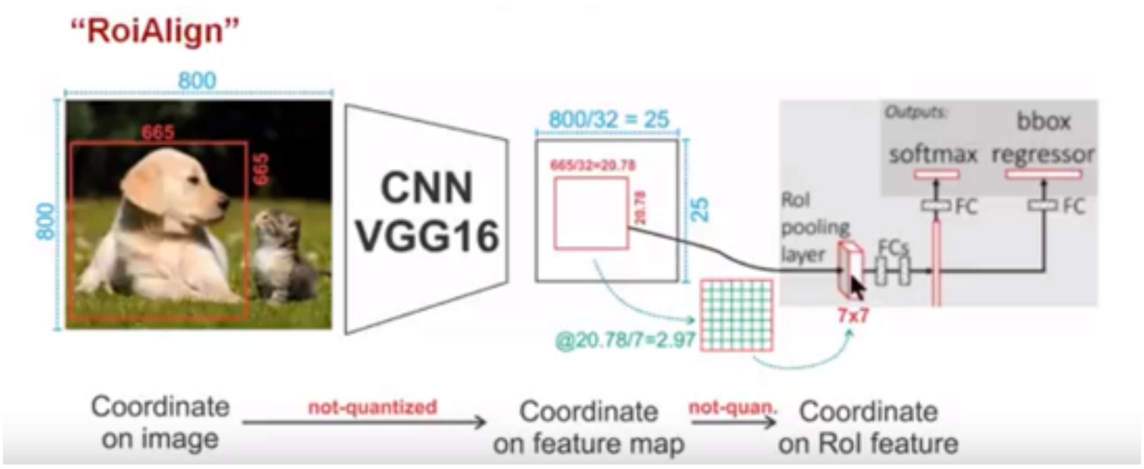


Figure 3.14 How RoI Align layer solves the misalignment problem

Figure 3.15 shows how RoI Align layer solves the misalignment problem using the same example as Figure 3.11. In feature map level, the bounding box with size of 665*665 reshaped into size 20.78*20.78 without quantization. When it comes to RoI features, the size becomes 20.78/7 = 2.97. And by using bilinear interpolation, the value of each pxiel in the 7*7 RoI feature is computed. As a result, the misalignment problem of RoI pooling layer was solved. Figure 3.16 shows how Mask R-CNN predicts accurate segmentation using RoI Align layer.

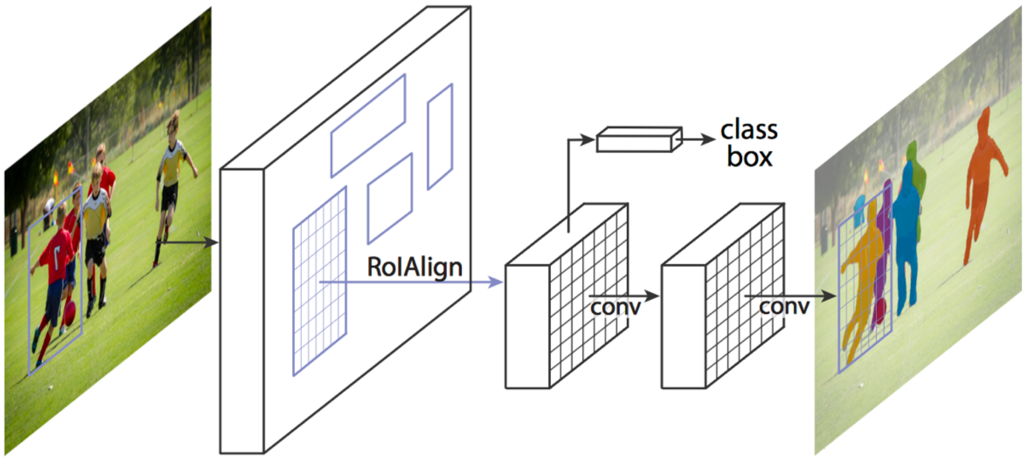


Figure 3.15 Overview of Mask R-CNN using RoI Align [25]

3.2.3 Feature Pyramid Network (RPN) in Mask R-CNN

Besides the RoI Align layer, Mask R-CNN also improves the backbone network and feature processing architecture of Faster R-CNN. In Mask R-CNN, the backbone network was changed into ResNet [13]. And for feature processing at the end of backbone network, they explore another more effective backbone proposed by Lin et al. [64], called Feature Pyramid Network (FPN). FPN uses a top-down architecture with lateral connections to build an in-network feature pyramid from a single-scale input.

The forward passing of CNN is a top-down pathway, usually the feature map becomes smaller during the computation of convolutional kernel. However, there are some feature layer outputs the same size features as the input features, which is defined as same network stage. In the method of Feature Pyramid Network, they defined each same network stage in the backbone architecture as one level of the feature pyramid, then select the output of the last layer of each stage as the reference of feature maps, which is reasonable because the deepest layer of each stage should have the most activated features. In the paper, they choose ResNet as backbone network and build a feature pyramid consists of $\{C_2, C_3, C_4, C_5\}$, which are the output of conv2, conv3, conv4, conv5 layer of the ResNet and have strides of {4,8,16,32} on the input image. The conv1 layer is not included in the pyramid because of the capacity of GPU memory.

In order to combine features with different resolution in the top-down pathway, FPN first uses up-sampling on the high-level feature maps which has stronger semantic information and are more abstract, and then lateral connects these features with the former level features to reinforce the high level features. Figure 3.17 shows details of the lateral connections of FPN. High level features are 2x up-sampled using nearest up-sampling. And the features from former layer pass a convolutional layer with kernel size 1 for changing feature depth into the same size as high level features (in this paper, the depth $d = 256$). These two set of features are combined together using pixel-wise add operation. This operation is repeated until all levels of the feature pyramid $\{C_2, C_3, C_4, C_5\}$ are added to generate the most detailed feature map $\{P_2, P_3, P_4, P_5\}$. Before these features are input into next network, $\{P_2, P_3, P_4, P_5\}$ will pass a convolutional network with kernel size 3 to erase the aliasing effect caused by up-sampling.

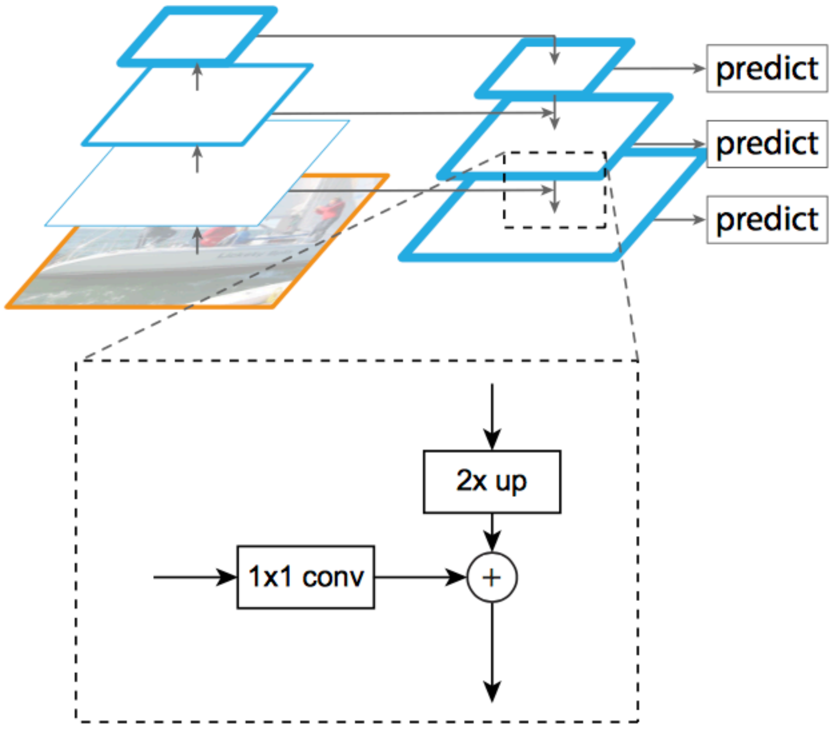


Figure 3.16 A building block illustrating the lateral connection and top-down pathway [64]

3.2.4 Mask R-CNN for Human Pose Estimation

As RoI Align layer makes pixel-wise segmentation possible, Mask R-CNN can be also extended for the human pose estimation task. They model a keypoint's location as a one-hot mask, and adopt Mask R-CNN to predict K masks, one for each of K keypoint types.

We borrow the idea of this task to build our proposed model, the detail of the implementation will be discussed in Section 4 together with other tasks.

Chapter 4.

Self and Mutual Occlusion Aware Multi-Person Skeleton Estimation using Multi-Task Deep Learning Network

In this section, we will introduce our proposed multi-task skeleton estimation network. In section 4.1, we will introduce the whole system framework of our proposed model, and in section 4.2, we will discuss how we design our basic parallel multi-task model. Section 4.3 shows two architectures of our further-designed serial model. And the experiment results are illustrated in section 4.4.

4.1. System Framework

As we discussed in Section1 and Section3. For pose (skeleton) estimation, most of the deep learning approaches mainly focus on the joint feature. However, this feature is not sufficient, especially when the joint is occluded or not intact. In fact, many features other than joint can also contribute to skeleton estimation, such as body boundary, body orientation and occlusion condition. By adopting multi-task strategy, these features can be efficiently combined inside deep learning model for skeleton estimation.

When we begin to build our multi-task skeleton estimation model, we choose Mask R-CNN as a basic model because the top-down RPN network makes it feasible to implement new task into the multi-task and the pose estimation task of Mask R-CNN also has a lot of space to improve.

We add body boundary, body orientation and occlusion condition in order to solve the occlusion problem, which is a general problem in multi-person pose (skeleton) estimation. The occlusion in multi-person case can be separated into two classes: self-occlusion and mutual-occlusion (shown in Figure 4.1). The self-occlusion is discussed in Section 3.1, where some joints disappear from the image when the person does not face the camera straight. Self-occlusion is general in single-person cases, however when it comes to multi-person situations, mutual-occlusion also happens frequently, where joints disappear when two people overlap each other or partly cropped by the image boundary.

In our proposed model, we introduce body orientation to deal with self-occlusion, (mutual) occlusion mask and body segmentation to deal with mutual-occlusion, and joint visibility mask to deal with both of them.

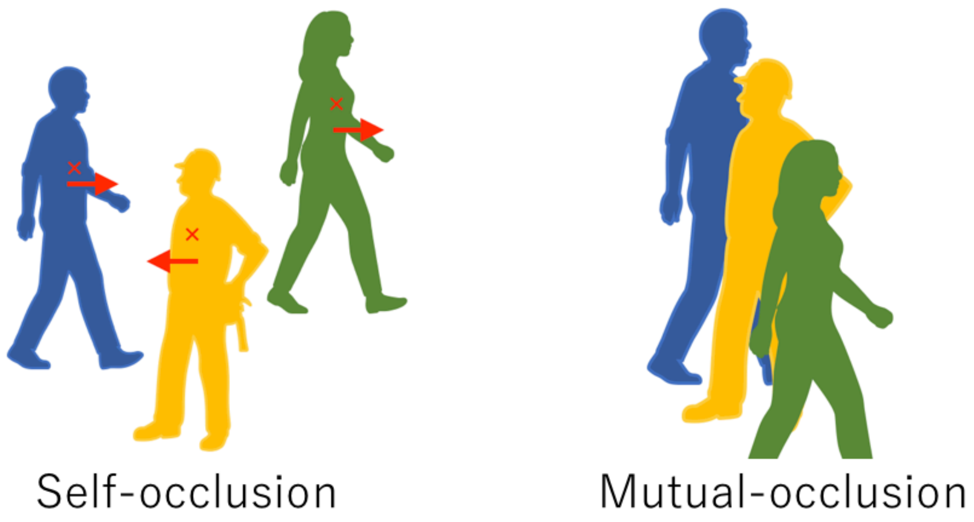


Figure 4.1 Illustration of occlusion problems in multi-person pose (skeleton) estimation

The system framework of our proposed method is illustrated in Figure 4.2. The input image will first be resized to 1024*1024 in the preprocessing step, then input into a Mask R-CNN layer heads for feature extraction, which consists of Feature Pyramid Network, Region Proposal Network and an RoI Align layer at last to generate RoI features. Notice that the two-stage object detection and classification branch in original Mask R-CNN is also done in this layer heads, which is not illustrated in Figure 4.2. The RoI features are then fed into our multi-task training network. We will show the details of this part in the next section.

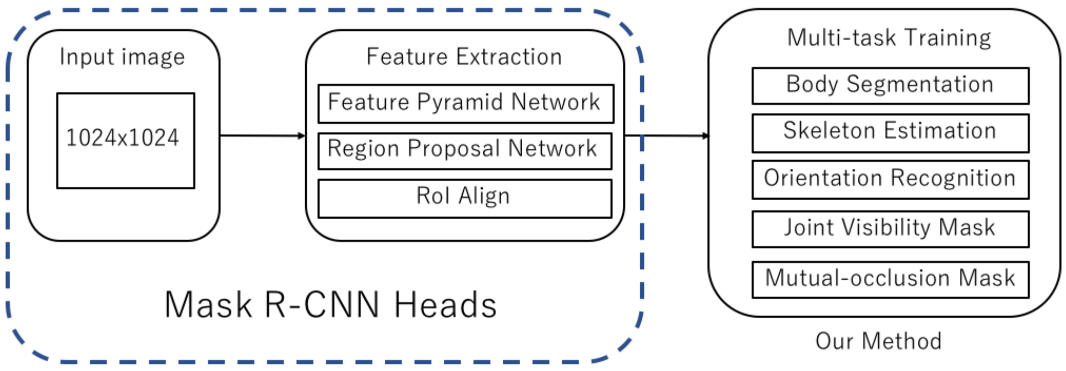


Figure 4.2 The framework of our multi-task skeleton estimation model

4.2. Proposed Parallel Multi-Task Skeleton Estimation Network

We first tried a general architecture of multi-task network that trains each task paralleled, which has been proved to work well in the paper of Mask R-CNN. The architecture is illustrated in Figure 4.3, the network has three branches (body segmentation branch, joint position estimation branch and occlusion-orientation branch) to output five multi-task results:
$\{Segmentation, Joint\ Heatmap, Orientation, Joint\ Visibility, Mutual\ Occlusion\}$

In the training step, each output of the multi-task results will be fed into a loss layer with corresponding ground truth label using different loss function specialized for each task. The multi-task loss L is defined as the sum of the loss from each task:

$$L = L_{segm} + L_{joint} + L_{ori} + L_{vis} + L_{occ} \tag{4.1}$$

From the next section, we will introduce the structure of each branch in our proposed method and how we define the loss function.

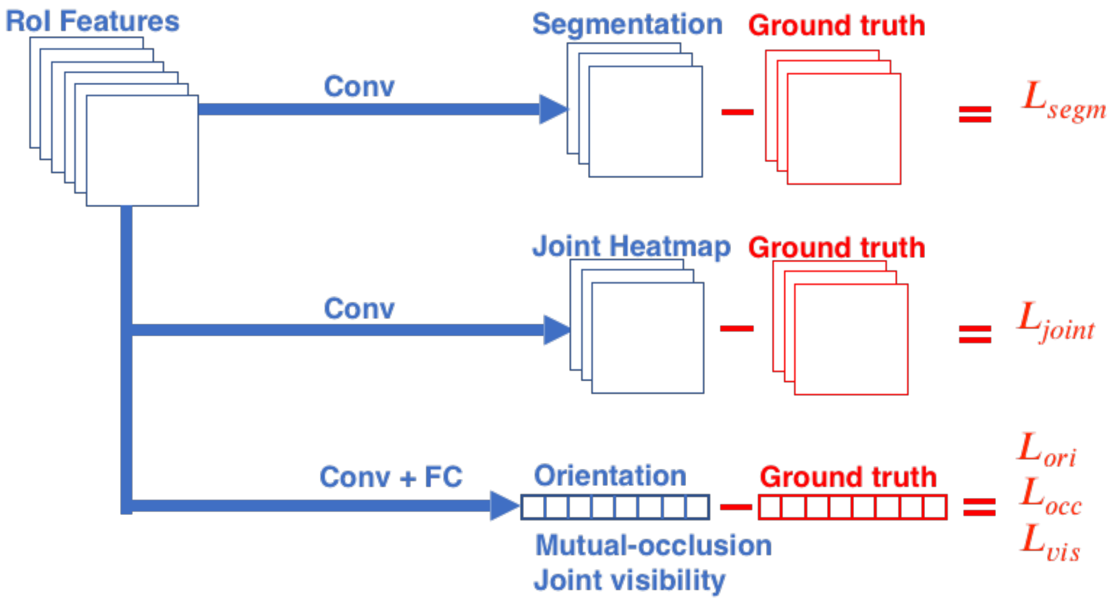


Figure 4.3 Architecture of our parallel multi-task training network

4.2.1 Body Segmentation Branch

In the body segmentation branch of our proposed method, we borrow the segmentation part of Mask R-CNN and change the detection classes into 2 (person and background). The architecture of this branch is illustrated in Figure 4.4. We predict an $m \times m$ mask from each RoI using an FCN [58], where the 14*14 size RoI features first pass a segmentation head consists of a stack of four convolutional layers with kernel size 3 and depth 256, followed by a 2× deconv layer and a final convolutional layer with kernel size 3, producing an output resolution of 28×28 with depth 2. This FCN structure allows each layer in the mask branch to maintain the explicit $m \times m$ object spatial layout without collapsing it into a vector representation that lacks spatial dimensions. This process can be simply expressed as:

$$Segm = \mathrm{conv}_{kernel=3}^{depth=2}\{\mathrm{deconv}_{kernel=2}^{depth=256}[\mathrm{conv}_{kernel=3}^{depth=256}(F)\ ]^4\} \qquad (4.2)$$

where F is the RoI features, $conv$ and $deconv$ represent the convolution and deconvolution operations.

The segmentation loss L_{seg} is calculated between ground truth and network output with the right class given for each RoI using cross entropy:

$$L_{segm} = -\frac{1}{N_{RoI}}\sum_{i=1}^{N_{RoI}}(Segm_i * class) * \log Segm_i' \qquad (4.3)$$

where i is the index of RoIs, $class$ is the predicted class of the object in each RoI (person or background).

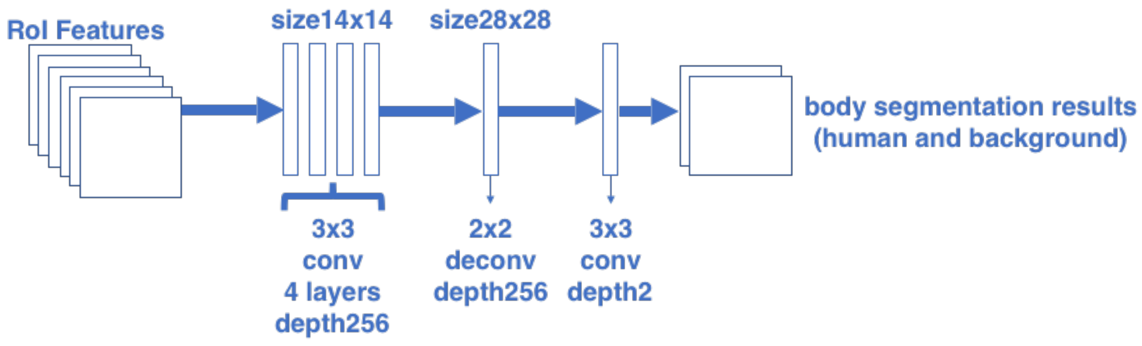


Figure 4.4 Architecture of the body segmentation branch

4.2.2 Joint Position Estimation Branch

In the joint position estimation branch we model the joint position as a one-hot mask, and use the architecture similar to the body segmentation branch to predict K masks, one for each of K joint types (e.g., left shoulder, right elbow). The architecture is shown in Figure 4.5, similar to the body segmentation task, we use FCN structure to output these K masks of each joint. For the reason that joint is much smaller in each RoI, the output joint mask size is set twice as the segmentation mask in order to give accurate results. The 14*14 RoI features first pass a stack of eight convolutional layers with kernel size 3 and depth 256, followed by a deconvolutional layer with kernel size 2 and a 2x bilinear interpolation up-scaling layer, producing an output resolution of 56×56 with depth K ($K = 17$). The expression of this branch is:

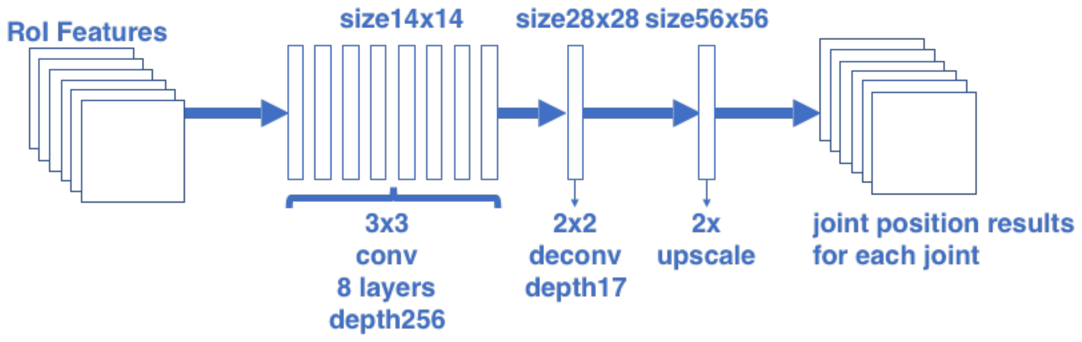


Figure 4.5 Architecture of the joint position estimation branch

$$J = Joint_{pos} = \text{upscaling}\{\text{deconv}_{kernel=2}^{depth=17}[\text{conv}_{kernel=3}^{depth=256}(F)\ ]^{8}\} \qquad (4.4)$$

The joint position loss L_{joint} is calculated between ground truth and network output with each joint for the person in each RoI. Instead of the pixel-wise cross entropy used in body segmentation branch, we use one-hot label ground truth where only one correct joint position is true, turning the loss into a classification loss over $m \times m$ ways:

$$L_{joint} = -\frac{1}{17N_{ROI}}\sum_{i=1}^{N_{ROI}}\sum_{j=1}^{N_{joint}=17}\text{Softmax}(J_{ij}) * \text{OneHot}(\log J'_{ij}) \qquad (4.5)$$

where the number of joint N_{joint} is set to 17. This loss function can force the network to output a probability contribution where only one pixel in the mask give the peak value after softmax operation, which shows better converge speed than the regression loss and heatmap MSE loss.

4.2.3 Occlusion and Orientation Branch

In the occlusion and orientation branch, information related to the self-occlusion and mutual-occlusion problem is predicted as vector outputs. Similar to the classification and detection network of Mask R-CNN, we also use fully connected network to generate vector. Considering the estimation of joint visibility is related to the joint position estimation, we change the size of RoI features from 7 into 14 (same as joint position estimation). The architecture of this branch is shown in Figure 4.6, which can be expressed as:

$$Occlusion\&Orientation = \text{FullyConnection}_{1024}[\text{conv}_{kernel=14}^{depth=1024}(F)\ ]\} \quad (4.6)$$

where $Occlusion\&Orientation$ includes the joint visibility mask (length 17 vector), body orientation (length 9 vector) and mutual-occlusion mask (length 2 vector).

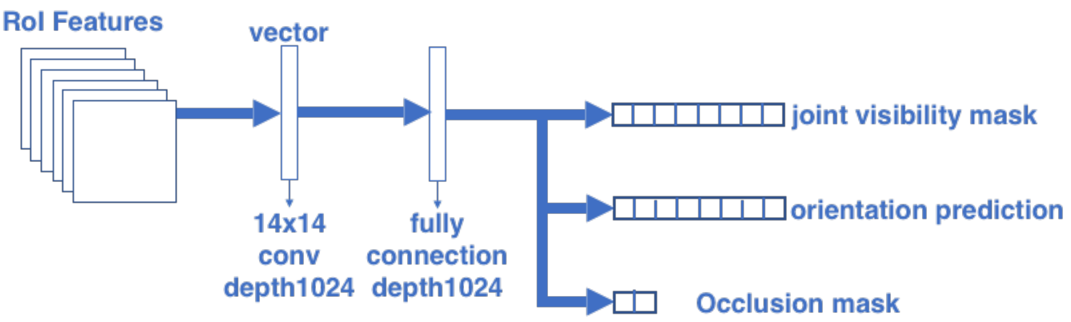


Figure 4.6 Architecture of the orientation-occlusion branch

The loss of these 3 outputs are calculated separately with cross entropy with softmax:

$$L_{ori} = -\frac{1}{N_{ROI}}\sum_{i=1}^{N_{ROI}} \mathrm{Softmax}(Ori_i) * \log Ori_i' \tag{4.7}$$

$$L_{vis} = -\frac{1}{N_{ROI}}\sum_{i=1}^{N_{ROI}} \mathrm{Softmax}(Vis_i) * \log Vis_i' \tag{4.8}$$

$$L_{occ} = -\frac{1}{N_{ROI}}\sum_{i=1}^{N_{ROI}} \mathrm{Softmax}(Occ_i) * \log Occ_i' \tag{4.9}$$

where the terms with $'$ is the ground truth labels.

4.3. Proposed Serial Multi-Task Skeleton Estimation Network

In Section 4.2, we introduce our proposed multi-task skeleton estimation model training each task paralleled. However, when we treat these tasks paralleled, the information that can be shared between different tasks is the RoI features extracted from FPN and RoI Align layer. Especially for the body segmentation task and joint position estimation task, we use deep convolutional networks which makes the activation of ground truth label can hardly affect the RoI features during back propagation. Actually, the information added in our model are strongly related with each other. Figure 4.7 shows an example of the relationship among our training targets. The body segmentation describes the boundary of human skeleton, so it can be a limitation for the joint position estimation. Also, joint positions are helpful to model the structure of human pose, which can be a strong reference for body orientation estimation. Of course this is not the only way that these information are related, there are more combinations that our training targets can benefit from each other. In order to take advantage of these relationship and share the gradient of multi-task label in training, we improve our parallel model into a serial model by connecting different task at the convolutional branch.

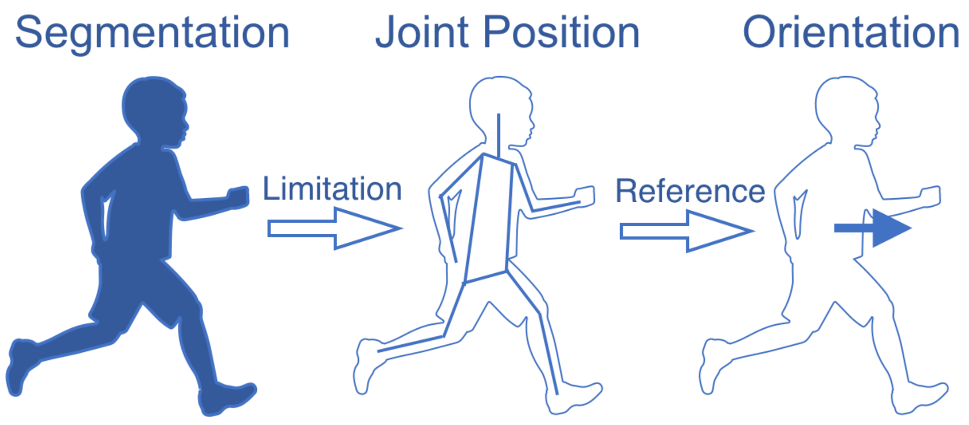


Figure 4.7 An example of the relationship our training targets

Figure 4.8 shows the of forward passing architecture of our proposed serial multi-task training network. For the reason that the orientation and occlusion outputs are vectors which is highly collapsed, we set this branch as the last output layer. For the left two branches, we build two models, of which one set body segmentation branch at first and feed its output into the joint position branch (shown in the left), and the other one directly input joint position results into the left two branches (shown in the right). We keep the connection between joint position branch and orientation and occlusion branch unchanged because the joint position is obviously a better reference for the prediction of body orientation than the body segmentation.

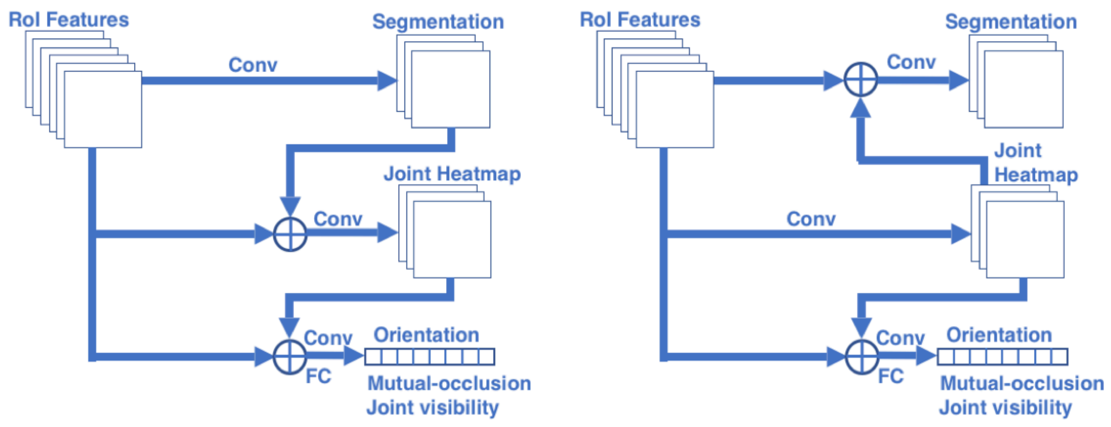


Figure 4.8 Proposed two serial multi-task training network

The new architecture of joint position estimation branch combined with body segmentation input is shown in Figure 4.9, where we add body segmentation output (resized to 14*14) into the RoI features to make the body segmentation be referred into convolutional process as a texture.

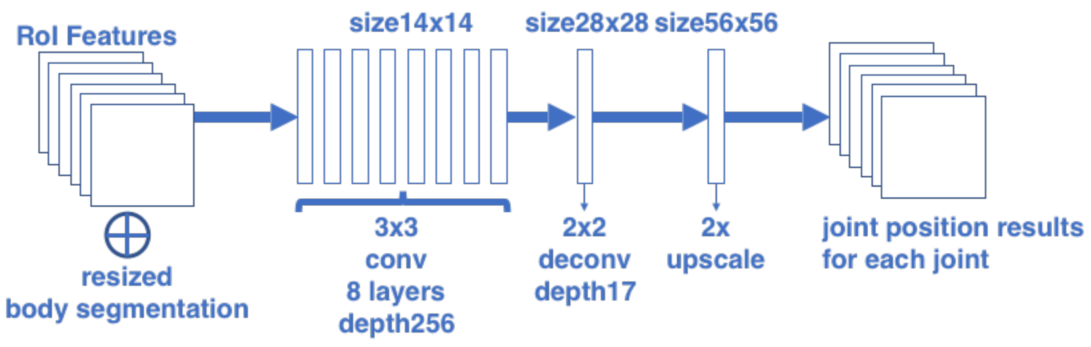


Figure 4.9 Joint position estimation branch combined with segmentation

For the body segmentation branch combined with joint position results, we add the output of the layer in joint position estimation branch before the last up-scaling layer (size 28*28), which is illustrated in Figure 4.10.

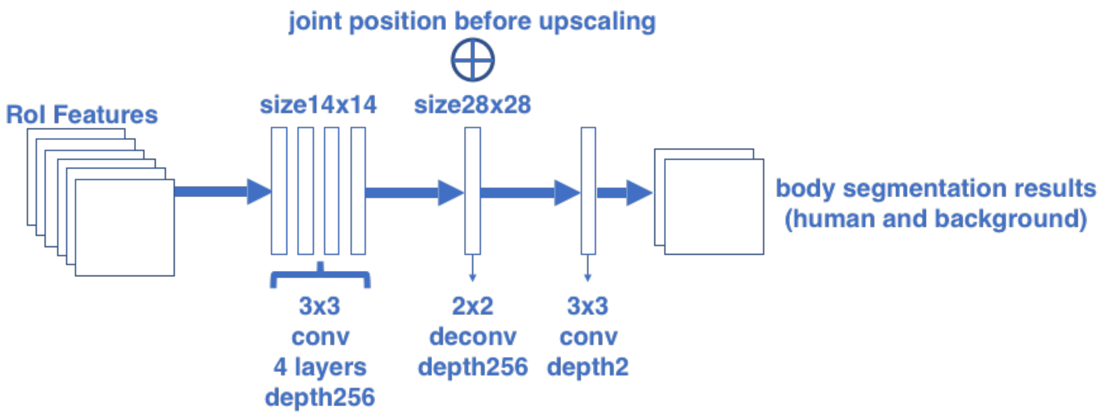


Figure 4.10 Body segmentation branch combined with joint position

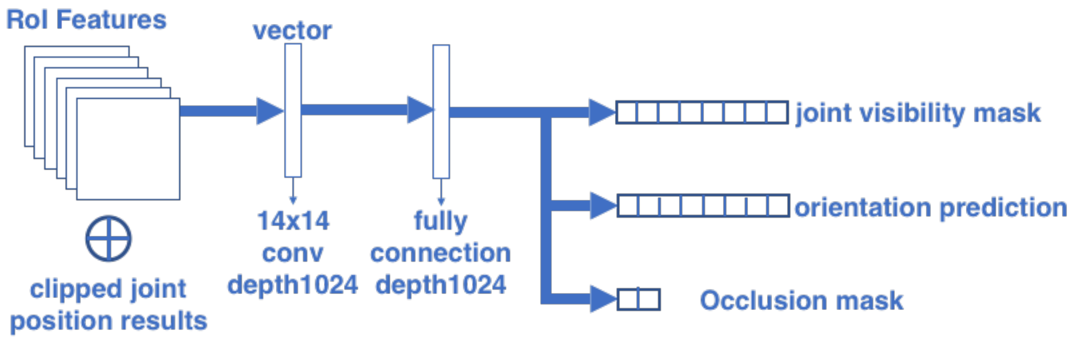


Figure 4.11 Orientation and occlusion branch combined with joint position

In the last connection between joint position estimation branch and orientation and occlusion branch, we clip the joint position results to fix the RoI features size and input it into the fully connected network together with RoI features (shown in Figure 4.11).

4.4. Experiment

In this section, we will show the details of the training dataset and the results of our proposed model. The evaluation and comparison will also be demonstrated in this part.

4.4.1 COCO Keypoint Detection Dataset

We use COCO Keypoint Challenge Dataset [63] for the training of body segmentation, skeleton estimation, joint visibility mask in our proposed model. COCO (Common Objects in Context) Dataset is an open dataset build by Microsoft and facebook etc., which has a large volume of images for general object detection and segmentation tasks. Keypoint detection is one of the tasks in COCO Dataset, which requires localization of person keypoints in challenging, uncontrolled conditions. Figure 4.12 gives a demonstration of this task, each person in the image has a set of annotations including segmentation, class name (in this task it is person), bounding box position, the number of labeled keypoints and a list of body joint information (labeled in the format (x, y, v) with 17 joints, where x, y is the location of joint, v is the visibility of joint that $v = 0$ means not labeled, $v = 1$ means labeled but not visible, $v = 2$ means labeled and visible).

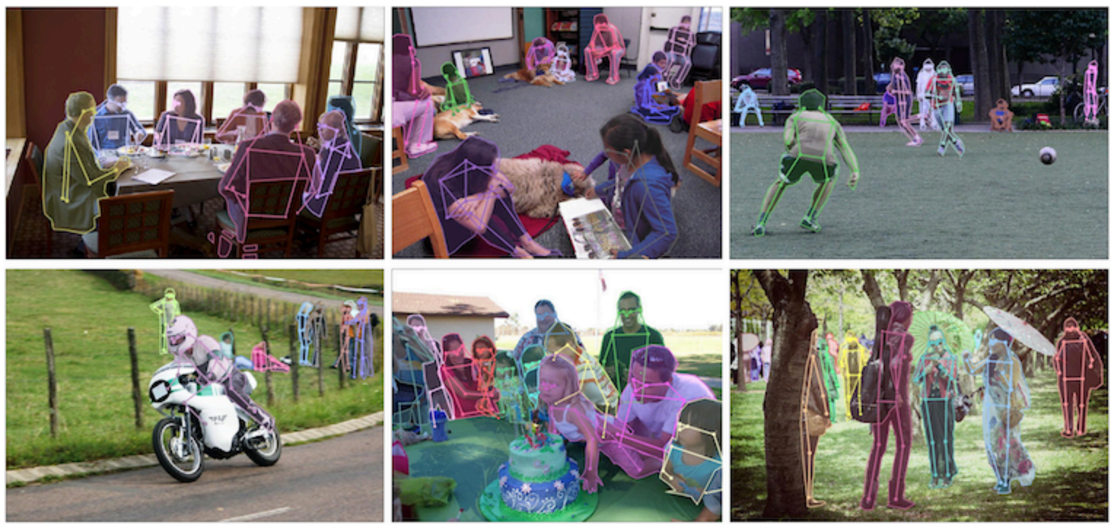


Figure 4.12 Demonstrations of COCO Keypoint Detection Dataset from their homepage

We trained our model on the COCO Keypoint Detection Dataset 2017 for the body segmentation, joint position estimation and joint visibility mask tasks, which includes 56599 images for taining and 2346 images for validation. We filtered the instances with less than 3 joints in order to teach our model to learn the whole body structure better and accelerate the converge of our model.

4.4.2 Extended Sub Dataset with Mutual-Occlusion and Body Orientation

For the reason that we add mutual-occlusion mask and orientation recognition task in our proposed model, which requires the annotations that do not exist in the COCO Keypoint Detection Dataset, we build a sub dataset with our team for the training on these tasks. In Table 1 we shows the scale of our sub dataset. We labeled about 2600 images from COCO Keypoint Detection Dataset 2017 with over 8000 human instances in different size and situations, where we use 90% of the whole dataset as our training dataset, and the left images are used for validation..

Table 1 Dataset for mutual-occlusion mask and orientation recognition

	Training	Validation	Total
Number of images	2306	260	2566

The label system for orientation recognition that we followed is shown in Figure 4.13, which refers the label system of our previous work in in Section 3.1 that we divides the horizontal space into 8 parts with each part of 45 degrees to label different body orientations. However, the image in COCO Dataset is much more complex than the customer dataset, where some instances is hard to be categorized into certain orientation. For this reason, we add a label 9 to deal with those instance orientations that are hard to distinguish.

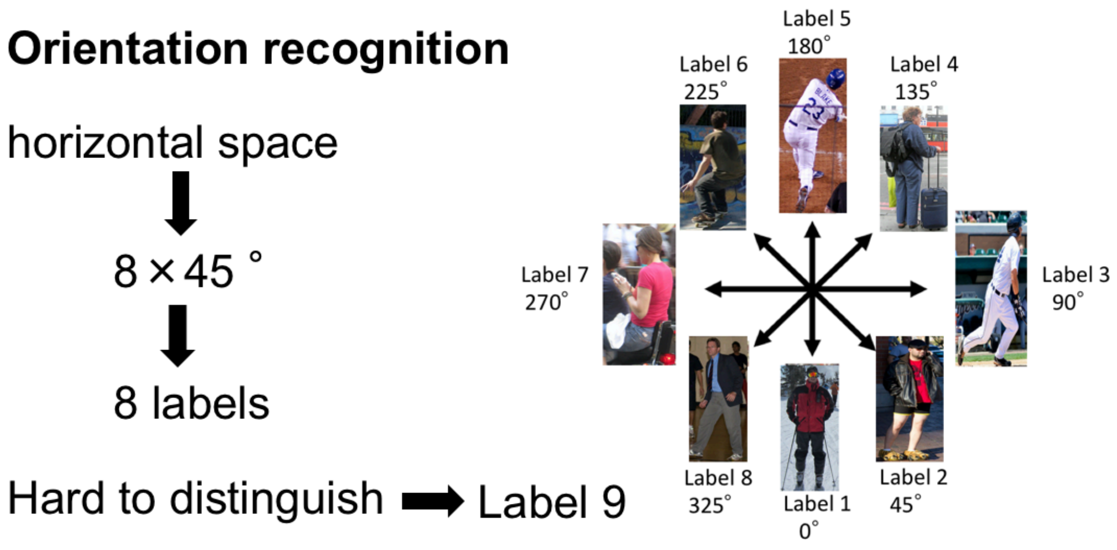


Figure 4.13 The label system for orientation recognition in our sub dataset

The label system for mutual-occlusion mask is shown in Figure 4.14. Unlike joint visibility mask that is a binary mask for each joint of each instance, we define mutual-occlusion mask as a binary mask for the whole instance in one bounding box. It is somehow similar to the classification class label that categorize the person in each bounding box into 2 classes: occluded and not occluded. In order to distinguish this mutual-occlusion label from self-occlusion, we only label the instances which are cropped by the edge of image or occluded by other person (or object) with Label 1 (occluded). For the instance that has self-occlusion due to its orientation, we label it as Label 0 (not occluded).

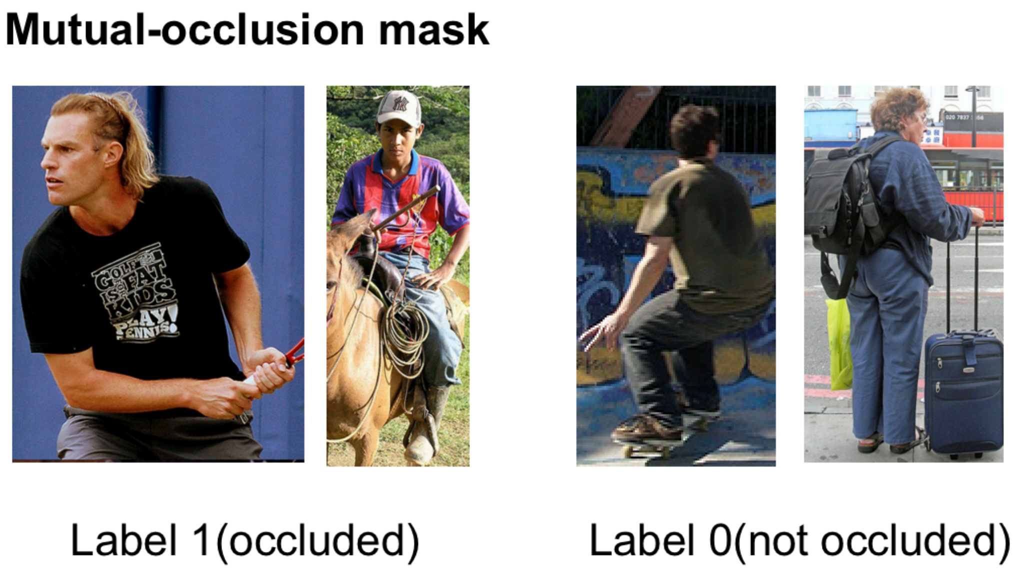


Figure 4.14 The label system for mutual-occlusion mask in our sub dataset

4.4.3 Experiment and Comparison

We first train our model on the COCO Keypoint Detection Dataset 2017 for the body segmentation task, joint position estimation task and joint visibility mask task. After 200k iterations with a batch size of 1, we keep the weights of the network of these three tasks and then train all the branches of our model on the sub dataset built by our own for 30k iterations.

To evaluate the performance of our model, we use Percentage of Correct Keypoints (PCK), which is a widely used evaluation metric. The correct keypoint is defined as the predicted joint whose distance from the true joint is less than a given threshold. In this evaluation step, we calculate the PCK of the joints with visibility > 0 with a threshold of 0.1*bounding box height. The evaluation results and comparisons between our different proposed models is shown in Table 2, where we test our models on a test set of 1000 images picked from COCO Keypoint Detection Dataset 2014 validation set. In this table, “Joint+Segm” represents our parallel multi-task model before training on the occlusion and orientation sub dataset, “Segm→Joint” represents our serial multi-task model that input joint position estimation results into the body segmentation branch, “Joint→Segm” represents our serial multi-task model that input body segmentation results into the joint position estimation branch, and “with Occ&Ori” represents models trained on the occlusion and orientation subset.

Table 2 Evaluation results of proposed models using PCK (%)

	Nose	Eye	Ear	Shoulder	Elbow	Wrist	Hip	Knee	Ankle	Overall
Joint+Segm	89.3	90.8	90.0	82.6	77.1	73.4	76.7	74.2	72.9	80.8
Joint+Segm withOcc&Ori	90.3	91.6	90.0	84.1	78.4	75.6	75.9	75.8	73.4	82.7
Segm→Joint	89.6	89.9	88.6	84.4	81.8	77.0	79.0	78.2	73.7	82.4
Segm→Joint withOcc&Ori	**94.1**	**94.1**	**93.3**	85.5	82.0	75.2	78.4	77.5	73.9	83.7
Joint→Segm	93.5	93.7	92.8	85.5	81.4	76.1	78.2	76.6	73.6	83.5
Joint→Segm withOcc&Ori	93.7	93.8	93.2	**86.9**	**82.6**	**77.8**	**80.5**	**78.9**	**74.1**	**84.6**

The evaluation results show that after training on our occlusion and orientation sub dataset, the accuracy of each model has an overall increase on each joint. And our serial multi-task models get higher accuracy than the parallel multi-task model. As a result, the serial model starts from joint position estimation branch gives the best performance, which we think is because the joint position estimation branch takes more advantage of the label from other branches in the back propagation. Some results of this model on multi-person images are shown in Figure 4.15. And the examples of wrong joint position estimation results are shown in Figure 4.16, where the image on the left top is a bad example that our model does not separate the two ankles but activate them at the same time, which leads to the low accuracy of ankle. And for the human instances with uncommon pose (image on the right top), our model also cannot make right estimation. The two images on the bottom show examples where the body segmentation and joint position estimation give wrong results together, which is a problem of multi-task network that we need to solve in the future.

We also test the orientation recognition task of our model on a test set labeled on images from the LSP dataset [65], which was also a widely used benchmark for human pose estimation. The details of orientation test dataset are shown in Table 3, we did not only focus on the normal orientations in the horizontal space but also labeled those images with instance which is hard to distinguish, thus the test data becomes balanced.

Table 3 Test dataset for orientation recognition

	8 orientations	Other orientation	Overall
Number of images	873	137	1000

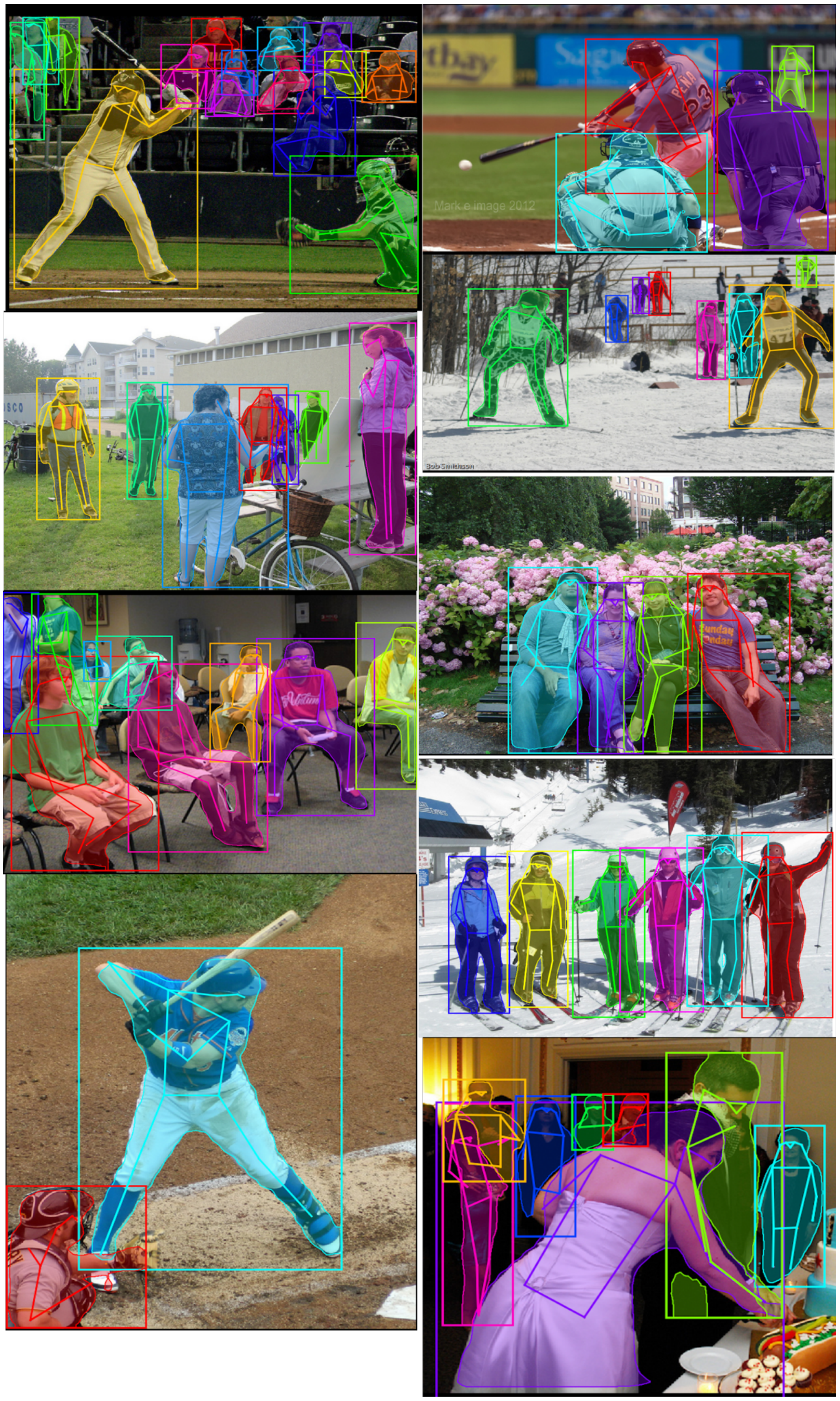


Figure 4.15 Body segmentation and joint position estimation results on COCO dataset

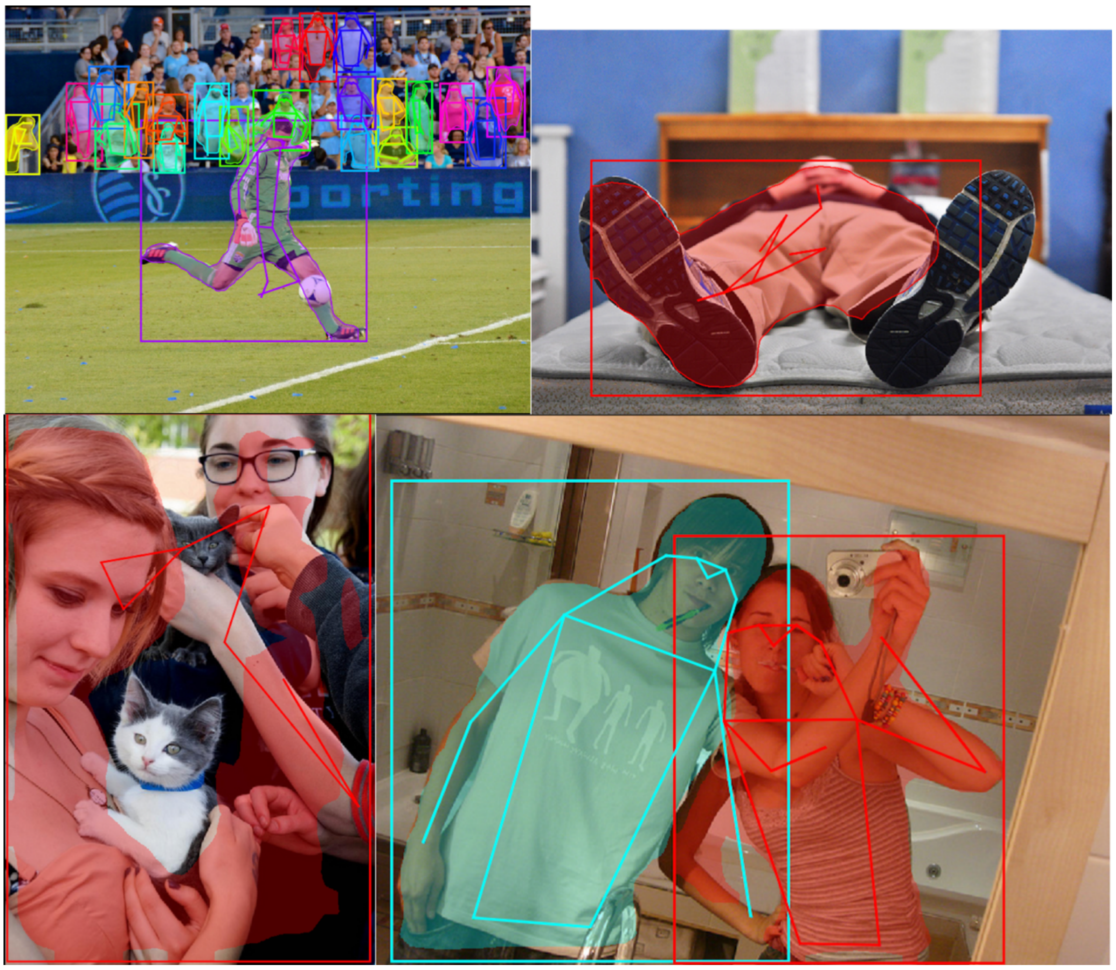


Figure 4.16 Some wrong joint position estimation results of our model

Table 4 Accuracy of orientation recognition in a strict principle (%)

Label	1	2	3	4	5	6	7	8	9
Degree	0	45	90	135	180	225	270	315	-
Joint+Segm with Occ&Ori	40.2	47.4	58.7	38.6	34.5	38.1	63.1	38.6	48.1
Segm→Joint with Occ&Ori	79.2	**50.0**	**67.7**	46.4	80.1	**62.3**	**79.0**	42.3	49.4
Joint→Segm with Occ&Ori	**88.6**	47.2	58.8	**50.0**	**85.9**	54.7	75.6	**47.1**	**77.1**

Table 4 shows the accuracy of the orientation recognition task of our proposed models when we only permit one orientation is to be correct. The results are not so good because we treat the orientation label very carefully when building the training dataset in order to make the model more sensitive to the orientations that are not at right angles. However, we still can see the benefit after we input joint positions estimation results into the occlusion and orientation branch. The serial model starts with joint position estimation gets the best overall accuracy, where we think this model benefits from the increase of joint position accuracy.

Then we use a compatible principle that allows a neighboring error. Our models show more satisfying performance on the orientation prediction (shown in Table 5). For the normal 8 orientations, our two serial models get similar high accuracy. But for the other orientation (label 9), the serial model starts with joint estimation branch has a complete good recognition rate than the other models, which means this model learned a general recognition ability for body orientation. Figure 4.17 and Figure 4.18 visualize several results of the whole multi-task outputs of our proposed model on LSP Dataset and COCO Dataset, where the red arrow in the center of each bounding box shows the body orientation recognition result, and the number with blue background on the left top of bounding box represents the mutual-occlusion condition of each person (1 is mutual-occluded and 0 is not mutual-occluded).

Table 5 Accuracy of orientation recognition when a neighbour error allowed (%)

Label	1	2	3	4	5	6	7	8	9
Degree	0	45	90	135	180	225	270	315	-
Joint+Segm with Occ&Ori	91.7	95.8	93.6	86.7	79.0	89.2	**93.7**	88.7	48.1
Segm→Joint with Occ&Ori	**98.2**	98.4	**100**	**90.9**	83.6	**94.4**	93.3	**94.7**	49.4
Joint→Segm with Occ&Ori	98.1	**98.5**	99.2	**90.9**	**94.7**	92.0	93.2	93.8	**77.1**

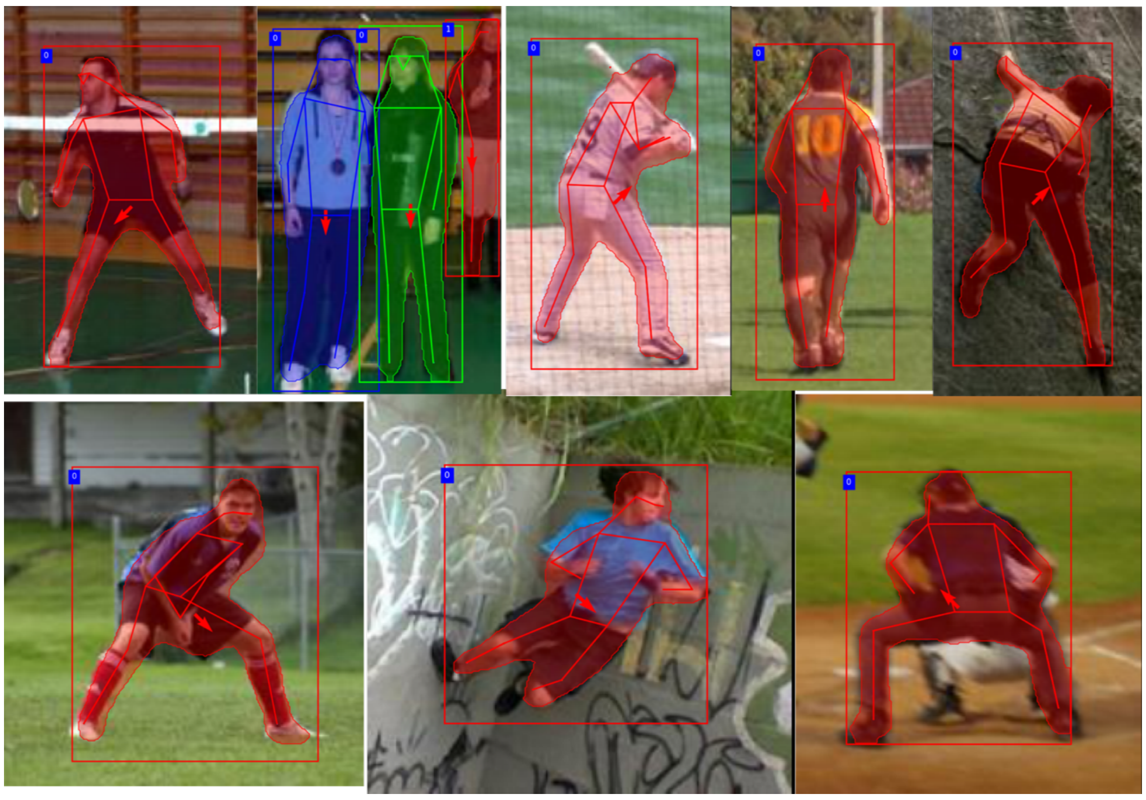


Figure 4.17 Examples of the results on LSP dataset with orientation recognition

We also compare our multi-task model with other pose estimation methods. The comparison among our model, OpenPose [22] and MultiPoseNet [66] is shown in Table 6, where we use the same test dataset and evaluation configuration as Table 2. Our serial model starts with joint position estimation gets higher overall PCK than MultiPoseNet, which benefits from the higher accuracy of the face parts. For the whole body joints without face parts, our model gives similar PCK as MultiPoseNet. The OpenPose performs higher PCKs taking the advantage of its bottom-up architecture and Part Affinity Filed network. But we found that OpenPose has an obvious over detection tendency on the "not joint" objects because it does not use bounding box to limit the area of convolution.

Table 6 Comparison between proposed models and other methods using PCK (%)

	Nose	Eye	Ear	Shoulder	Elbow	Wrist	Hip	Knee	Ankle	Overall
OpenPose [22]	92.2	92.1	92.3	89.5	83.9	74.2	83.3	83.5	80.8	85.7
MultiPoseNet[66]	88.9	88.5	88.9	84.5	82.0	79.6	78.6	79.7	78.0	83.2
Joint→Segm with Occ&Ori	93.7	93.8	93.2	86.9	82.6	77.8	80.5	78.9	74.1	84.6

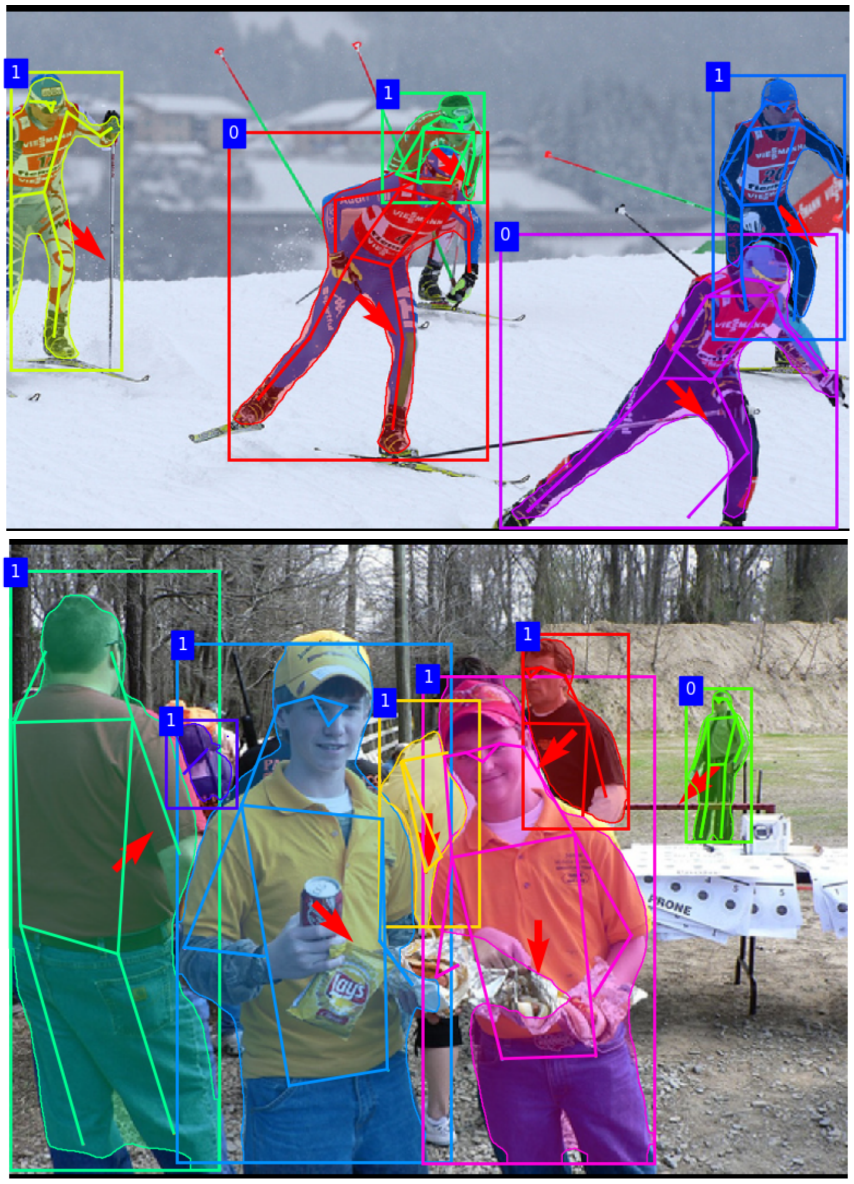


Figure 4.18 Examples of the results on COCO Dataset with all multi-task outputs

In order to evaluate the ability to make right joint detections, we defined a Correct Detection Rate (CDR) to deal with the over detection tendency, which can be expressed as:

$$CorrectDetectionRate = \frac{(\sum Joint_{detected}) \cap (Joint_{groundtruth})|_{vis>0}}{\sum Joint_{detected}} \quad (4.2)$$

where $Joint_{detected}$ is the joints detected by the network, $Joint_{groundtruth}$ is the joints labeled in the ground truth annotations. Only the joints which are labeled with location annotation contribute to the Correct Detection Rate when they are detected. When a joint out of annotation is detected, it will be counted as an over-detection.

The comparison between our proposed method with other methods using Correct Detection Rate are shown in Table 7. As a trade-off of higher accuracy of joint position estimation, OpenPose gets a low overall Correct Detection Rate, which means its over-detection is more than other methods. While our method gets higher Correct Detection Rate by using joint visibility mask to penalize the joint that does not exist in the image.

Table 7 Comparison between proposed models and other methods using CDR (%)

	Nose	Eye	Ear	Shoulder	Elbow	Wrist	Hip	Knee	Ankle	Overall
OpenPose [22]	72.2	70.5	66.2	74.8	69.3	72.8	72.7	74.8	66.5	71.2
MultiPoseNet[66]	85.6	81.1	77.6	88.3	82.4	80.5	83.7	80.3	71.0	81.2
Joint+Segm with Occ&Ori	82.7	80.4	75.2	86.0	80.0	78.4	88.4	87.7	82.9	82.4
Segm→Joint with Occ&Ori	89.3	86.6	79.2	85.2	77.7	77.8	77.8	85.3	82.7	82.4
Joint→Segm with Occ&Ori	84.7	86.6	80.6	85.2	82.3	81.4	85.8	87.8	79.3	83.7

Chapter 5.

Conclusion

In this chapter, we will give the conclusion of this research and future work.

5.1. Summary

In this research, we present a multi-person skeleton estimation method using multi-task deep learning network. Our proposed network model is an extend multi-task training network based on a Mask R-CNN layer heads, and it consists of five tasks: (1) skeleton estimation, (2) body segmentation, (3) joint visibility mask, (4) body orientation recognition, (5) mutual-occlusion mask, and they are seperated into 3 branches: body segmentation branch, joint position estimation branch, orientation and occlusion branch. We first build our parallel proposed multi-task network with each task separately in training steps. In order to strengthen the gradient from labels of each multi-task ground truth, we then improve our proposed method into 2 serial multi-task models by connecting different branch in the training. In evaluation step, we first evaluate the accuracy of the joint position estimation of our model using Percentage of Correct Keypoints. We also test the orientation recognition ability of our proposed model on a test dataset based on LSP dataset. In comparison step, we first compare our different proposed models. It shows that by adding body orientation and mutual-occlusion mask into the training, our proposed model performance better on both joint position estimation and orientation recognition. In the two serial multi-task models, the model starts from joint position estimation gives better performance benefiting from the ground truth of other branches in back propagation. We also compare our model with other skeleton estimation methods. Besides PCK to evaluate the accuracy of detected joints in ground truth labels, we also define a Correct Detection Rate to deal with the over-detection problem. Considering both these two evaluation metrics, our proposed model can give a satisfying results compared with other methods.

For the training of joint position estimation, body segmentation and joint visibility mask, we use COCO Keypoint Detection Dataset. And for further training for orientation recognition and mutual-occlusion mask, we build a new sub dataset which includes about 2600 images from COCO Keypoint Detection Dataset with their original annotations and extended with body orientation and mutual-occlusion labels as new annotation.

5.2. Future Work

There are still some problems need to be solved in our proposed models such as the low accuracy of ankle joints and the segmentation sometimes cannot help the joint position estimation task. In the future, we would like to continue improving this model by using more valuable information (e.g. body parsing annotations instead of body segmentation) or using more complex architecture to define the connections hidden in human skeleton.

Thanks

This research is under the instruction of Professor Kamijo. He gives me the direction and advices not only on the researches but also on my life. Also researcher Gu gives me a lot of advices on how to solve problems and how to write papers. I won't finish my thesis without their meaningful discussions and help.

To the other members in the lab, Kamiya, Y. Zhang, Bao, Mahdi, Ehsan, X. Wang, and, I want to thank them for giving me wonderful study atmosphere and kind help in the lab. Finally, I want to thank my parents and friends for their kind and unselfish supports.

Reference

[1] C. Sminchisescu and A. Telea. Human Pose Estimation from Silhouettes. A Consistent Approach Using Distance Level Sets. in Proceedings of the International Conference on Computer Graphics, Visualization and Computer Vision (WSCG), 2002.

[2] A. Mittal, L. Zhao, and L. S. Davis. Human body pose estimation using silhouette shape analysis. in Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance, 2003, pp. 263–270.

[3] D. Ramanan, D. A. Forsyth, and A. Zisserman. Tracking People by Learning Their Appearance. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), vol. 29, no. 1, pp. 65–81, Jan. 2007.

[4] D. Weiss, B. Sapp, and B. Taskar. Sidestepping intractable inference with structured ensemble cascades. in Advances in Neural Information Processing Systems, 2010, pp. 2415–2423.

[5] A. Cherian, J. Mairal, K. Alahari, and C. Schmid. Mixing Body-Part Sequences for Human Pose Estimation. in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 2353–2360.

[6] P. F. Felzenszwalb and D. P. Huttenlocher. Pictorial structures for object recognition. International Journal of Computer Vision (IJCV), vol. 61, no. 1, pp. 55–79, Jan. 2005.

[7] Y. Yang and D. Ramanan. Articulated Human Detection with Flexible Mixtures of Parts. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), vol. 35, no. 12, pp. 2878–2890, Dec. 2013.

[8] L. Pishchulin, M. Andriluka, P. Gehler, and B. Schiele. Poselet Conditioned Pictorial Structures. in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2013, pp. 588–595.

[9] B. B. Le Cun, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. in Neural Information Processing Systems (NIPS), 1989.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. in Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.

[11] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[12] C. Szegedy et al. Going deeper with convolutions. in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp.1–9.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778.

[14] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 1653–1660.

[15] J. Carreira, P. Agrawal, K. Fragkiadaki, and J. Malik. Human pose estimation with iterative error feedback. in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 4733–4742.

[16] A. Haque, B. Peng, Z. Luo, A. Alahi, S. Yeung, and L. Fei-Fei. Towards viewpoint invariant 3D human pose estimation. in Proceedings of the European Conference on Computer Vision (ECCV), 2016, pp. 160–177.

[17] S.-E. Wei, V. Ramakrishna, T. Kanade, Y. Sheikh. Convolutional pose machines. the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[18] A. Newell, K. Yang, J. Deng. Stacked Hourglass Networks for Human Pose Estimation. the IEEE Conference on Computer Vision and Pat- tern Recognition (CVPR), 2016.

[19] L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. V. Gehler, B. Schiele. Deep Cut: Joint Subset Partition and Labeling for Multi Person Pose Estimation. the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp.4929-4937, 2016.

[20] R. Girshick. Fast R-CNN. the International Conference on Computer Vision (ICCV), 2015.

[21] E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, B. Schiele. DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model. European Conference on Computer Vision (ECCV), pp.34-50, 2016.

[22] Z. Cao, T. Simon, S.-E. Wei, Y. Sheikh. Real-time Multi-Person 2D Pose Estimation using Part Affinity Fields. the IEEE Conference on Computer Vision and Pattern Recognition(CVPR), 2017.

[23] G. Papandreou, T. Zhu, N. Kanazawa, A. Toshev, J. Tompson, C. Bregler, K. Murphy. Towards Accurate Multi-person Pose Estimation in the Wild. the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[24] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, K. Murphy. Speed/accuracy tradeoffs for modern convolutional object detectors. the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[25] K. He, G. Gkioxari, P. Dollar, and R. Girshick, Mask R-CNN. in Proceedings of ICCV, 2017.

[26] S. Ren, K. He, R. Girshick, and J. Sun, Faster R-CNN: Towards real-time object detection with region proposal networks. In NIPS, 2015.

[27] Azizpour and I. Laptev, Object detection using strongly-supervised deformable part models. in Proceedings of the European Conference on Computer Vision (ECCV), 2012, pp. 836-849.

[28] G. Ghiasi, Y. Yang, D. Ramanan, and C. C. Fowlkes, Parsing occluded people. in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014, pp. 2401-2408.

[29] U. Rafi, J. Gall, and B. Leibe, A semantic occlusion model for human pose estimation from a single depth image. in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2015, pp. 67-74.

[30] A. Haque, B. Peng, Z. Luo, A. Alahi, S. Yeung, and L. Fei-Fei, Towards viewpoint invariant 3D human pose estimation. arXiv preprint arXiv:1603.07076, 2016.

[31] T. Gandhi and M. M. Trivedi, Image based estimation of pedestrian orientation for improving path prediction. in 2008 IEEE Intelligent Vehicles Symposium, 2008, pp. 506–511.

[32] N. Dalal and B. Triggs, Histograms of oriented gradients for human detection. in IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol.1, pp. 886–893, 2005.

[33] M. Goffredo, I. Bouchrika, J. N. Carter, and M. S. Nixon, Performance analysis for automated gait extraction and recognition in multi-camera surveillance. Multimed Tools Appl, vol. 50, no. 1, pp. 75–94, Oct. 2009.

[34] J. Liu, Y. Gu, and S. Kamijo, Joint Customer Pose and Orientation Estimation using Deep Neural Network from Surveillance Camera. in IEEE International Symposium on Multimedia (ISM), 2016.

[35] Y. Huang and L. Mucke, Alzheimer mechanisms and therapeutic strategies. Cell, vol. 148, no. 6, pp. 1204–1222, 2012.

[36] Wikipedia, Biological neural network. https://en.wikipedia.org/wiki/Biological_neural_network.

[37] J. J. DiCarlo, D. Zoccolan, and N. C. Rust, How does the brain solve visual object recognition?. Neuron, vol. 73, no. 3, pp. 415–434, 2012.

[38] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning internal representations by error propagation. tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[39] S. Ioffe and C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift. in International Conference on Machine Learning, pp. 448–456, 2015.

[40] A. L. Maas, A. Y. Hannun, and A. Y. Ng, Rectifier nonlinearities improve neural network acoustic models. in Proc. ICML, vol. 30, 2013.

[41] K. He, X. Zhang, S. Ren, and J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. in Proceedings of the IEEE international conference on computer vision, pp. 1026–1034, 2015.

[42] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition. Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[43] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., Imagenet large scale visual recognition challenge. International Journal of Computer Vision, vol. 115, no. 3, pp. 211–252, 2015.

[44] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus, Deconvolutional networks. in Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, pp. 2528–2535, IEEE, 2010.

[45] M. D. Zeiler and R. Fergus, Visualizing and understanding convolutional networks. in European conference on computer vision, pp. 818–833, Springer, 2014.

[46] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning representations by back-propagating errors. nature, vol. 323, no. 6088, pp. 533–536, 1986.

[47] J. A. Suykens and J. Vandewalle, Least squares support vector machine classifiers. Neural processing letters, vol. 9, no. 3, pp. 293–300, 1999.

[48] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al., Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467, 2016.

[49] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, Caffe: Convolutional architecture for fast feature embedding. in Proceedings of the 22nd ACM international conference on Multimedia, pp. 675–678, ACM, 2014.

[50] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, On the importance of initialization and momentum in deep learning. in International conference on machine learning, pp. 1139–1147, 2013.

[51] J. Duchi, E. Hazan, and Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, vol. 12, no. Jul, pp. 2121–2159, 2011.

[52] M. D. Zeiler, Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701, 2012.

[53] G. Hinton, N. Srivastava, and K. Swersky, Rmsprop: Divide the gradient by a running average of its recent magnitude. Neural networks for machine learning, Coursera lecture 6e, 2012.

[54] D. Kingma and J. Ba, Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[55] A. Haque, B. Peng, Z. Luo, A. Alahi, S. Yeung, and L. Fei-Fei, Towards viewpoint invariant 3D human pose estimation. in Proceedings of the European Conference on Computer Vision (ECCV), 2016, pp. 160–177.

[56] X. Chen and A. L. Yuille, Articulated pose estimation by a graphical model with image dependent pairwise relations. in Advances in Neural Information Processing Systems, 2014, pp. 1736–1744.

[57] X. Chen and A. L. Yuille, Parsing occluded people by flexible compositions. in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3945–3954.

[58] J. Long, E. Shelhamer, and T. Darrell, Fully convolutional networks for semantic segmentation. in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3431–3440.

[59] R. Girshick, J. Donahue, T. Darrell, and J. Malik, Region-based convolutional networks for accurate object detection and segmentation. IEEE transactions on pattern analysis and machine intelligence, vol. 38, no. 1, pp. 142–158, 2016.

[60] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, Selective search for object recognition. International journal of computer vision, vol. 104, no.2, pp. 154–171, 2013.

[61] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, The pascal visual object classes (voc) challenge. International journal of computer vision, vol. 88, no. 2, pp. 303–338, 2010.

[62] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, Attention-based models for speech recognition. in Advances in Neural Information Processing Systems, pp. 577–585, 2015.

[63] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. Zitnick, Microsoft COCO: Common Objects in Context. In ECCV, 2014.

[64] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In CVPR, 2017.

[65] S. Johnson and M. Everingham, Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation. In Proceedings of the 21st British Machine Vision Conference (BMVC), 2010.

[66] M. Kocabas, S. Karagoz and E. Akbas, MultiPoseNet: Fast Multi-Person Pose Estimation using Pose Residual Network. In ECCV, 2018.

Publication List

International Conference:

Zhang, H., Gu, Y., Kamijo, S.(2019, January) Orientation and Occlusion Aware Multi-Person Pose Estimation using Multi-Task Deep Learning Network . In Consumer Electronics (ICCE), 2018 IEEE International Conference. IEEE. (Best Paper Award)