Tampere University

Haochen Zong

EVALUATION OF TRAINING DATASET AND NEURAL NETWORK ARCHITECTURES FOR HAND POSE ESTIMATION IN REAL TIME

Faculty of Information Technology and Communication Sciences (ITC)
Master's thesis
April 2022

Abstract

Haochen Zong: Evaluation of Training Dataset and Neural Network Architectures for Hand Pose Estimation in Real Time
Master's thesis
Tampere University
Master's Degree Programme in Data Engineering and Machine Learning
April 2022

This thesis aims at implementing an application to recognise hand gesture in real-time using the Unreal Engine 4 (UE4) game engine. This thesis completes three tasks. First, neural network models are built to get 2D coordinates of hand joints from an RGB image. Second, the optimal model is found which has more precise prediction and less prediction time by using different neural network architectures and training datasets. Third, 2D hand pose estimation is implemented by using the trained model in real-time in the UE4 game engine. While there are some research works on hand pose estimation from an RGB image, they only focus on prediction accuracy not prediction time.

In this thesis, two neural network architectures for hand pose estimation (HRNet and ResNet) are studied by analyzing their architectures and performance in testing. Each architecture is trained by four datasets, so eight models are generated. Each model is tested on those four datasets. Then, the eight models are tested on self-taken photos. Their performances are compared through observations based on the accuracy of predicted hand joints' position and the real ones.

The prediction accuracy and prediction time are the evaluation indices for evaluating models. This research finds that prediction accuracy of HRNet is little higher than ResNet, but the prediction time of HRNet is little higher than ResNet. The training models on the dataset, OneHand10K has batter performance compared to other datasets.

Keywords: hand pose estimation, hand gesture, Unreal Engine 4, deep neural network

Preface

I would like to show my gratitude to my supervisors, Professor Tapio Elomaa and Senior Research Fellow, Timo Nummenmaa. They guided me and supported me when I was doing my thesis work.

I would like to give thanks to my girlfriend. She gave me support in spirit and study during my thesis work so I was confident in continuing and complete my thesis.

Tampere, Finland, 18th April 2022

Haochen Zong

Contents

List of Figures

List of Tables

1 INTRODUCTION

The development of technology has applied in mobile applications, personal computers, etc. It has changed people's daily lives. One of the typical scenarios of technology in applications is hand pose estimation. Users do some hand gestures as commands to control smart home devices. They use built-in webcams to capture users' hand gestures and do the corresponding actions, e.g. open television, draw curtains, etc. Moreover, players can play some games by doing some hand gestures without using mouse or keyboard. However, the prediction accuracy of hand pose estimation on some devices is bad. Also, the prediction time is long. Some scenarios have high requirement in prediction accuracy but some need fast hand pose prediction. The trade-off between accuracy and speed is a challenge for researchers.

A method to overcome this challenge is testing hand models' performance on prediction accuracy and prediction time in different scenarios and some of them are chosen based on the requirements of specific scenarios. The goal of this thesis is implementing an application to recognise hand gesture in real-time using the UE4 game engine. The procedure of this thesis work's pipeline includes researching neural network architectures for hand gesture recognition, training hand models, getting prediction of trained models, comparing the performance of inference quality and speed. The performance of each hand model is evaluated for hand gesture estimation tasks on self-taken photos. These models are used to do 2D hand pose estimation on UE4 game engine.

This thesis work is generalizable to real-time applications which need hand pose estimation. More accurate and fast estimation can be achieved. In this thesis, neural network models are built to get 2D coordinates of hand joints from an RGB image. Also, the optimal model is found which has more precise prediction or less prediction time. The real-time applications can use webcams to capture frames and get 2D coordinates of users' hand joints. It can use this data to estimate the meaning of hand gestures to do the next step. Application developers can choose one of the models according to their emphasis on accuracy or time.

The structure of the thesis is divided into four parts. Chapter 2 introduces the fundamental knowledge of Artificial Neural Networks, Convolution Neural Networks and game engine. Chapter 3 introduces the work of literature review. Different models of hand pose estimation from a depth images or an RGB image were introduced. Chapter 4 introduces the evaluation of hand models for a single image and in real-time application. Chapter 5 draws the conclusion of the thesis.

2 THEORY

In this thesis, two artificial neural network architectures (ANN) for hand pose estimation, HRNet and ResNet are studied. Each of them is trained by four datasets. These two models are related to the knowledge of ANN and Convolutional Neural Network (CNN). Because of this, the essential elements of ANN and its training process are necessary to be introduced in the first section. CNN is described in the second section. Its main content is the construction and function of each layer of CNN. Next, the trained models are used to do 2D hand pose estimation in real-time using the Unreal Engine 4 game engine (UE4). Hence, in the third section, general knowledge of game engine, the main systems I used in UE4 are described.

2.1 Artificial Neural Network (ANN)

The name and structure of ANN are modeled from the neural networks in the brains of animals. In addition, the way data is transformed in ANN resembles the way organic nerves convey impulses. The fundamental theory of the perceptron, activation function, multilayer perceptron (MLP), loss function, optimization strategies in ANN, and forward and backward propagation (FP & BP) are all covered in this chapter.

2.1.1 Perceptron

The perceptron is the fundamental unit of ANN (Mcculloch and Pitts 1943). It consists of numerous inputs, an adder, an activation function, and an output y. The multiple inputs of perceptron are input values which constitute a vector, $\mathbf{x} = (x_1, x_2, ..., x_n)$. The connection intensity of each perceptron in ANN is affected by the weight value. The greater the weight of an input, the more significant it is. Each input has a weight associated with it. The weights are represented by a vector, $\mathbf{w} = (w_1, w_2, ..., w_n)$ and it times $\mathbf{x}$ by dot production. The sum is then compared with a threshold (Kanal 2003). The adder then combines $\mathbf{x}$ and the threshold. The structure of a perceptron is shown in Figure 2.1.

The perceptron is used as a binary classifier. The threshold is used to classify the output of the adder to get the binary result, indicated as $0, 1$. The accurate classification of samples is controlled by the appropriate threshold value. It determines which output will be used. The output is 1 if $\mathbf{w} \cdot \mathbf{x}$ is greater than the threshold. The result is 0 if the value is less than the threshold. On the Equation 2.1, if the threshold is moved to the side of $\mathbf{w} \cdot \mathbf{x}$, b signifies the negative threshold.

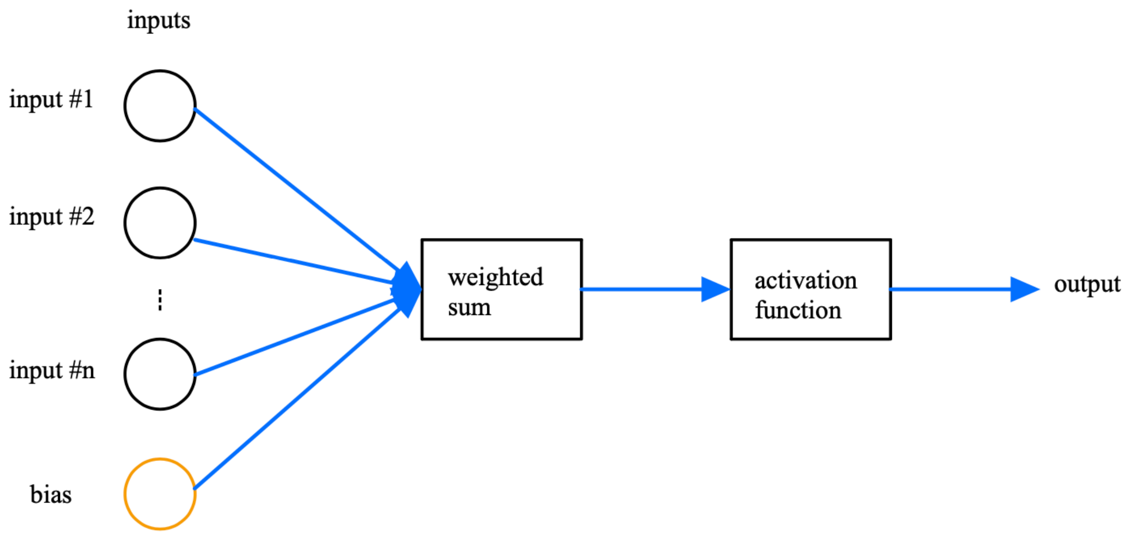


Figure 2.1 *The structure of perceptron*

$$y = \begin{cases} 1, \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0, \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \end{cases} \tag{2.1}$$

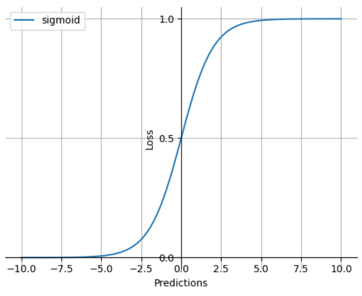
2.1.2 Activation function

The activation function f changes the linear output of perceptron to nonlinear so the ANN is be capable for nonlinear problems. The f takes the output from the adder and does nonlinear transformation to it. It clamps the output between a range, e.g. $(-1, 1)$, $(0, 1)$. Some popular activation functions are *Sigmoid*, *Tanh*, $ReLU$, *Softmax*.

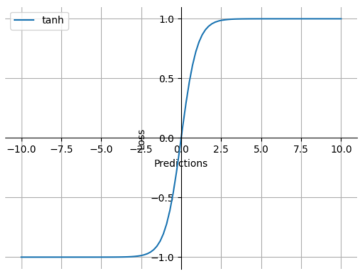
The function of *Sigmoid* is shown as Equation 2.2 (Kyurkchiev 2016). When the independent variable x is getting closer to negative infinity, the $\sigma(x)$ will be closer to 0. When x is becoming positive infinity, the $\sigma(x)$ will be near 1. The $\sigma(x)$ equals to 0.5 when $x = 0$. Hence, the range of $\sigma(x)$ is $(0, 1)$. The *Sigmoid* does normalization for all of the output, so it is good at predicting probability. Figure 2.2(a) shows the plot of *Sigmoid*.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

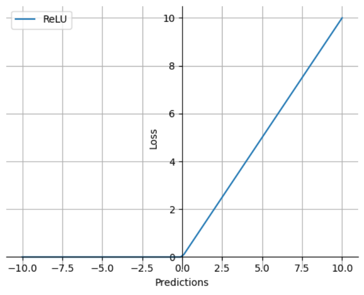
The function of *Tanh* is shown as Equation 2.3. When the independent variable x is getting closer to negative infinity, the $tanh(x)$ will be closer to -1. When x is becoming positive infinity, the $tanh(x)$ will be near 1. The $tanh(x)$ equals to 0 when $x = 0$. Hence, the range of $tanh(x)$ is $(-1, 1)$. Figure 2.2(b) shows the plot of *Tanh*.



(a) Sigmoid activation function



(b) Tanh activation function



(c) ReLU activation function

Figure 2.2 *Some activation functions*

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.3}$$

The function of *ReLU* (Xu et al. 2015) is shown as Equation 2.4. It can also be written as $R(x) = max(0, x)$. When the independent variable x changes from 0 to negative infinity, the *ReLU* will be 0. When x is the positive value, the *ReLU* will be same as x. Hence, the range of *ReLU* is $[0, \infty)$. Figure 2.2(c) shows the plot of *ReLU*.

$$R(x) = \begin{cases} x, \text{if } x > 0 \\ 0, \text{if } x \leq 0 \end{cases} \tag{2.4}$$

The *softmax* (Bridle 1989) is used in multiple classification. It maps the output of multiple perceptrons to a range of $(0, 1)$. The function of *softmax* is shown as Equation 2.5, where i denotes a category in multiple classification and the number of them is J. Its input is a a vector of real numbers. *softmax* normalizes this vector to $(0, 1)$ and the sum of y_i is 1. Compared to the function *max*, *max* only outputs the maximum value, but *softmax* outputs the probabilities of obtaining a some of categorizes. The highest y_i denotes a category i has the highest probability. This category is the desired prediction.

$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}, \text{for } i = 1, ..., J \tag{2.5}$$

2.1.3 Multilayer Perceptron (MLP)

Traditional MLP has three layers: input layer, hidden layer and output layer. The layer consists of multiple perceptrons. Each perceptron is connected to all of the perceptron in the next layer, which means each layer is fully-connected. The input or output layer has one layer but the hidden layers might have multiple ones or no layer. The workflow of ANN is divided into two parts, forward propagation and backward propagation. The structure of MLP is shown in the Figure 2.3.

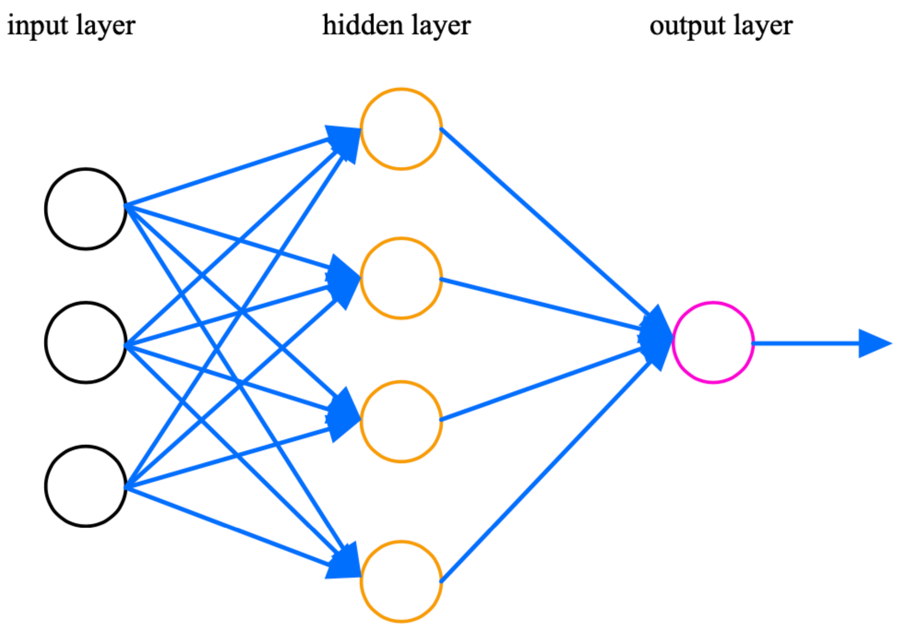


Figure 2.3 *The structure of MLP*

2.1.4 Forward Propagation (FP)

The forward propagation starts from the input layer and ends at the output in the forward direction. The n input data $(x_0^{(0)}, x_1^{(0)}, ..., x_n^{(0)})$ is put to the first perceptron in the first layer of hidden layers, then to others. For this perceptron, if the corresponding weights, and bias are $(w_{0,0}, w_{0,1}, ..., w_{0,n})$, b_0 respectively, the activation function f of the first perceptron processes the data and generates the output, as shown in the Equation 2.6.

$$x_0^{(0)} = f(w_{0,0}x_0^{(0)} + w_{0,1}x_1^{(0)} + \cdots + w_{0,n}x_n^{(0)} + b_0) \tag{2.6}$$

The rest of perceptron in this layer do the same way as the first one. Each output of them composes a matrix, as shown in the Equation 2.7. The Equation 2.8 verctorizes Equation 2.7, where **W** denotes the matrix of the weight, k means the number of perceptrons in the first layer of hidden layers.

$$\begin{bmatrix} x_0^{(1)} \\ x_1^{(1)} \\ \vdots \\ x_n^{(1)} \end{bmatrix} = f\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix} \begin{bmatrix} x_0^{(0)} \\ x_1^{(0)} \\ \vdots \\ x_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}\right) \tag{2.7}$$

$$\mathbf{x}^{(1)} = f(\mathbf{W}\mathbf{x}^{(0)} + \mathbf{b}) \tag{2.8}$$

Then, these outputs are transferred to the successive layers as their input. The the last layer of the hidden layers transfers its output to the output layer. Finally, the output layer calculates the result. If the output layer has p nodes which means there are p outputs y, the forward propagation is described as a function, as shown in the Equation 2.9.

$$f(x_1, x_2, ..., x_n) = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_p \end{bmatrix} \tag{2.9}$$

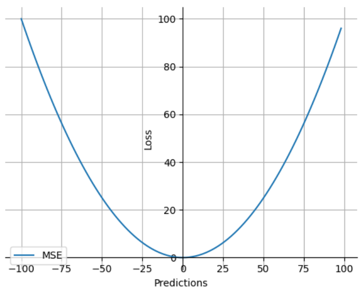
2.1.5 Loss function

The loss function measures the difference between the predicted ith value $\hat{y}_i$ and the expected value y_i, denoted as $L(y_i, \hat{y}_i)$, which means it can measure the performance of the prediction ability of an ANN. The loss function of regression includes *mean square error (MSE)*, *mean absolute error (MAE)*, *huber loss*, etc. The loss function of classification includes *log loss*, *hinge loss*, etc.

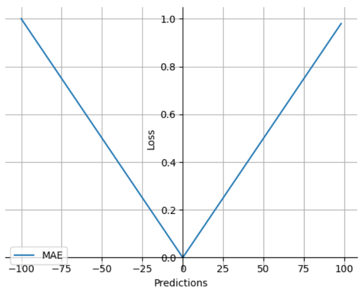
MSE calculates the sums of squares of the difference between predicted value $\hat{y}$ and the expected value y, as shown in the Equation 2.10. The range of *MSE* is $[0, \infty)$. Figure 2.4(a) shows the plot of *MSE*.

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.10}$$

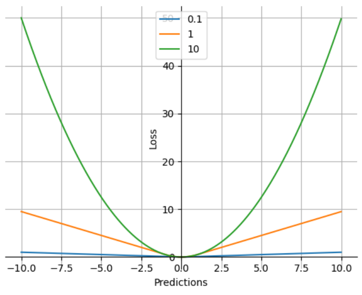
MAE calculates the sums of the absolute value of the difference between predicted value $\hat{y}$ and the expected value y, as shown in the Equation 2.11. The range of *MAE* is $[0, \infty)$. Figure 2.4(b) shows the plot of *MAE*.



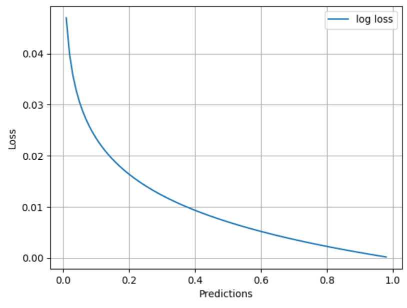
(a) MSE loss function



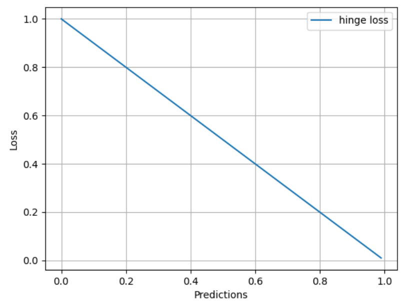
(b) MAE loss function



(c) Huber loss function



(d) Log loss function



(e) Hinge loss function

Figure 2.4 *Some loss functions*

$$L = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.11}$$

huber loss improves robustness of *MSE* to outliers. It uses δ to measure the form of itself. When $|y_i - \hat{y}_i|$ is less or equal than δ, *huber loss* is same as *MSE*. Otherwise, *huber loss* is linear error. It is shown in the Equation 2.12. Figure 2.4(c) shows the plot of *huber loss* and compares the plots when $\delta = 0.1, 1, 10$.

$$L_\delta(y,\hat{y}) = \begin{cases} \frac{1}{2}(y-\hat{y})^2, \text{if } |y-\hat{y}| \le \delta \\ \delta \cdot (|y-\hat{y}| - \frac{1}{2}\delta), \text{otherwise} \end{cases} \tag{2.12}$$

log loss is also called *Cross-entropy loss function.* It measures the performance of logistic regression, as shown in the Equation 2.12. Figure 2.4(d) shows the plot of *log loss.*

$$L = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)] \tag{2.13}$$

hinge loss is used in the maximum of margin, especially *Support Vector Machine (SVM)*, as shown in the Equation 2.14. Figure 2.4(e) shows the plot of *hinge loss.*

$$L = max(0, 1-y\hat{y}) \tag{2.14}$$

2.1.6 Optimization algorithms in ANN

The initial weights in FP are not the optimal ones in ANN. To get the better ones, ANN should be trained by optimization algorithms to update weights in each epoch. The goal of optimization algorithms in deep learning is minimizing the loss function. Choosing appropriate algorithms and hyperparamters improves the training efficiency. The optimization algorithms includes *gradient descent* (Lemaréchal 2010) and its related algorithms, e.g. *Batch Gradient Descent (BGD)*, *Adaptive Moment Estimation (Adam)*, etc.

In multivariable calculus "gradient" represents the vectorization of each parameters' partial derivatives, denoted as $\nabla f(x,y)$. At the point (x_0, y_0), the gradient is $\nabla f(x_0, y_0)$. It shows that, at the point (x_0, y_0), the function $f(x,y)$ increaes most rapidly in the direction of $\nabla f(x_0, y_0)$. In the reversed direction, the function decreases most rapidly.

Gradient descent obtains the minimum of lost function in the reversed direction of gradient. If the loss function is convex function, *gradient descent* will find the global optimum. If not, it will find global or local optimum. The learning rate controls the step of gradient descent towards to the local minimum. If the learning rate is small, the closer to the expected value, the slower the *gradient descent* moves, which means that more epochs will be needed in minimization of the lost function. Figure 2.5 shows that the loss changes based on the weights changes when the learning rate is small. The big learning rate causes the gradient descent bounces around the local minimum rather than reaches to it.

Batch Gradient Descent (BGD) uses all of the samples in training. *Stochastic Gradient Descent (SGD)* (Robbins 2007) chooses a sample and minimizes the loss

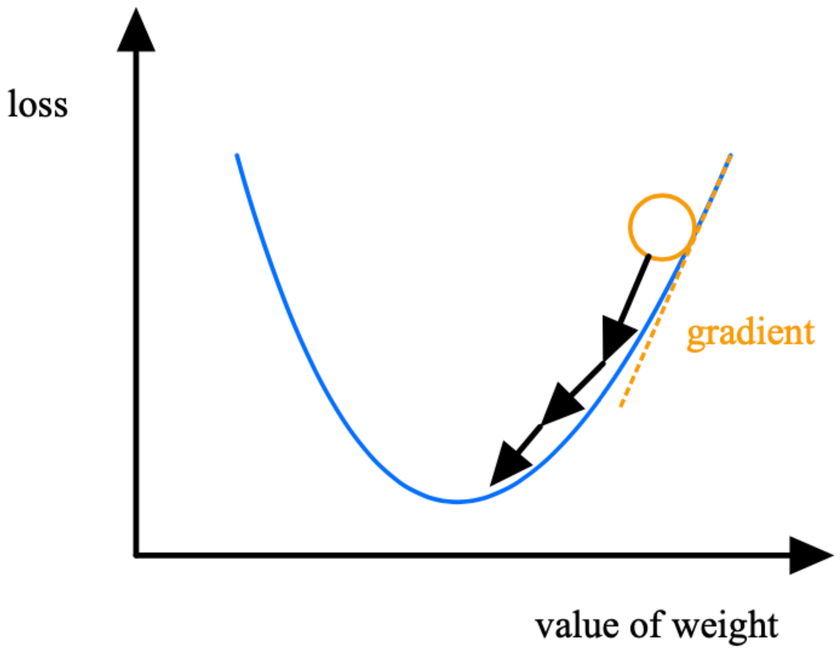


Figure 2.5 *The loss of gradient descent*

function of each sample. In each epoch, the optimization of *SGD* may not move to the global optimum accurately. Given all of the epochs, *SGD* moves to but may fluctuate around the global optimum. It is good at the large dataset compared to normal *gradient descent* and *BGD*. *Mini-batch Gradient Descent (MBGD)* takes the balance between *SGD* and *BGD*. It takes a mini-batch from all of the samples in each epoch to calculate gradient.

2.1.7 Back Propagation (BP)

The Back Propagation (BP) (Rumelhart, Geoffrey E. Hinton, and Williams 1988) and FP trains ANN so the ANN can correct the weights and biases in a loop to improve the accuracy of the prediction. The loss function measures this accuracy. It calculates the distance between the predicted output by ANN and the expected output. The more the distance is, the more error is. In FP, the input data goes through input layer, hidden layer and output layer. If the output does not meet the requirements, the predicted value $\hat{y}$ and the expected value y will be generated to loss function and fed into BP. In BP, all of the weights in ANN are calculated the loss function gradient layer by layer. This gradient is used in optimization algorithm to minimize the loss function by updating all weights in a loop. In each epoch of the loop, BP fine-tunes the weights and biases based on the loss value of the previous epoch. The training ends until the error meets the requirement of ANN.

In FP, the perceptron values, output and the error are calculated. They are put into BP based on the Chain Rule. The partial derivative of E with respect to w_i is calculated by the product of the partial derivative of E with respect to y and the partial derivative of y with respect to w_i, as shown in the Equation 2.15.

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial w_i} \tag{2.15}$$

To reduce the error between the predicted value $\hat{y}$ and the expected value y, the $\hat{y}$ should be updated by updating weights, as shown in the Equation 2.16, where w denotes weight. Updated weight is w^+. Error is E. The learning rate is η which means the step size of each epoch. When all of the weights are updated, the updated predicted output will be obtained based on the same way as Equation 2.16. The process of updating weight and the predicted value will be looped until the error meets the requirement of ANN.

$$w^+ = w - \eta \cdot \frac{\partial E}{\partial w} \tag{2.16}$$

2.2 Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) ((Lecun et al. 1998)) is a form of ANN which is good at processing two-dimensional data, e.g. images. Traditional CNN contains three layers: convolutional layer, pooling layer and fully-connected layer. Compared to MLP, CNN does not flatten all of the pixels of the image but keeps the spacial structure of the pixels, so the features of the images are saved. Besides, CNN can decrease the large amount of data of images to improve the efficiency of processing images and save the computation load.

2.2.1 Convolutional layer

The convolutional layer calculates pixels based on cross-correlation operation between input tensor and kernel tensor. In cross-correlation, each pixel value on the input tensor times corresponding value on the kernel tensor. Then, each product is added together as the first value of the output tensor. The cross-correlation operation causes the size of output less than the input tensor which means some pixels on the boundaries of the image are lost. To get the rid of pixels lost, extra zero value pixels are added around the image boundaries. Figure 2.6 shows the kernel moving on the padding input image and does cross-correlation. The blue area on the input image means zero padding. The result of calculation is put as the first value on the upper-left corner of the output image.

The step of a kernel tensor movement is controlled by a parameter, stride. It

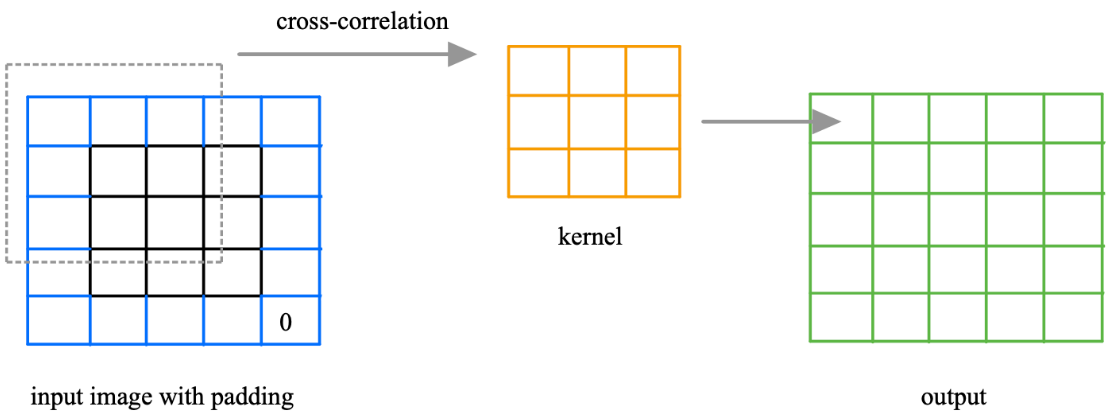


Figure 2.6 *The cross-correlation with padding*

controls the number of rows and columns passed per slide. The more the stride is, the less computation of cross-correlation is. If the stride is 1, the kernel tensor will move 1 column horizontally or 1 row vertically. Meanwhile, if the size of input tensor, kernel tensor are $i_h \times i_w$ and $k_h \times k_w$, the output tensor size is shown in the Equation 2.17.

$$(i_h - k_h + 1) \times (i_w - k_w + 1) \tag{2.17}$$

Given padding and stride, the size of the output tensor is changed from the Equation 2.17 to the Equation 2.18, where p_h, p_w are the padding horizontally and vertically respectively. The stride horizontally and vertically are s_h, s_w respectively.

$$\left\lfloor \frac{i_h - k_h + p_h + s_h}{s_h} \right\rfloor \times \left\lfloor \frac{i_w - k_w + p_w + s_w}{s_w} \right\rfloor \tag{2.18}$$

To make the size of the input tensor and the output one same, $p_h = k_h - 1$ and $p_w = k_w - 1$ so the Equation 2.18 is simplified to the Equation 2.19.

$$\left\lfloor \frac{i_h + s_h - 1}{s_h} \right\rfloor \times \left\lfloor \frac{i_w + s_w - 1}{s_w} \right\rfloor \tag{2.19}$$

In a convolutional layer, the kernel tensor starts moving from the upper-left corner of the input tensor. If it reaches the right side of the input tensor, it will move from the second row of the upper-left side. When the kernel tensor moves by one step, the cross-correlation operation is down by one time until it reaches the bottom-right corner. This process means that the kernel tensor filters the image and extract the local features. Finally, the convolutional layer generates the feature map of this image.

If the input has multiple channels d, the channels of kernel should be same as the input, d. The cross-correlation is calculated in each channels and then add them together. For example, color image has three channels: R, G, B so the channels of kernel should be 3. Figure 2.7 shows the cross-correlation computation with three input channels.

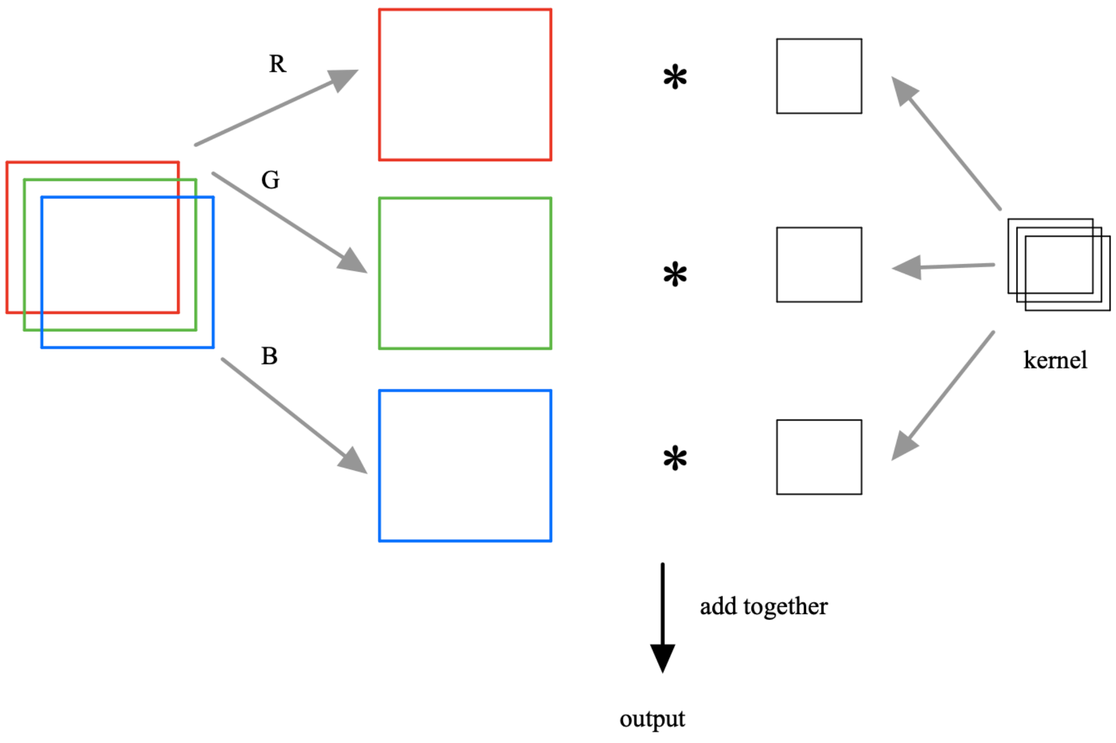


Figure 2.7 *Cross-correlation computation with three input channels*

2.2.2 Pooling layer

For the feature map generated by the convolutional layer, the pooling layer reduces its spatial resolution to decrease the load of computation of CNN. It also decrease the number of parameters during the calculation in CNN, so the computation efficiency is improved. As the number of layers increases in CNN, the receptive field of each perceptron is enlarged, which means the number of perceptrons in the former layers affects this one in FP will be increased. In this way, the perceptrons in the final layer will be sensitive to all of the input data.

In the pooling layer, the pooling window slides on the feature map from the upper-left corner to the upper-right corner based on specified stride with/without padding. In each step of a pooling window moving, it calculates the corresponding elements on the feature map. Then, this window starts sliding from the left side to the right side on the next row back and forth until it reaches the bottom right corner. Finally, the output image is generated.

There are many kinds of calculation ways of pooling layers: maximum pooling, average pooling, stochastic pooling, etc. For the maximum pooling, the pooling window selects the maximum value from the corresponding elements on the feature map. For the average pooling, the pooling window calculates the average value. For the stochastic pooling, the pooling window selects the element based on its value. The bigger the element is, the high probability this element is selected. Figure 2.8 shows the process of 2×2 max pooling. Each color represents each part which does max pooling. If the feature map has multiple channels d, the channels of pooling window should be same as the input, d. The pooling is operated in each channels.

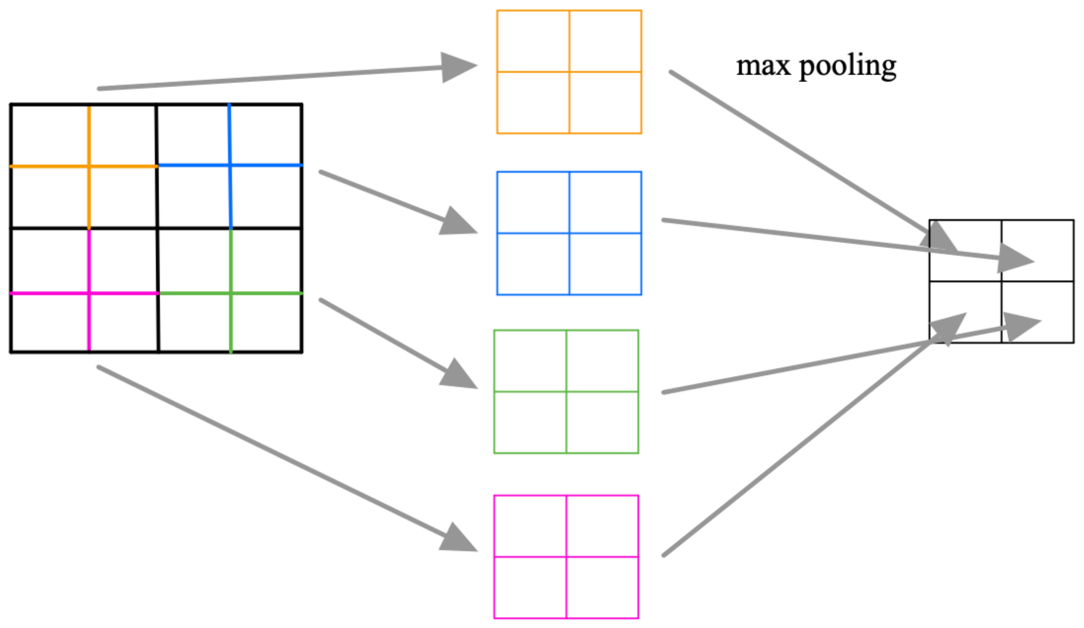


Figure 2.8 *The process of* 2×2 *max pooling*

2.2.3 Common CNN models

During the development of CNN, many kinds of CNN models were proposed and developed. *LeNet* is one of the earliest CNN model which was developed by Yann LeCun at Bell Labs in 1989. It was used in recognizing the hand written numbers. The Final version of *LeNet* was called *LeNet-5* (Lecun et al. 1998). Figure 2.9 shows the structure of LeNet-5.

The input image has 32×32 pixels, only one channel. Except for the input layer, *LeNet-5* comprises seven layers: 3 convolutional layers, 2 average pooling layers and 2 fully-connected layer. The first layer is a convolutional layer. It has six 5×5 kernels and generates six 28×28 pixels feature maps. Its activation function is *tanh*. Its stride is 1. The second layer is a pooling layer whose window's dimension

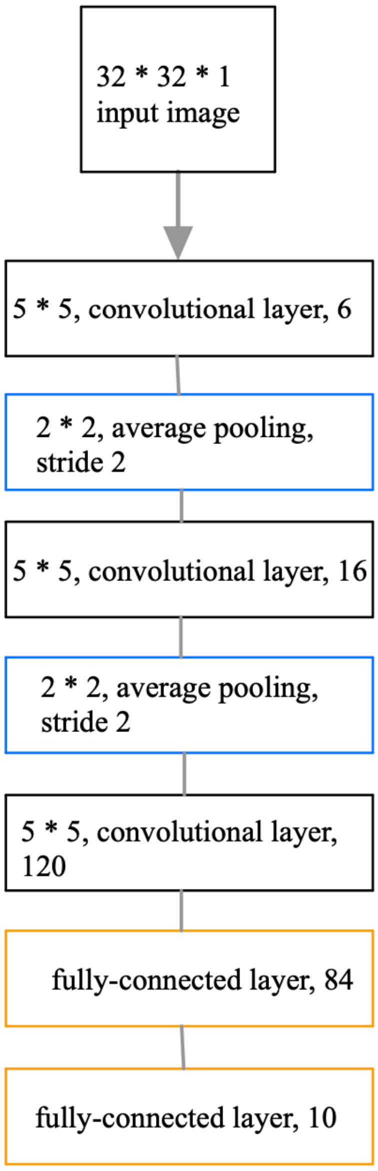

32 * 32 * 1
input image
5 * 5, convolutional layer, 6
2 * 2, average pooling,
stride 2
5 * 5, convolutional layer, 16
2 * 2, average pooling,
stride 2
5 * 5, convolutional layer,
120
fully-connected layer, 84
fully-connected layer, 10

Figure* *2.9 *The structure of LeNet-5*

is 2×2 with the stride 2. It generates $14 \times 14 \times 6$ feature maps.The thrid layer is also a convolutional layer which is similar as the first one. It generates of $10 \times 10 \times 16$ feature maps. The forth layer is a pooling layer. The fifth layer is similar as the first one but it generates 120 feature maps. The sixth and seventh layers are all fully-connected layers. The activation function of the first one is *tanh* and the second one is *Softmax*.

Except for *LeNet-5*, there are other kinds of CNN models. In 2012, Alex Krizhevsky and other researchers proposed *AlexNet* (Krizhevsky, Sutskever, and Geoffrey E Hinton 2012). Its structure is similar as *LeNet-5* but with deeper layers. It uses multiple convolutional layers to extract features. *ZF Net* was proposed by Zeiler and Fergus in the competition ILSVRC 2013 (Zeiler and Fergus 2013). It improves *AlexNet* by changing the hyperparameters. In the next year of ILSVRC, Simonyan and Zisserman designed and proposed *VGGNet* (Simonyan and Zisserman 2014). They used 16 layers including convolutional layer and fully-connected layer. The dimensions of convolutional kernels and pooling windows are 3×3 and 2×2 respectively. In ILSVRC 2015, Kaiming He proposed *ResNet* (He et al. 2015). He implemented skip connections and batch normalization in this model. Also, it does not have fully-connected layer.

2.3 Game Engine

2.3.1 Game engine and its subsystems

Game engine is a software framework which provides an integrated environments for game developers. It provides different kinds of built-in tools to simplify and accelerate game development so game developers can concentrate on game design and the interaction between the game and players. The engines could be operated on single or different operating systems, e.g. Windows, Linux, etc. The developed games could be operated on hardware, e.g. personal computers (*PC*), *PlayStation*, *Xbox*, *Nintendo Switch*, etc. The popular game engines in the world are *Unreal Engine*, *Unity3D*, *Cocos*, etc.

Game engine is a complex system consists of many subsystems, e.g. input management, audio management, physics engine, etc. Table 2.1 shows the main subsystems and their main functions of a game engine.

Table 2.1 *Main subsystems of game engine*

subsystems	**main function**
input management	recognize the users' commands, update objects' data and properties
audio management	generate background music and sound effects
collision management	detect collision among objects
animation management	simulate movement of characters
network management	synchronize all players' status via network
2D/3D graphics management	display 2D/3D graphics on the screen
resources management	maintain resources, load assets, avoid performance decreasing
physics engine	simulate laws of physics
artificial intelligence	implement artificial intelligence in games
graphics interface	manage the GUI of the game

2.3.2 Unreal Engine

Unreal Engine is one of widely used game engines in the world. It was developed by *Epic Games*. The first generation of *Unreal Engine* was released in 1998 and the latest generation, *Unreal 5* will be shipped in early 2022. In this thesis, our work is implemented in *Unreal 4*.

Unreal 4 is widely used in game development, film making, medical simulation, car industry, etc., because it has excellent performance in pipeline integration, real-time rendering, simulation, etc. For pipeline integration, *Unreal 4* supports different kinds of industry standards, e.g. Universal Scene Description (USD), Filmbox (fbx). It reads or writes the files in these formats with lower time consuming. One of plug-ins, Datasmith helps game developers import metadata from other software quickly. One of the visual tool, Visual Dataprep decreases repetitive data preparation. For rendering, *Unreal 4* has Path Tracer, which could decrease the number of setups in rendering. It also has Forward Rendering, which is good at rendering for VR experiences. Moreover, Movie Render Queue in *Unreal 4* could generate high-quality rendered images for movie rendering works. For simulation, *Unreal 4* has both Cascade and Niagara to make visual effects (VFX). In Cascade, data could not be shared between particle systems and the other parts in the engine. Also, customizing modules in Cascade is not user-friendly. However, the next generation of VFX, Niagara overcomes these weak points so it makes VFX system flexible and easy to use. Moreover, *Unreal 4* uses NVIDIA's NvCloth solver which is good at particle simulation for cloth.

Unreal Engine provides Blueprint Visual Scripting system, which could setup variables on the node-based interface. Developers could define object-oriented classes or objects and create customized character, events, functions by Blueprint without coding by themselves. They can also interact with C++, e.g. calling functions in C++, inheriting C++ classes, etc. For the game developers who can not program in C++, *Blueprints* helps them program easily. To realize a complex gameplay element or construct an object by Blueprint, developers could connect different nodes, events, functions and variables together with lines.

2.3.3 Skeleton and skeleton hierarchy

In most 3D applications, a skeleton is a digital hierarchical framework that is used to define bones or joints in a character and in many ways mimics a real biological skeleton (Unreal Engine 4.27 documentation 2021). A skeleton is composed of bones and joints. For bones, they represents the visible relationship between joints and they are not calculable. For joint, there are three types of joints: ball joint, universal joint and hinge joint. Each joint might has multiple sub-joints and each joint might connects multiple bones. Joints and bones are connected linearly to generate joint chain, which represents the anatomical positions of the joints of creatures in 3D applications. It always starts at the highest joint and ends at the final bones of the joint chain. In some 3D applications, e.g. *Unreal Engine*, joints' attributes could be changed so people could control the range of rotations of joints.

Skeleton hierarchy is composed of joints and joint chains with hierarchical relationship. It always starts from the only one root joint. For the joints on the higher skeleton hierarchy, it is called parent joint so the lower ones are child joints. The repetitive structure of joint chain shows that a bone always starts from a parent joint and points at the child joint. Hence, parent joint or child joint are relative concepts. If the parent joint is rotated, all of the connected joint chains will be rotated as the same degree of it. In *Unreal Engine*, Skeleton Tree shows the skeleton hierarchy. People could check a specific bone or all of the bones on the same hierarchy.

2.3.4 Skeletal Animation

Skeletal Animation is driven by bone and skinned mesh. Similar as the biological skin and bones, in 3D game development, skin is composed of vertices and polygon surfaces. Also, each bone controls the shape and position of the skin. Each vertex might be controlled by multiple bones. Because of this, the mesh does not has gap on the joint when the parent bone and the child bone are stretching the same vertex. For skinned mesh, the mesh has bone weighting. According to the skeleton hierarchy, when developers change the position and orientation of the parent bone, all of the child bones see this bone as the origin and they will be translated or rotated or scaled according to it. Skeletal Animation means that the skinned mesh is animated based on the skeleton animation.

2.3.5 Inverse Kinematics

Inverse Kinematics (IK) describes the behavior that when a child node moves, the parent nodes will be moved or rotated according to the child node. In contrast, Skeletal Animation implements Forward Kinematics (FK) which means changing the parent bone to control the child bones. However, FK saves the data of position and rotation of each bone which takes much memory. Also, this data is only available for a specified skeleton. The *Unreal 4*, provides an algorithm (IK solver) to control to rotation in the Inverse Kinematic systems. A target is given to a joint chain. The end of this chain tries to achieve this target during the movement. For the skeleton with IK, developers control the all of the bones' targets. The IK system determines how each bone react their environment.

3 LITERATURE REVIEW

The literature review introduces some solutions of hand pose estimation selected from the *HIM2017* Challenge: V2V-PoseNet, Hand PointNet and Pose-REN. Moreover, an on-device and real-time hand tracking solution via MediaPipe is introduced.

3.1 The survey of depth-based hand pose estimation

Popular solutions of three tasks in hand pose estimation are discussed from the *HIM2017* Challenge (Yuan et al. 2017). One of the tasks, single-frame 3D pose estimation is related to the topic of this thesis. The existing problems in this task are the occlusion caused by hands or objects and the variation of hand shape. As the Table 3.1 shows, the selected popular solutions used the methods: 3D CNN (3D), detection-based methods (De), hierarchical models (Hi), structured models (St), multi-stage models (M) and residual models (R). Also, each popular solutions in *HIM2017* was evaluated to find the cases of success and failure in Mean errors (in mm). This paper evaluates top 10 solutions by analyzing the errors: joint visibility (occluded/visible), subject was visible or invisible during training (seen/unseen). In this thesis, the method of *V2V-PoseNet*, *oasis* and *THU_VCLab* are researched.

Table 3.1 *Example of solutions of single-frame 3D hand pose estimation.*

method	model	input	3D	De	Hi	St	M	R
V2V-PoseNet	3D CNN	$88 \times 88 \times 88 voxels$	Y	Y	N	N	N	Y
oasis	hierarchical PointNet	1024 3D points	N	N	N	Y	N	N
THU_VCLab	Pose-REN	96×96	N	N	Y	N	Y	Y

3.1.1 The analysis of single-frame 3D hand pose estimation result

The top 10 solutions of single-frame 3D pose estimation are analysed in these aspects: annotation error, hand joints' occlusion and unknown subject in the training set, view point and articulation. They are compared based on the annotation error, occluded or unknown subject, occluded joints' number, view point, articulation and joint type.

First, annotation error is caused by the measurements during annotation. This paper qualified this error and found that the three selected methods' outputs are close to ground truth.

Second, the paper compared each method in the condition of occlusion or unknown subject. Hierarchical method is good for occlusion but it is unimportant in their project. Next, the average of top two detection-based methods accuracy is better than regression-based ones. Then, cascaded methods outperform single-stage ones. After that, some structural methods outperform differently in different error thresholds. Given the simple hand gestures in the thesis, these methods will be concerned if they perform well in other aspects. For the dimensions of CNN, 2D is simpler than 3D but 3D can be used if its accuracy is higher than 2D.

Third, the view point means that the angle between camera orientation and the palm. It is not extreme for the thesis work so the suitable normal range is $[70, 120]$. The degree of hand articulation is concerned in this range. In this paper, *V2V-PoseNet* has the lowest average error. The *oasis* is the second one, lower than *THU_VCLab*.

Fourth, in this thesis work, the bending angles of fingers are unpredictable. Given the view point range, $[70, 120]$ on the third point, this thesis chooses the methods which has lowest average error: *V2V-PoseNet* < *oasis* < *THU_VCLab*.

Fifth, in this thesis work, the joints of the hand are all visible and the presence of the subject in the training set is seen.

The three methods: *V2V-PoseNet*, *oasis* and *THU_VCLab* perform well in the analysis of these aspects, so they are selected to research in this thesis.

3.2 Voxel-to-voxel Prediction Network (V2V-PoseNet)

There are two disadvantages in many existing 3D hand gesture recognition methods (Chen et al. 2020; Fourure et al. 2017). First, the input of them is the depth maps. which are actually 3D data but they input them as 2D images. This deforms the shape of hand gestures. Second, they also map 2D depth images to 3D coordinates of hand gestures' keypoints directly. This mapping is highly non-linear. Additionally, they are affected by the poor quality of images, self-occlusion in input images, etc. A new form of a model, *voxel-to-voxel prediction network for pose estimation (V2V-PoseNet)* was proposed to overcome these two disadvantages (Moon, Chang, and Lee 2018).

3.2.1 The comparison of V2V-PoseNet and other 3D hand gesture recognition models

The target of both traditional and V2V-PoseNet is estimating the 3D coordinates of all hand keypoints. For a 2D gesture image, the existing 3D hand gesture recognition methods input it to a 2D CNN and output the 3D coordinates of hand gestures' keypoints.

In contrast, the model of V2V-PoseNet does a voxel-to-voxel prediction so it was designed as 3D CNN. The authors first project the points in the 2D image to 3D space and discretize this space based on the size of the voxel they defined before. Second, the authors followed the method (Oberweger and Lepetit 2017). They calculated the reference point of the 2D hand depth map and put this map to a 2D CNN. This network outputs the offset from the calculated reference point to the center of ground-truth joint locations (Moon, Chang, and Lee 2018). They added this offset to the reference point as a refined reference point. Third, the authors drew a cubic box around the refined reference point so that they could extract the target hand. If the depth point is in the voxel, the authors would assign the voxel value to 1. If not, the value would be assigned to 0. Hence, they changed this 2D image into a form of 3D voxelization as the input of 3D CNN. This processing gets rid of perspective deformation. The output would be each voxel's probability for each gesture's keypoints, which would decrease the difficulty of the learning task. The voxel with the highest probability for each gesture's keypoints would be extracted. Then, this voxel would be changed to the form of real-world coordinates as the ultimate result of V2V-PoseNet. Figure 3.1 shows the structure of V2V-PoseNet and many existing 3D hand gesture recognition.

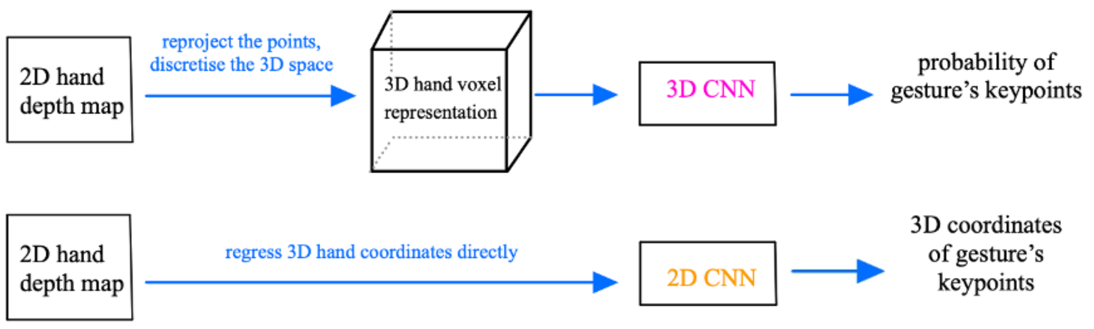

2D hand depth map
reproject the points, discretise the 3D space
3D hand voxel representation
3D CNN
probability of gesture's keypoints
2D hand depth map
regress 3D hand coordinates directly
2D CNN
3D coordinates of gesture's keypoints

Figure 3.1 *(top) The process of 3D hand gesture recognition using V2V-PoseNet, (bottom) the process of many existing 3D hand gesture recognition.*

3.2.2 The network architecture of V2V-PoseNet

The network architecture of V2V-PoseNet is based on the "stacked hourglass" network (Newell, Yang, and Deng 2016). Its structure is shown on the top of Figure 3.2. The first block in the network is a volumetric basic block, whose activation function is ReLU (Xu et al. 2015). It also contains a volumetric convolution and volumetric batch normalization (Ioffe and Szegedy 2015). After that, a volumetric downsampling block, which is a volumetric max-pooling layer downsamples the feature map.

Then, the output goes through three volumetric residual blocks one by one to extract local features.

The structures of the encoder and decoder are shown on the bottom of Figure 3.2. The encoder has two connected groups of volumetric downsampling block and volumetric basic block. In contrast, the decoder has two connected groups of volumetric upsampling block and volumetric basic block. The volumetric downsampling block decreases the spatial size of the feature map but the upsampling one enlarges it. To make the progress of upsampling in decoder stable, encoder and decoder are connected with voxel-wise addition. The volumetric residual blocks in the encoder and decoder increase the channels' number.

After the decoder, the output is processed by two consecutive volumetric basic blocks. Then, The probability of the gesture's keypoints would be obtained from a volumetric convolutional layer.

The output of V2V-PoseNet is generated to the 3D heatmap, so people can check it easily.

$$H_n^*(i,j,k) = exp\left(-\frac{(i-i_n)^2 + (j-j_n)^2 + (k-k_n)^2}{2\sigma^2}\right) \tag{3.1}$$

In the Equation 3.1, $H_n^*(i,j,k)$ denotes the ground-truth 3D heatmap and n means nth keypoint. The standard deviation is $\sigma = 1.7$. For nth ground-truth keypoint, (i_n, j_n, k_n) is the mean of Gaussian distribution.

The loss function of V2V-PoseNet is MSE.

$$L = \sum_{n=1}^{N} \sum_{i,j,k} ||H_n^*(i,j,k) - H_n(i,j,k)||^2 \tag{3.2}$$

3.3 Hand PointNet

The performance of 3D gesture estimation will be weaken for these reasons. First, the accuracy will decrease because high dimensions of 3D gesture, hand orientations variety, similar fingers and their occlusions. Second, the resolution of input will rise the time and space complexity of 3D CNN. Third, sparse 3D point cloud will increase the computation load of 3D convolution. To solve these problems, a way was found to learn 3D hand articulations from 3D point cloud based on the *hierarchical PointNet* and *Fingertip Refinement Network*, which are the improved version of the basic *PointNet* (Ge, Y. Cai, et al. 2018).

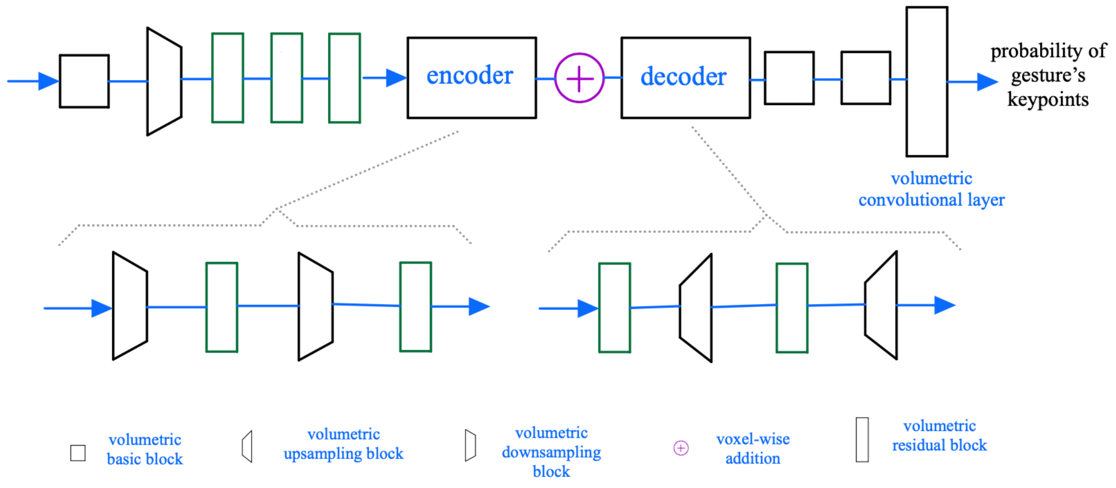

encoder
decoder
probability of gesture's keypoints
volumetric convolutional layer
volumetric basic block
volumetric upsampling block
volumetric downsampling block
voxel-wise addition
volumetric residual block

Figure 3.2 *(top) The structure of V2V-PoseNet, (bottom left) the structure of encoder, (bottom right) the structure of decoder*

3.3.1 The process of Hand PointNet gesture estimation

The authors change a depth image of a hand to a form of 3D point cloud. Then, the authors use oriented bounding box (OBB) whose orientation is determined by principal component analysis (PCA) to perform downsampling and normalization of point cloud. The goal of normalization is improving the authors' method robustness to varieties of hand orientation. The normalized points with surface normals will be the input of *hierarchical PointNet*. This network extracts the features of the hand and output is 3D hand joint locations but with low dimension. The *Fingertip Refinement Network* takes the k nearest neighbouring fingertip locations and get the 3D hand structures. This network generates the refined their 3D locations as result. The structure of the whole process is shown on the Figure 3.3.

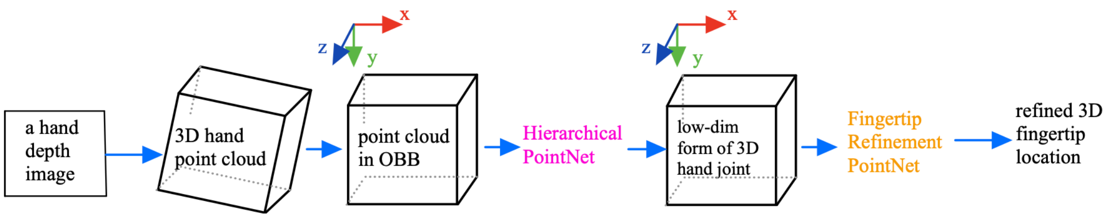

a hand depth image
3D hand point cloud
point cloud in OBB
x
y
z
Hierarchical PointNet
low-dim form of 3D hand joint
Fingertip Refinement PointNet
refined 3D fingertip location

Figure 3.3 *The process of 3D hand pose estimation using Point Sets to a hand depth image*

3.3.2 Hierarchical PointNet

The basic PointNet is a type of neural network, which inputs a group of points and outputs the features of this point cloud. Given enough neurons in the network (Ge, Y. Cai, et al. 2018), it was proved that PointNet has the ability to approximate arbitrary continuous set functions (Qi et al. 2016). To improve the performance of this network in the extraction of the local structures a hierarchical way, the authors design *hierarchical PointNet*. Figure 3.4 shows the structure of the *hierarchical PointNet*.

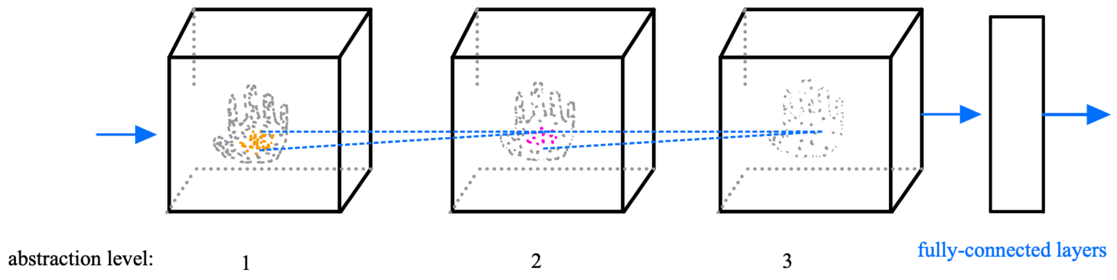


Figure 3.4 *The structure of Hierarchial PointNet*

The *hierarchical PointNet* has three abstraction levels. The more abstraction levels, the less point cloud density. For the first level, N_1 points would be chosen as the centroids of an area and the k nearest neighbouring (kNN) of them would be grouped as a new area. Then, the PointNet would capture the C_1-dim feature of each area. Finally, the features, centroids, and d-dim coordinates would be transferred to the second level. In the second and third level, the same steps would be looped but N_1, C_1 would be changed to N_2, C_2 and N_3, C_3. After the last level, authors would use basic PointNet to abstract a global point cloud feature from all of the input points of the third level.

3.3.3 Fingertip Refinement Network

The estimation accuracy of basic PointNet on fingertip is lower than the estimation on other joints. Also, the authors could refine straightened fingers' locations are easily. Hence, the authors designed *Fingertip Refinement Network* based on basic PointNet which only focuses on straightened fingers to better the performance of estimation for fingertip locations.

The process of *Fingertip Refinement Network* starts with the judgement that if the finger straightened or not. If the authors find them, they would find the kNN points of fingertip location from the 3D point cloud. Then, they normalize them in OBB as the input of the *Fingertip Refinement Network* and outputs the refined fingertip 3D location.

3.4 Pose Guided Structured Region Ensemble Network (Pose-REN)

3.4.1 Overview of Pose-REN

CNN-based hand pose estimation methods are implemented based on the prediction of the hand joints heatmaps or the regression of hand joints coordinates (Ye, Yuan, and Kim 2016; Wan et al. 2017). However, more optimal features of CNN can not be obtained by these methods for hand pose estimation. To solve this problem, *Pose Guided Structured Region Ensemble Network (Pose-REN)* was proposed, which is built based on the cascaded framework (Chen et al. 2020). It estimates 3D hand pose from a depth image. Compared to the work (Sun et al. 2015), which trains multi models to refine hand pose iteratively, Pose-REN only uses one model and updates feature maps during each iteration.

The whole process of Pose-REN includes initialization of a simple CNN (Init-CNN), pose guided region extraction and structured region ensemble respectively. The authors first input a initial hand pose, $pose_0$ from a depth image to a Init-CNN as the initialization. Figure 3.5 shows this process.

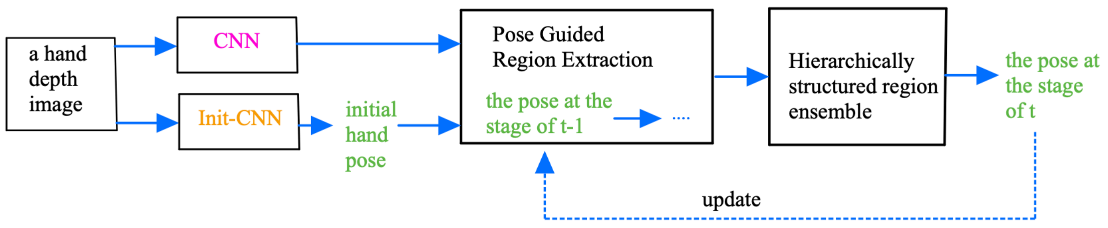

a hand depth image
CNN
Init-CNN
initial hand pose
Pose Guided Region Extraction
the pose at the stage of t-1
....
Hierarchically structured region ensemble
the pose at the stage of t
update

Figure 3.5 *The whole process of Pose-REN*

The authors input previously estimated pose, $pose_{t-1}$ and the output is the refined hand pose, $pose_t$. In Equation 3.3, P denotes the 3D locations of J hand joints. A cascaded stage of each iteration is t. A depth image is represented as D. The regression model is R.

$$P^t = R(P^{t-1}, D) \tag{3.3}$$

Pose-REN can extract feature regions and generates the features of different joints of different fingers. The authors used structured connection to regress the $pose_t$. They iterated the process of generating $pose_t$ to update the $pose_{t-1}$ to get more accurate result. The Equation 3.4 shows the final estimated hand pose P^T will be obtained at the stage T.

$$P^T = R(P^{T-1}, D) \tag{3.4}$$

3.4.2 Pose guided feature extraction method

This stage is the highlight of this paper. Feature maps are generated by feeding a depth image into a CNN with residual connections. In this CNN, there is a a Rectified Linear Unit (ReLU) following a convolutional layer. This group is repeated twice and there are max pooling layers following these two groups. Between two max pooling layers, there are the residual connections. The feature map, F of the last convolutional layer is going to be extracted to generate feature regions. This process is guided by the the estimated hand pose from previous stage, $P_{t-1} = \{(p_{xi}^{t-1}, p_{yi}^{t-1}, p_{zi}^{t-1})\}_{i=1}^{J}$.

For the i^{th} hand joint, the authors use the intrinsic parameters of the depth camera to project the real world coordinates into the pixel coordinates of image, as shown in the Equation 3.5.

$$(p_{ui}^{t-1}, p_{vi}^{t-1}, p_{di}^{t-1}) = proj(p_{xi}^{t-1}, p_{yi}^{t-1}, p_{zi}^{t-1}) \tag{3.5}$$

Then, the authors use a rectangular window to crop the feature region for the i^{th} hand joint. They normalize and convert $(p_{ui}^{t-1}, p_{vi}^{t-1}, p_{di}^{t-1})$ into coordinates of feature maps to obtain this window's coordinates. Afterwards, they get the feature region, as shown in the Equation 3.6. The coordinates of the top-left corner are b_{ui}^{t} and b_{vi}^{t}. The feature region's width and height are w and h respectively.

$$F_i^t = crop(F; b_{ui}^t, b_{vi}^t, w, h) \tag{3.6}$$

3.4.3 Structured region ensemble

When the authors get the feature regions of each joints of each fingers, they use hierarchically structured region ensemble to model each joint's constraints, because human hand has many constraints and correlations between different joints (Lin, Wu, and Huang 2000). Figure 3.6 shows the structure of this strategy. Different color of "feature region.T" or ".R" represent they are different regions.

The authors first input the feature regions into fully connected layers respectively. Then, they integrate the output of fully connected layers hierarchically. After that, the authors concatenate the features of each finger and the output would be transferred to the fully connected layer. The final hand pose would be regressed.

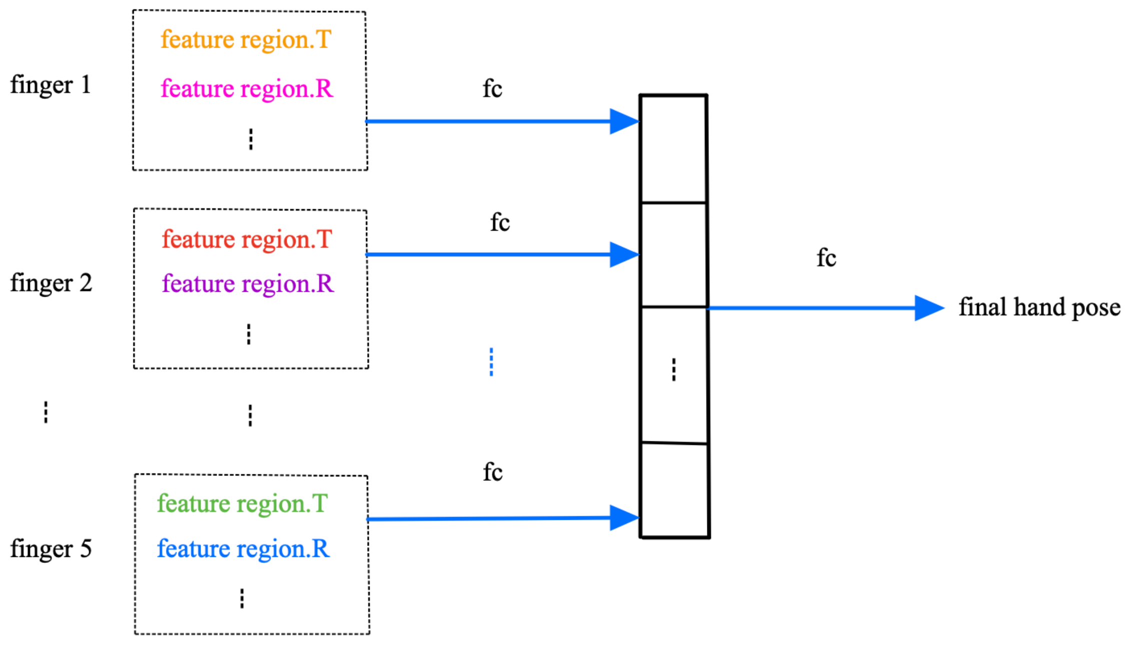


Figure 3.6 *The structure of the structured region ensemble*

3.5 On-device and real-time hand tracking via MediaPipe

Existing hand pose recognition methods either require specialized hardware (e.g. Ge, Liang, et al. 2018) or can not be real-time (e.g. Ge, Ren, et al. 2019). A new solution was proposed to overcome this limitation (Zhang et al. 2020). They designed a hand pose estimation model which includes two parts: palm detector and hand landmark model. The palm detector takes a RGB image and generates a hand image inside a bounding box. The hand landmark model receives the palm detector's output and predicts the 2.5D landmarks of this hand. Figure 3.7 shows this process. This solution is implemented by MediaPipe, which can build machine learning methods on different kinds of platforms. Hence, this solution can track hand gestures in real-time on mobile devices.

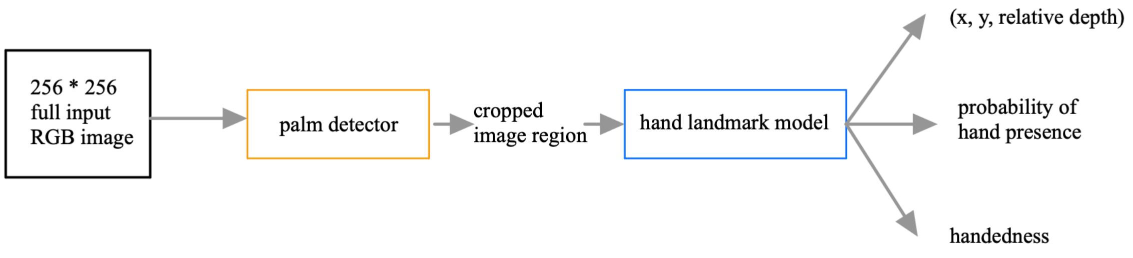


Figure 3.7 *The structure of on-device and real-time hand tracking via MediaPipe*

3.5.1 Palm detector

The existing problems for hand tracking are: first, hands have different sizes. Second, some of the images of hand are occluded or self-occluded. Third, compared to face, hands have less distinct features which is hard to detect them. The proposed method via MediaPipe designs a palm detector, using encoder-decoder feature extractor and minimizing focal loss to solve these problems.

Compared to hand detector, palm detector is simpler. A hand has a palm with five fingers. The different movement of fingers makes this hand hard to detect, but the palm has simple shape so it is easy to estimate the a square bounding box (W. Liu et al. 2016). For the palm detector, it receives a full input image and uses an oriented hand bounding box to locate palms.

3.5.2 Hand landmark model

The palm detector generates the cropped image by hand bounding box and transfers it to the hand landmark model, which decreases the need for data augmentation. The hand landmark model learns hand pose representation. In real-time tracking, the landmark prediction on the last frame generates the bounding box. Then, this box is inputted to this frame. In this way, the detector would not be applied on each frame. The hand landmark model uses regression to output a hand skeleton inside a detected hand region which consists of 21 2.5 D landmarks.

The hand landmark model generates three kinds of outputs. The first output is a group of 21 hand landmarks. Each landmark is represented as $(x, y, relativedepth)$. The authors used two datasets to learn the hand pose representation. The dataset of real-world images learns the (x, y) in $(x, y, relativedepth)$. The second dataset, synthetic dataset was created by authors. They sampled one hundred thousand images from a video of hand pose. Then, they rendered hand poses on the different backgrounds and mapped it to the corresponding 3D coordinates. The $relativedepth$ in $(x, y, relativedepth)$ is learned from the synthetic dataset. The second output is probability of the aligned hand presenting on the cropped images. The authors referenced the work, which the probability works as a threshold (Simon, Joo, I. Matthews, et al. 2017). If the score is lower than this threshold, the tracking would be reset. The third output is handedness, which is binary classification: right or left hand.

3.6 Residual Learning Framework (ResNet)

The Residual Learning Framework (ResNet) was proposed to decrease the difficulty of deep neural networks (He et al. 2015). The highlight of this paper is that the layers of ResNet learn residual functions. The advantages of ResNet are optimizing easily and it gets the accuracy by increasing depth of the neural networks.

There is the problem of degradation of training accuracy during the neural network converging. In this situation, the accuracy gets saturated when the neural network becomes deeper. Then, the accuracy would degraded greatly. Given this problem, the researchers proposed ResNet. In this model, every stacked layer fit for residual mapping. If x denotes the input of the layers considered and the desired underlying mapping is denoted by $H(x)$, the residual function could be approximated by nonlinear layers. The Equation 3.7 shows this approximation.

$$F(x) := H(x) - x \tag{3.7}$$

The Equation 3.7 can be reformulated and get $F(x) + x$, which helps the precondition the problem of not optimal identity mappings. This part was realized by ”shortcut connections” in feedforward neural network (Bishop 1995). The ”shortcut connections” implements identity mapping and add the output to the output of stacked layers. Moreover, the ”shortcut connections” is good for training networks because it does not require many parameters and the computation complexity is not much.

The output vectors of the stacked layers considered is denoted $\mathbf{y}$ and the input vectors is $\mathbf{x}$. The Equation 3.8 shows how y is obtained, where $F(\mathbf{x}, \{W_i\})$ is the residual mapping.

$$\mathbf{y} = F(\mathbf{x}, \{W_i\}) + \mathbf{x} \tag{3.8}$$

ResNet derived from the plain network and the plain network was designed based on VGG nets (Simonyan and Zisserman 2014). The Figure 3.8 shows the architecture of VGG, plain network and ResNet. In the plain network, the size of convolutional layer is 3×3 and it is downsampled with the stride of 2. This network has number 34 weighted layers and there are global average pooling and fully-connected layer in the end of the network. The architecture of ResNet is similar as the plain network but ”shortcut connections” are inserted to the plain network and this network becomes counterpart residual version.

3.7 High-Resolution Network (HRNet)

High-Resolution Network (HRNet) was proposed to solve the problems of position-sensitive vision (Wang et al. 2019). Compared to some existing models which transform between high and low resolution, HRNet can keep high-resolution representation through from the start to the end of process. HRNet has three versions: HRNetV1, HRNetV2 and HRNetV2p. My thesis work uses HRNetV2. This model

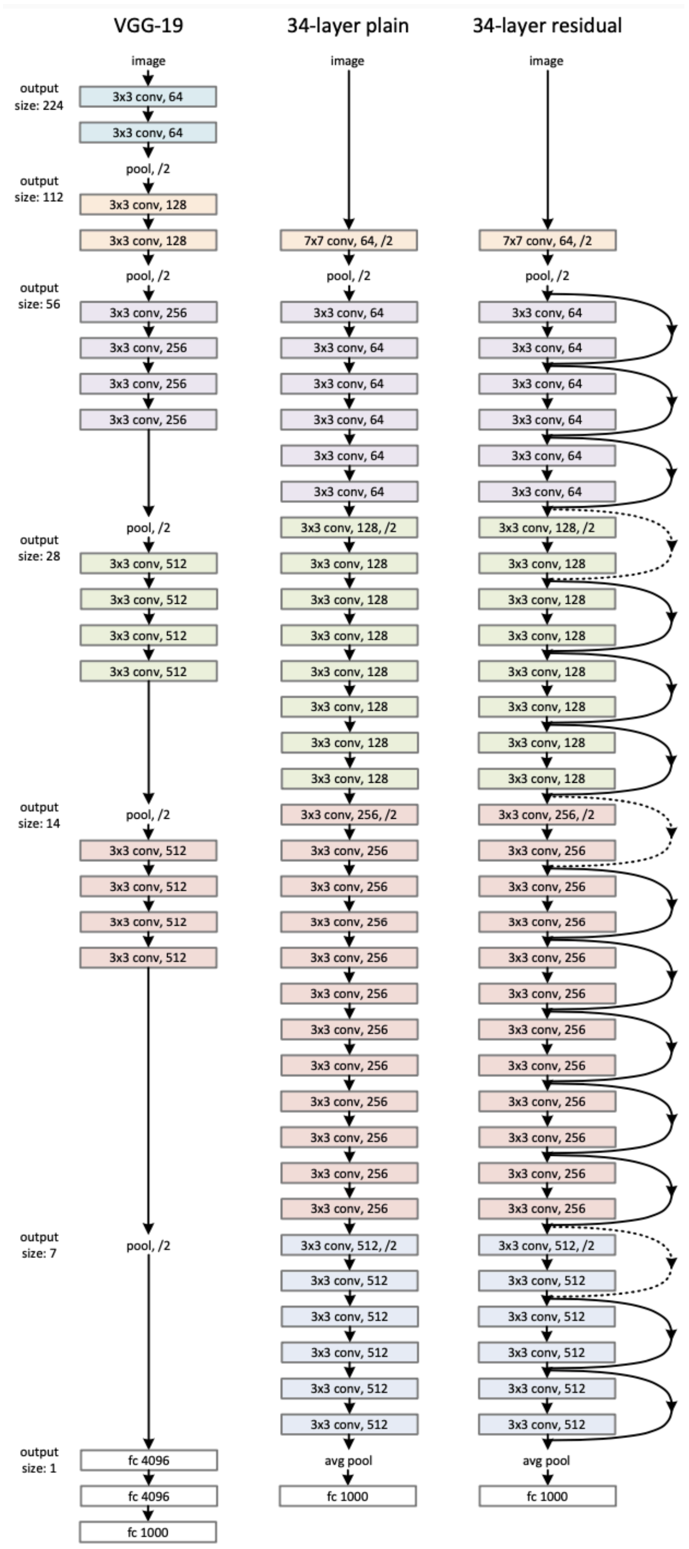


Figure 3.8 *The structure of VGG nets, plain network and ResNet (He et al. 2015)*

combines the resolution streams from high to low and turn the combination to semantic segmentation.

The input image is fed to a stem first. The stem has 3×3 convolutional layer with the stride 2. This stem decreases the resolution to 1/4. Then, the stem transfers its output to the main body, which consists of three steps.

The first step in the main body is parallel multi-resolution convolutions, as shown by Figure 3.9. This process starts from a high-resolution stream. Then, in each stage, resolution convolution streams are added from high to low. This form of representation is semantically strong. After that, those streams are connected in parallel, which can maintain high-resolution. The repeated fusions of these resolution helps high-resolution representations. The result of this process has many stages. In the result, n stages have n streams. Each stream corresponds to n resolutions.

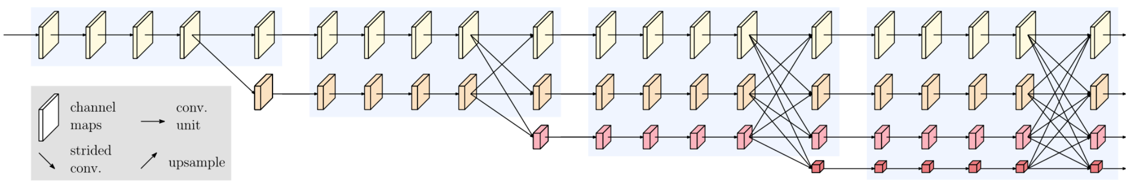


Figure 3.9 *The high-resolution network (Wang et al. 2019)*

The second step is repeated multi-resolution fusions keeps information exchanging among different resolution streams repeatedly.

The third step is representation head. The low-resolution is re-scaled to high-resolution by bilinear upsampling. In this process, the number of channels is not changed.

The instantiation of main body has 4 stages and each stage has its parallel convolution stream. The resolution of first stream is 1/4. The stream has 4 residual units and each unit has bottleneck and 3 × 3 convolution. The rest of streams' resolutions are 1/8, 1/16 and 1/32 respectively. They have 1, 4 and 3 modularized blocks respectively. Each block has 4 residual units and each unit has two 3 × 3 convolutions for each resolution. The resolution of the main body's output is 1/4.

4 EVALUATION OF HAND MODELS FOR A SINGLE IMAGE AND IN REAL-TIME APPLICATION

This thesis mainly researched getting 2D coordinates of hand joints from depth image and RGB image. Depth image is acquired using depth camera, and then the depth of object in the image is stored. However, the common cameras people often use don't have functions to store the depth information of object in an image. Hence, this thesis only evaluates the quality of different hand models for RGB images.

Although no works about doing 2D hand pose estimation in real-time using UE4, this thesis studies neural network architectures which do hand pose recognition from single RGB image. Given the state-of-the-art performance made by graph-based approaches using ResNet as a backbone to extract image feature maps (Kourbane and Genc 2022) and complete and efficient feature maps (Guo et al. 2021), this thesis selects ResNet (He et al. 2015) and HRNet (Wang et al. 2019). ResNet and HRNet can be applied from various tasks of computer vision, e.g. image recognition, object detection, object tracking, etc. They can be used to predict 2D coordinates of hand joints. Which task of computer vision can be applied is based on the dataset on which the model is trained. The avaliable datasets for 2D hand training are: COCO-WholeBody, OneHand10K, panoptic2d, RHD. The images and annotations of these datasets are different, which will be described in detail in the following subsections. In some degree, the quality of a model is determined by the datasets, because the image quality and accuracy of annotations varies in the different datasets. Hence, this thesis trained ResNet and HRNet on four common datasets for hand prediction, including COCO-WholeBody, OneHand10K, panoptic2d, and RHD.

4.1 Datasets

4.1.1 COCO-WholeBody

There are not whole-body annotations in existing datasets so people have to train their models independently on each dataset of parts of body, e.g. hand, face, etc. Given this situation, the dataset of COCO with whole-body annotations was extended and called the new one as ”COCO-WholeBody” (Jin et al. 2020). The bounding boxes and keypoints of parts of body in COCO are fully annotated. COCO-WholeBody is a large-scale dataset which aims at training models of human whole-body pose estimation. This dataset not only promotes the development of human whole-body pose estimation, but also promotes the related areas, e.g. hand detection. Moreover, this dataset is validated by cross-dataset evaluation.

The keypoints of hand, face and foot in COCO are annotated to generate the whole-body annotations with the original keypoints of body. Each person is annotated by 4 types of bounding boxes: face, left & right hand and person box. Also, each person is annotated by 133 keypoints: face, hands, body and feet have 68, 42, 17 and 6 keypoints respectively. Only clear enough boxes will be labled as ”valid” and they will be labeled keypoints. The annotation has four steps: manually label boxes, quality control, produce pseudo keypoint labels, manual correction of pseudo labels. The quality of annotation is measured by three annotators.

COCO-WholeBody has about 130 thousand labeled faces, left & right hand boxes, 4 million face keypoints and more than 800 thousand hand keypoints. Figure 4.1 shows an example in the dataset. The degree of blur of images are measured by log 10 of the Laplacian of their converted form. The images of hands in COCO-WholeBody have a variety of hand gestures and they are collected in-the-wild. The 2D hand poses are scaled and rotated first. Then they are clustered into the groups of ”palm”, ”fist” and ”others”.

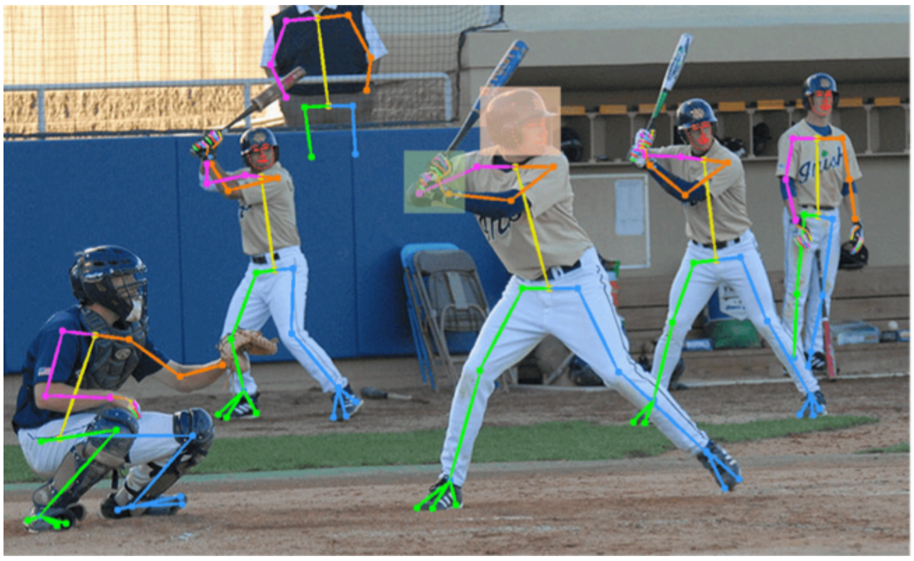


Figure 4.1 *The visualized image in COCO-WholeBody (Jin et al. 2020)*

4.1.2 OneHand10K

An RGB in-the-wild hand dataset, OneHand10K to raise the accuracy of hand pose detection and train their cascaded deep learning mode (Yangang Wang and Y. Liu 2019). This dataset contains silhouette information which is the important cue in the tasks of human hand tracking. Moreover, this dataset does 2D hand pose estimation but it is used in 3D hand pose estimation, given the 2D-to-3D regression algorithms (Zimmermann and Brox 2017). OneHand10K is supposed to give a benchmark to promote the development of hand pose estimation.

OneHand10K has 11703 RGB single hand images. Each image shows only one hand with different kinds of poses. The images are collected from volunteers by doing limited range of hand gestures. Their actions are captured under different environments.

The images in the existing RGB hand pose datasets are not annotated well, but in OneHand10K, each hand images are annotated with 21 joints. Each finger has 4 joints and the palm has 1 joint. For the occluded or invisible keypoints, they are annotated as $(-1, -1)$ so they will generate zero map in the heatmap. To keep the annotation quality, the cross-validation is used by using LabelMe (Russell et al. 2008). First, the dataset is divided into some packages and they are given to different users randomly. Each package has about 100 images. Second, the users selected 5 not fully annotated packages. If the package have been validated by more than three users, it will have been marked as ”completed”. The users will correct annotation results.

The image which have visible hand in OneHand10K is manually given the segmented masks. If the image has wrist pixels, these pixels will be enrolled into the segmentation masks. The Figure 4.2 shows the hand mask, the detected 2D hand pose and the joint heatmaps in OneHand10K.

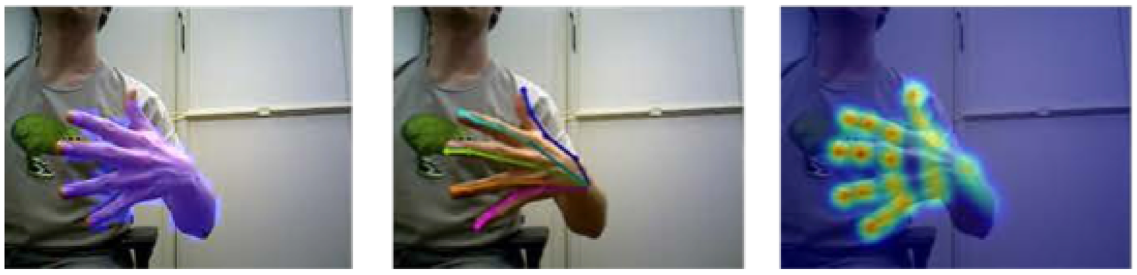


Figure 4.2 *The images of hand mask (left), the detected 2D hand pose (middle) and the joint heatmaps (right) in OneHand10K (Zimmermann and Brox 2017)*

4.1.3 Panoptic Studio dataset with annotated 2D keypoints for hand images (panoptic2d)

A method was proposed to train a detector for keypoints by using multi-camera system (Simon, Joo, I. A. Matthews, et al. 2017). This process is called ”multiview bootstrapping” and it has two steps. The first step is generating initial detector and triangulating keypoints. The second step is improving detector by training reprojected triangulations. The multiview bootstrapping can train hand keypoint detector for single images and the detector can operate the RGB images in real-time. My thesis work focuses the work of using detector to annotate raw dataset.

In multiview bootstrapping, the researchers generated an initial keypoint detector for hands' keypoints from different viewpoints. The training used MPII Human Pose dataset and New Zealand Sign Language (NZSL) Exercises (Andriluka et al. 2014; Deaf Studies Department of Victoria University of Wellington 2010). Then, the initial detector was used on Panoptic Studio dataset and generated noisy labels (Joo et al. 2017). These labels will be triangulated in 3D by using multiview geometry or marked as outliers.

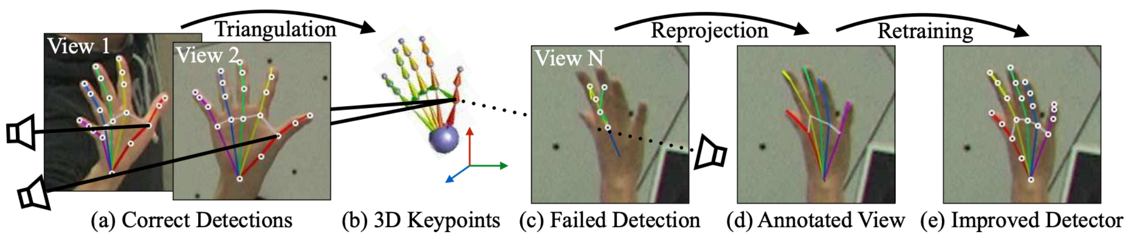


Figure 4.3 *The structure of Multiview Bootstrapping (Simon, Joo, I. A. Matthews, et al. 2017)*

The triangulation is implemented by Equation 4.1 and Equation 4.2. In each iteration i, each frame f, the initial detector $d_i(\mathbf{I}_v^f)$ processes all views V. The Equation 4.1 shows this step, where v denotes each view. This Equation 4.1 outputs a set D of $2D$ location.

$$D \leftarrow \{d_i(\mathbf{I}_v^f) \text{ for } v \in [1 \dots V]\} \tag{4.1}$$

Then, each point p are triangulated robustly into a 3D location. The final triangulated position is obtained by minimizing the reprojection error. The Equation 4.2 shows this process, where I_p^f is RANSAC inlier set. $P_v(\mathbf{X})$ denotes the projection from X to view v.

$$\mathbf{X}_p^f = \underset{\mathbf{X}}{\operatorname{argmax}} \sum_{v \in I_p^f} \|P_v(\mathbf{X}) - \mathbf{X}_p^v\|_2^2 \tag{4.2}$$

4.1.4 Rendered hand pose dataset (RHD)

The existing datasets for hand pose estimation are lack of variation, available samples and insufficient annotation, so they have bad performance on training deep neural network. To solve this problem, Rendered hand pose dataset (RHD) was proposed (Zimmermann and Brox 2017). RHD is a large scale 3D hand pose dataset. It is introduced based on synthetic hand models and it has various data augmentation, ground truth 3D keypoints. To promote to labeling performance of this dataset, the researchers used Mixamo which has human annotators in three-dimensional space. Then they used Blender to render images.

In the each frame of this dataset, camera location is selected randomly. All of the centers of hands are lying toward to camera center in the range of $[40, 65]$cm. To makes hands partially visible from the current perspective, the camera is rotated. When the location and orientation of camera are fixed, the background is chosen randomly and it did not has persons. When each frame is rendered, the researchers used light and global illumination to make the color of the background image was matched. They randomized light positions, light intensities and the skin's specular reflections. Hence, the visual diversity of the dataset is maximized.

Rendered hand pose dataset is made by 20 characters doing 39 actions. It is divided into validation set (R-val) and training set (R-train). R-train has 16 characters doing 31 actions and R-val has 4 characters doing 8 actions. The characters and actions are exclusively exist in either R-train or R-val. All of images' resolution are 320×320. As the Figure 4.4 shows, the skeleton of a hand has 21 keypoints, 33 segmentation masks and 1 background. The keypoints are annotated by 4 keypoints per finger and 1 for palm. The segmentation maps comprise 3 segments per finger, 1 for palm, 1 for person and 1 for background. For each hand keypoint, it is visible or occluded.

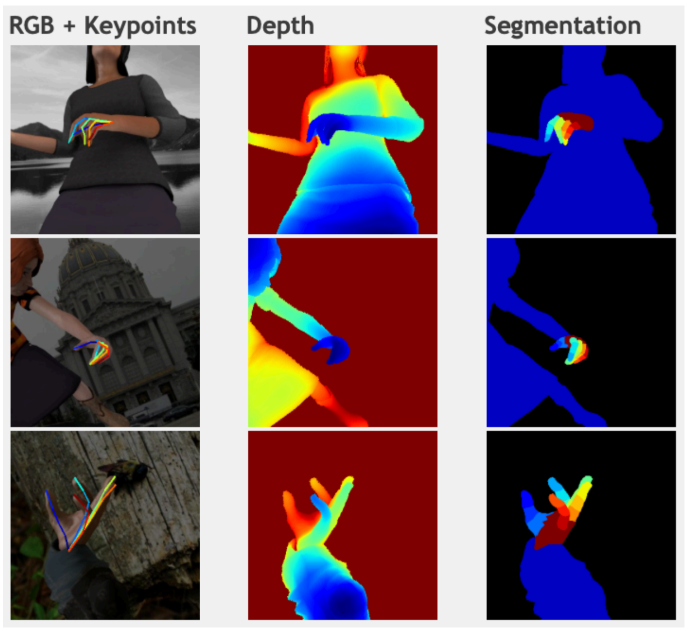


Figure 4.4 *The images in Rendered Handpose Dataset (Zimmermann and Brox 2017)*

4.2 Evaluating hand models for single images

There are four stages in evaluation of hand models: training ResNet and HRNet on four datasets (COCO-WholeBody, OneHand10K, panoptic2d and RHD), testing the value of distance for the trained models, analyzing prediction quality of hand landmarks for own photos and comparing inference time of trained models.

4.2.1 Training ResNet and HRNet on available datasets

The datasets, OneHand10K, panoptic2d and RHD contains hand images, bounding boxes of left hand and right hand as inputs and x, y coordinates of 21 hand joints as labels. The index and name of 21 hand joints are seen in the Figure 4.5. COCO-WholeBody dataset contains x, y coordinates of 134 body joints, including face, pose, left hand, right hand and feet, so training on this dataset only requires extracting the annotations of the hands. Moreover, the hand images in OneHand10K, panoptic2d, and RHD only contain one hand, so the indices of 21 hand joints of left and right hands are same. Given a single hand image may contain two hands in the COCO-WholeBody dataset, so the indices of hand joints from the both hands are different, which are visualized in the Figure 4.6.

The input of ResNet and HRNet is a single processed hand image. The original images from the dataset were cropped using ground-truth bounding boxes provided by the datasets. In this step, if a image contains two hands, it will generate two cropped hand images. Then, the cropped images were resized to 256×256 and the output is 2D coordinates of 21 joints of a hand. Both of ResNet and HRNet were trained 100 epochs with learning rate 0.0005. The training curves of HRNet and ResNet on four different datasets are shown in the Figure 4.7, Figure 4.8, Figure 4.9, Figure 4.10 respectively.

4.2.2 Accuracy of prediction of trained models

This thesis compared ResNet and HRNet for hand joints estimation based on the distance between predicted coordinates of joints and ground truth, which is a simple way to test the accuracy of predictions. According to the section 4.2.1, ResNet and HRNet were trained on four datasets respectively, which led to the eight trained models. Therefore, these eight trained models were also tested on four datasets. The optimal dataset could be checked for training hand joints estimation model. Moreover, the inference using hand models requires the bounding box of hands in an image. In testing, the ground-truth bounding boxes were provided by the corresponding datasets. However, when making inference for unlabeled data, e.g. own images or webcam images in real-time testing, a hand detection model was applied firstly to get bounding boxes.

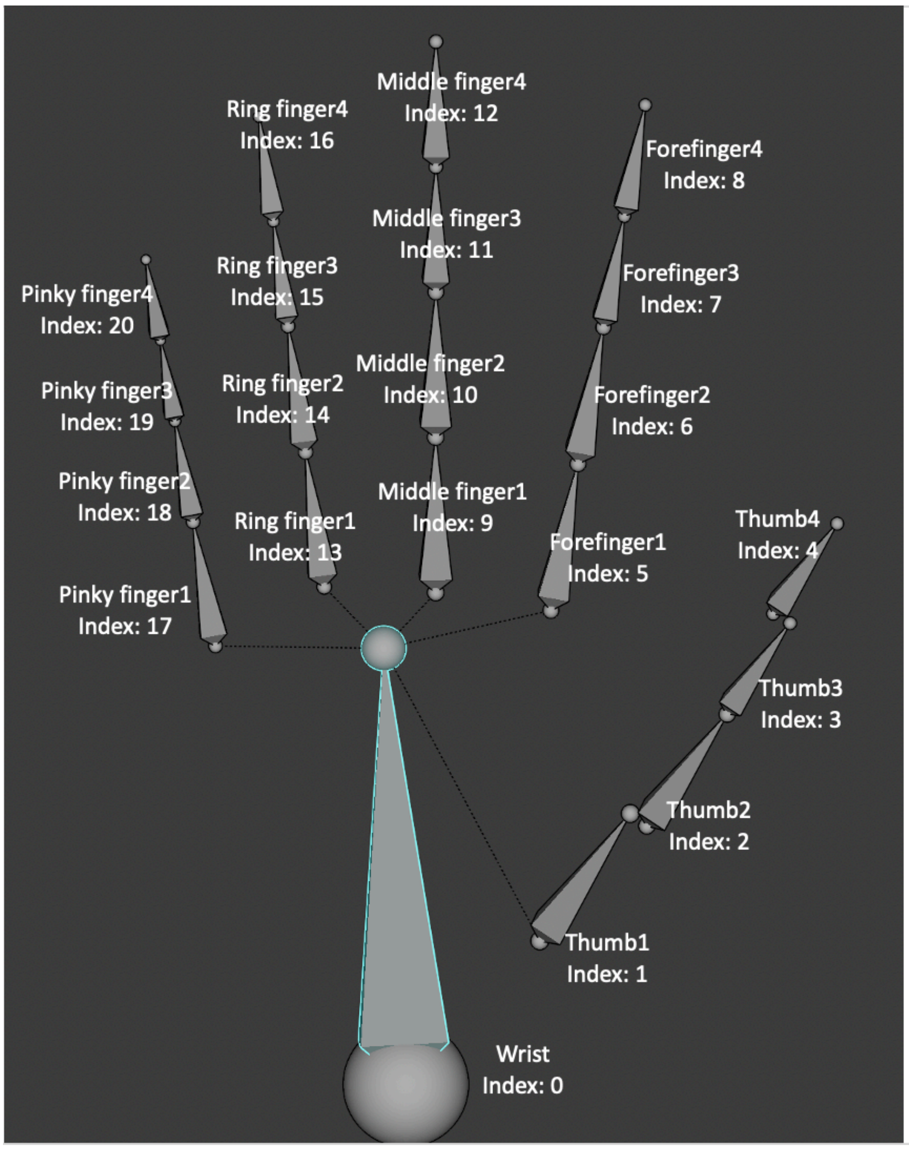


Figure 4.5 *The annotation of hand joints for the datasets, OneHand10K, panoptic2d, and RHD*

The testing results are shown in the Table 4.1. Given the testing result, the preditction error of HRNet is slightly lower than error of ResNet. For HRNet, the trained HRNet on OneHand10K achieved the best performance with reaching 11.21 average distance. The trained HRNet on RHD had 24.82 average distance, which was the worst prediction result. In contrast, RHD had the worst accuracy even the it had the lowest distance tested by itself.

Training dataset affects trained model a lot. For example, training HRNet on OneHand10K produces the lowest average distance error, 11.21, while training HR-Net on RHD produces the very high errors on other datasets, such as OneHand10K, although it performs well on RHD. In general, the average distances of ResNet on

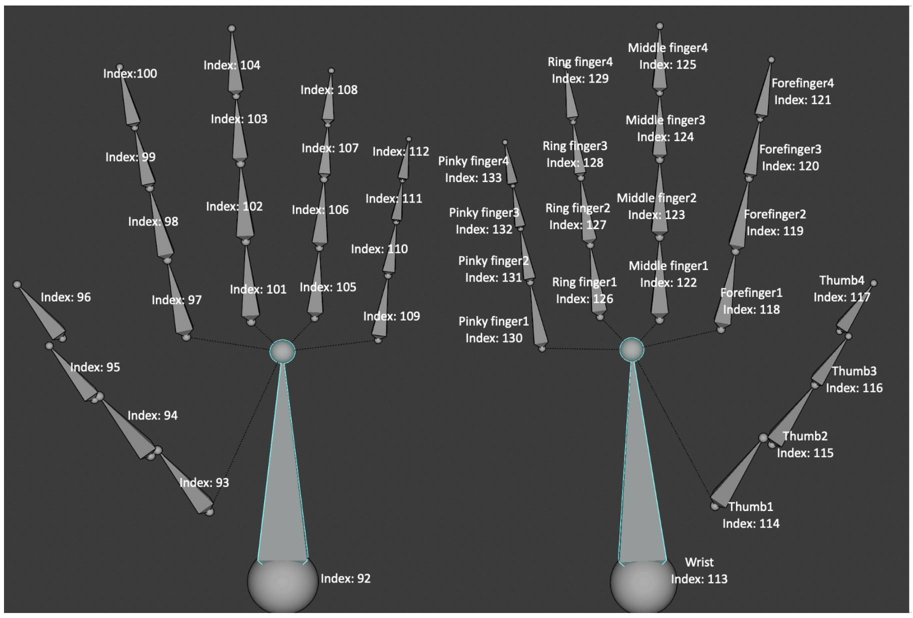
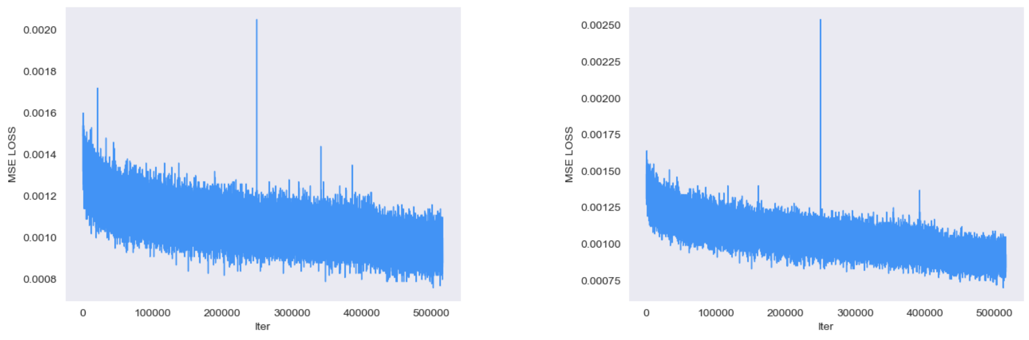


Figure 4.6 *The annotation of hand joints in the COCO-WholeBody*



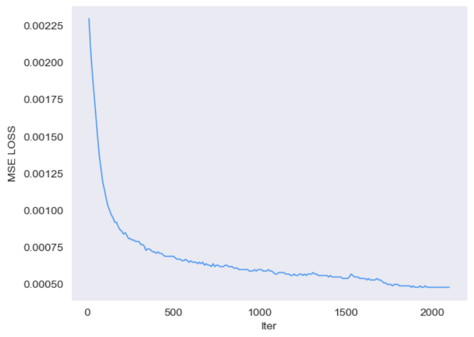
(a) HRNet (b) ResNet

Figure 4.7 *Loss and training iter of HRNet and ResNet on the data from COCO-WholeBody*

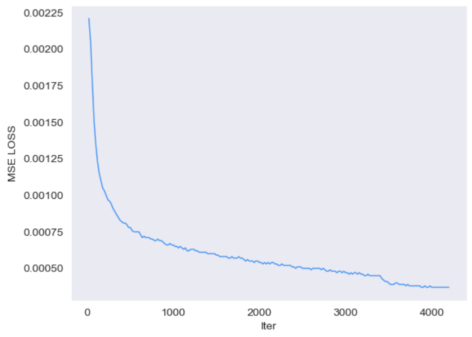
each training dataset were higher than HRNet's average distance.

4.2.3 Inference quality for own photos

The whole inference pipeline has two stages: hand detection for generating bounding boxes and predicting 2D coordinates of 21 hand landmarks using cropped image based on the predicted bounding box in the first stage. The pipeline is shown in the Algorithm 1. The hand detection model used in hand detection is Cascade R-CNN (Z. Cai and Vasconcelos 2017) and it was provided (Contributors 2020). Cascade R-CNN model was trained on OneHand10K for 20 epochs with learning rate 0.001

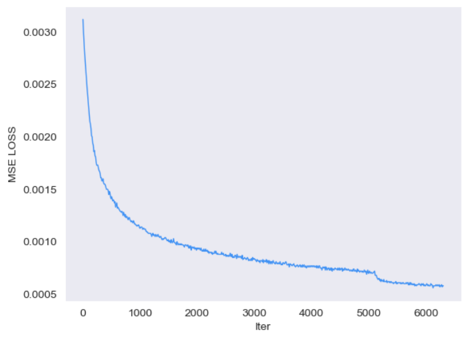


(a) HRNet

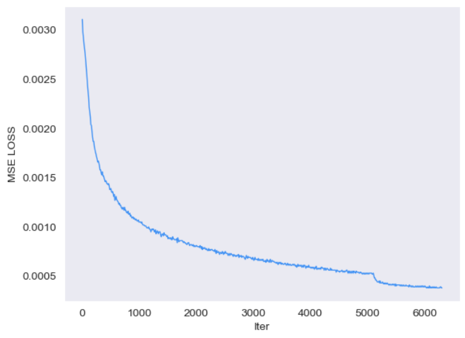


(b) ResNet

Figure 4.8 *Loss and training iter of HRNet and ResNet on the data from OneHand10K*

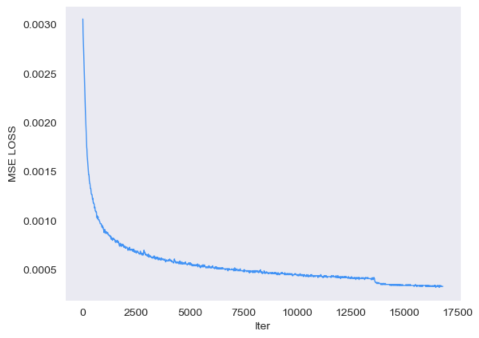


(a) HRNet

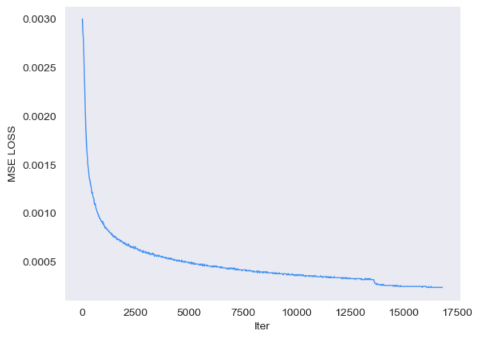


(b) ResNet

Figure 4.9 *Loss and training iter of HRNet and ResNet on the data from panoptic2d*



(a) HRNet



(b) ResNet

Figure 4.10 *Loss and training iter of HRNet and ResNet on the data from RHD*

Table 4.1 *The distance for testing hand joints estimation models on different datasets*

model	training dataset	testing dataset	distance	average distance
HRNet	COCO-WholeBody	COCO-WholeBody	4.0893	12.10
		OneHand10K	12.9400	
		panoptic2d	4.3049	
		RHD	27.0655	
	OneHand10K	COCO-WholeBody	6.8687	11.21
		OneHand10K	8.5522	
		panoptic2d	3.1187	
		RHD	26.2992	
	panoptic2d	COCO-WholeBody	5.1576	19.06
		OneHand10K	31.3975	
		panoptic2d	10.4198	
		RHD	29.2483	
	RHD	COCO-WholeBody	8.9891	24.82
		OneHand10K	71.5026	
		panoptic2d	17.6406	
		RHD	1.1405	
ResNet	COCO-WholeBody	COCO-WholeBody	4.1126	12.91
		OneHand10K	13.4151	
		panoptic2d	6.4795	
		RHD	27.6169	
	OneHand10K	COCO-WholeBody	8.7929	12.61
		OneHand10K	8.0327	
		panoptic2d	4.3049	
		RHD	29.3124	
	panoptic2d	COCO-WholeBody	5.4718	21.21
		OneHand10K	48.3435	
		panoptic2d	3.9036	
		RHD	27.1138	
	RHD	COCO-WholeBody	12.6383	25.13
		OneHand10K	69.3415	
		panoptic2d	17.4732	
		RHD	1.0690	

and input shape is 256×256.

This thesis tested the hand image with "OKAY" pose. In this image, most of joints in the original image are clear and there are less occlusion and bending. For example, middle finger, ring finger and pinky finger are straight, but only thumb and forefinger are bending. Joints, such as *forefinger4* and *thumb1* are occluded. The Figure 4.11, 4.12, 4.13, 4.14 show the prediction results by using ResNet and HRNet model trained on the COCO-WholeBody, Onehand10K, panoptic2d, RHD respectively.

According to these figures, for the models trained on COCO-WholeBody, the occlusion joint of forefinger are not predicted accurately for the both HRNet and

Algorithm 1: A pipeline for estimating coordinates of hand joints from a singe image

```
Data: A single RGB image
Result: 2D coordinates of hand landmarks
bounding boxes of hands ← Cascade R-CNN model as hand detection ;
while bbx in deteced bounding boxes do
    cropped hand image ← cropping original image in the region by bbx
      hand landmarks ← Hand joints estimation model
end
```

ResNet model. However, HRNet is better than the ResNet, by which the 4 joints of forefinger are not predicted well. For the models trained on the dataset of One-hand10K, HRNet and ResNet have good prediction on the straight fingers and occluding or bending joints. If HRNet and ResNet are compared , the prediction for bending joints, like *thumb1* of ResNet is better than HRNet. For the models trained on the dataset of panoptic2d, HRNet produce a very good prediction but ResNet doesn't predict well for forefinger of left hand and middle finger of right hand. For the models trained on RHD dataset, HRNet is better than the ResNet, but the prediction of bending and occluding joints of the both model are not good.

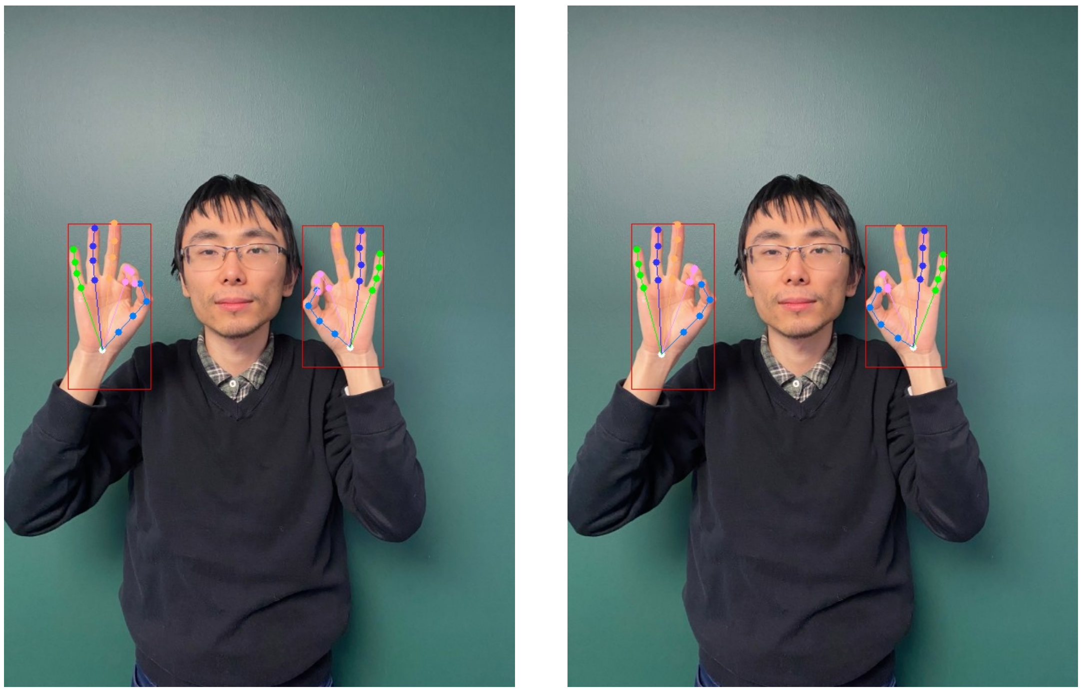
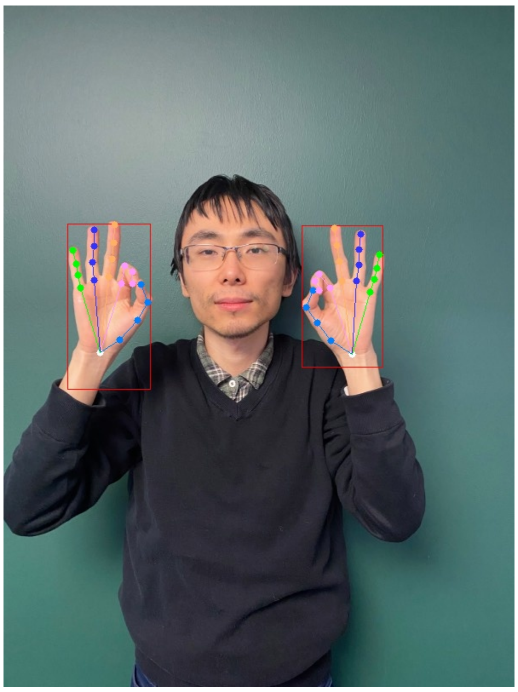

(a) HRNet (b) ResNet

Figure 4.11 *Visualization of estimation of hand joints ("OK" pose) using models trained on COCO-WholeBody*


(a) HRNet

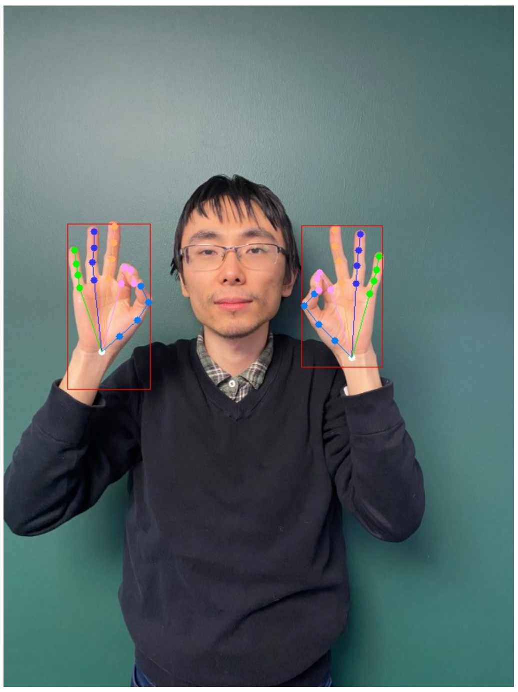
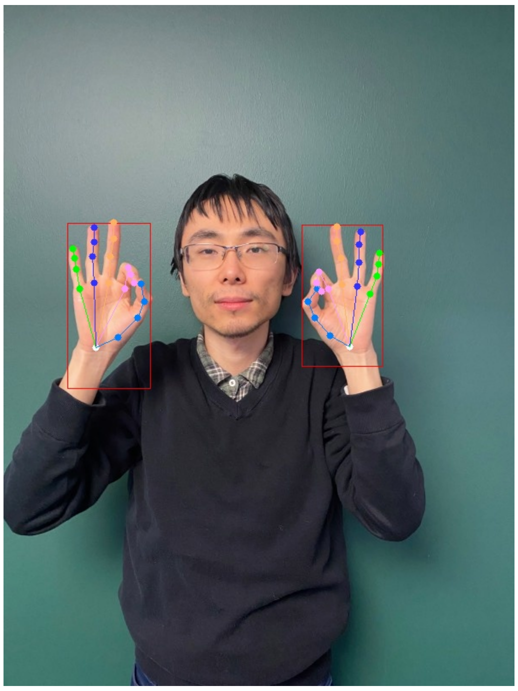

(b) ResNet

Figure 4.12 *Visualization of estimation of hand joints ("OK" pose) using models trained on Onehand10K dataset*


(a) HRNet

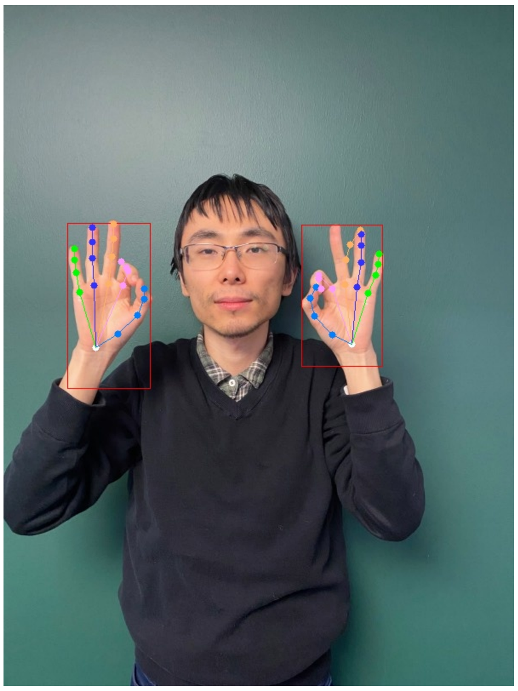
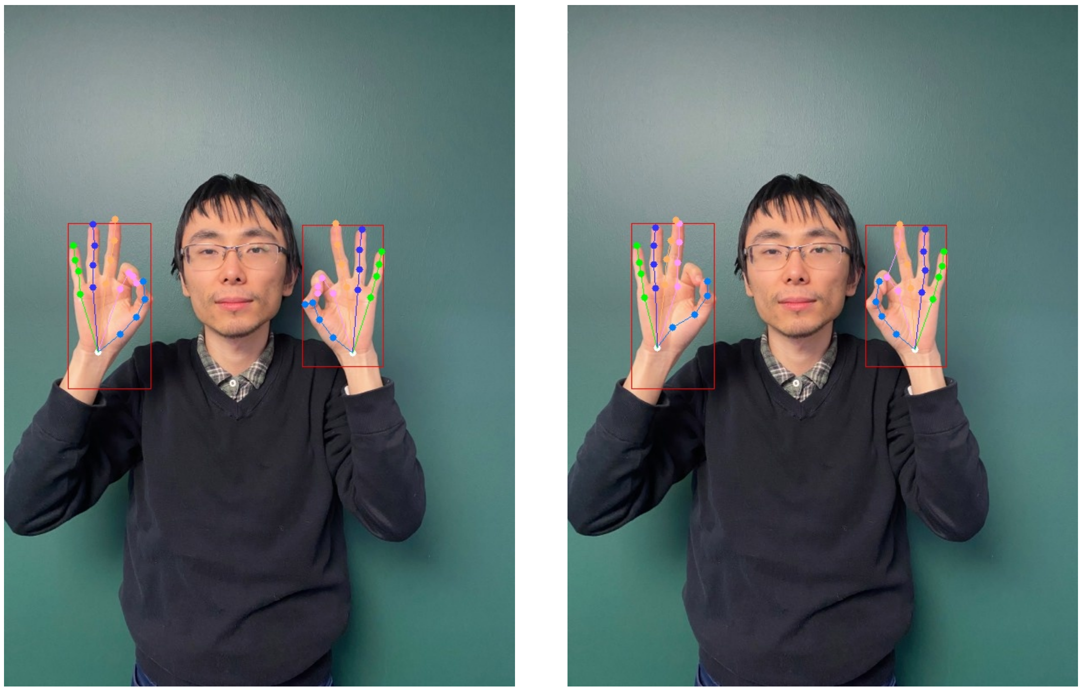

(b) ResNet

Figure 4.13 *Visualization of estimation of hand joints ("OK" pose) using models trained on panoptic2d dataset*



(a) HRNet (b) ResNet

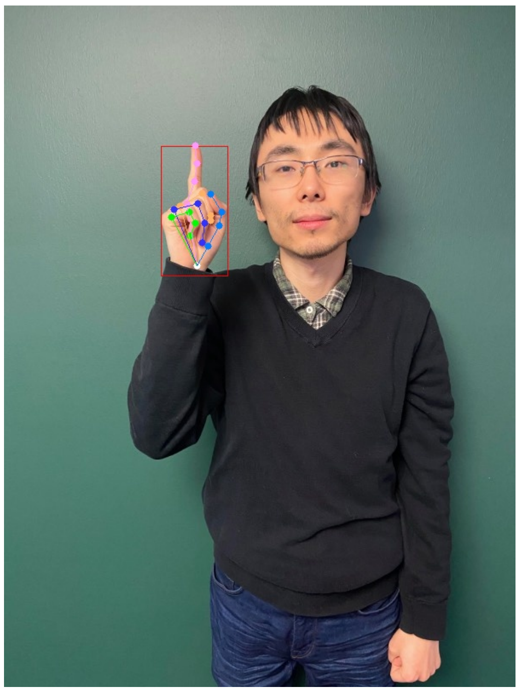
Figure 4.14 *Visualization of estimation of hand joints ("OK" pose) using models trained on RHD dataset*

Moreover, the prediction results of hand images with "one" pose generated by HRNet and ResNet trained on different datasets. They are visualized in the Figure 4.15, 4.16, 4.17, 4.18 respectively. Compared to the above poses, this image contains more bending and occluding joints. In this case, the predictions by models trained on the dataset of Onehand10K is obviously better than the models trained on other datasets. Therefore, training hand joints estimation model on Onehand10K can produce the better prediction and the performance of HRNet trained on different datasets is better than that of ResNet.

Therefore, HRNet and ResNet trained by those four dataset have the ability to predict 21 hand landmarks. For the simple gesture like "OK", HRNet outperformed ResNet trained on those datasets, but ResNet performed little better than HRNet on the joint *thumb1*. For the complex gesture like "1", the trained HRNet on OneHand10K outperformed ResNet on the same dataset.

4.2.4 Inference speed of trained models

In this section, the inference time of HRNet and ResNet is shown in the Table 4.2, by testing on the all of test dataset in COCO-WholeBody, Onehand10K, panoptic2d and RHD. From the Table 4.2, the inference speeds of HRNet and ResNet are quite


(a) HRNet

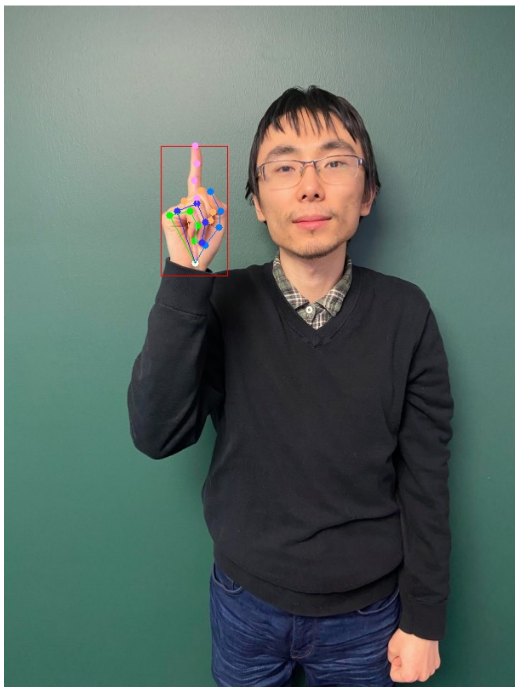
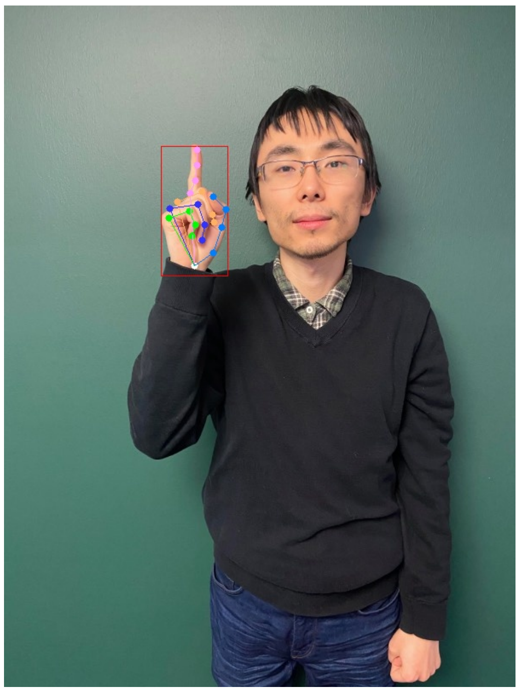

(b) ResNet

Figure 4.15 *Visualization of estimation of hand joints ("one" pose) using models trained on COCO-WholeBody*


(a) HRNet

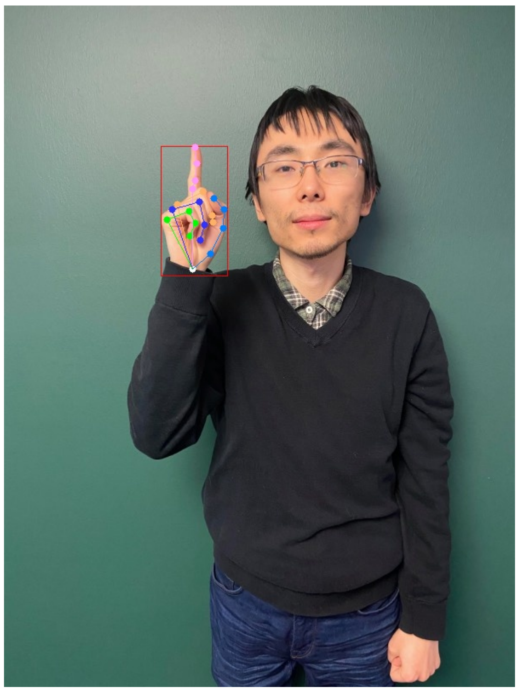

(b) ResNet

Figure 4.16 *Visualization of estimation of hand joints ("one" pose) using models trained on Onehand10K dataset*

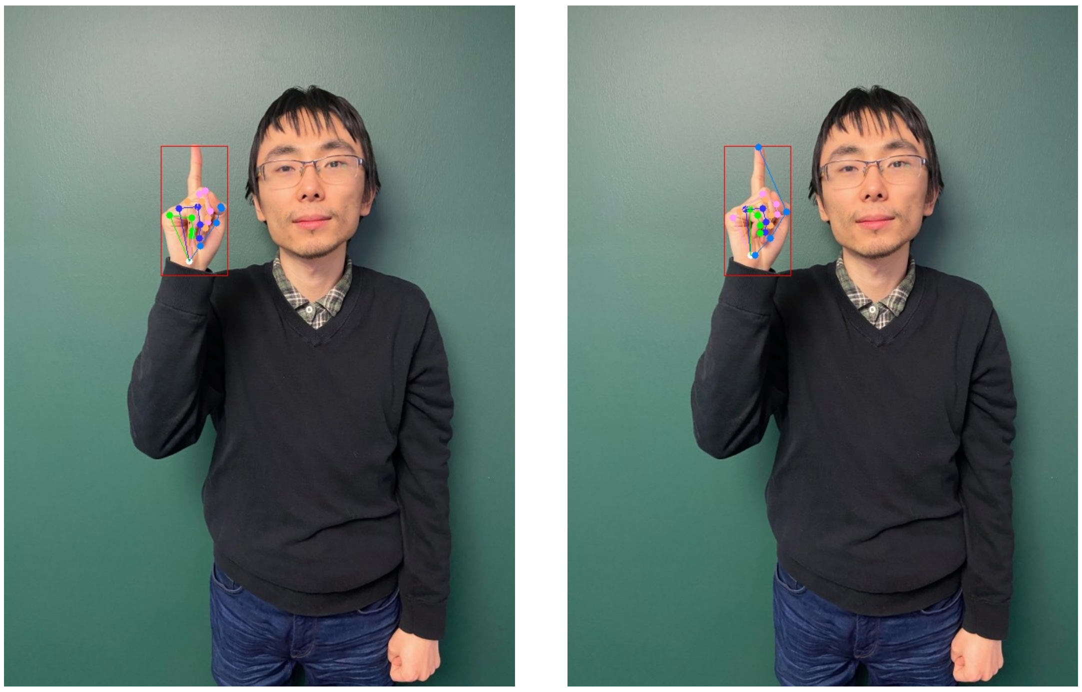

(a) HRNet (b) ResNet

Figure 4.17 *Visualization of estimation of hand joints ("one" pose) using models trained on panoptic2d dataset*

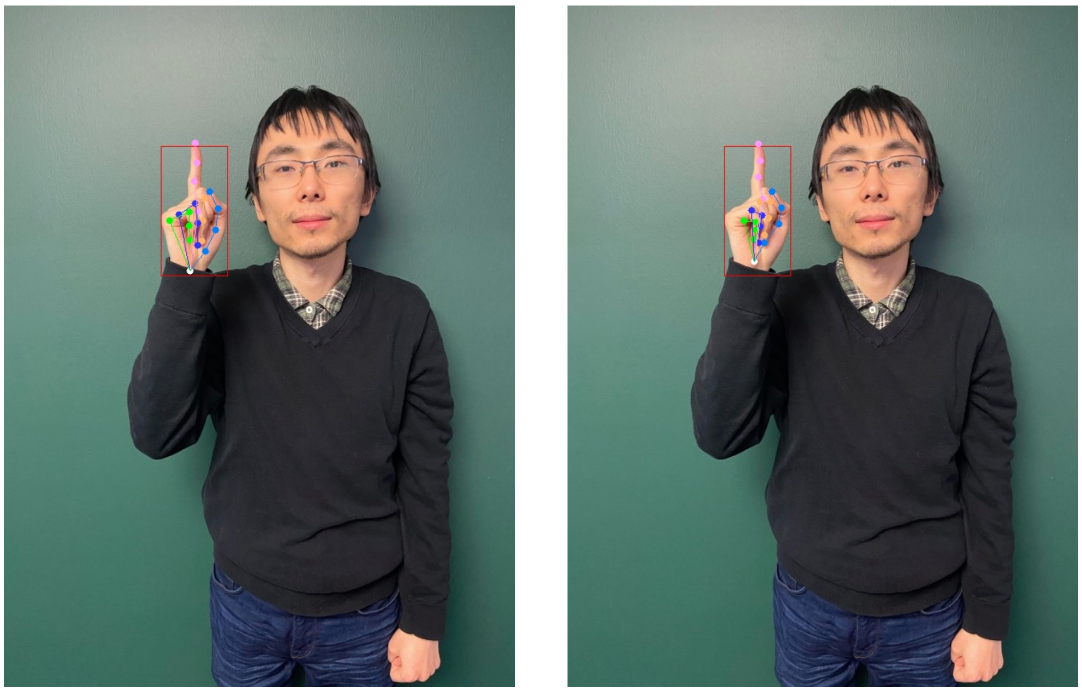

(a) HRNet (b) ResNet

Figure 4.18 *Visualization of estimation of hand joints ("one" pose) using models trained on RHD dataset*

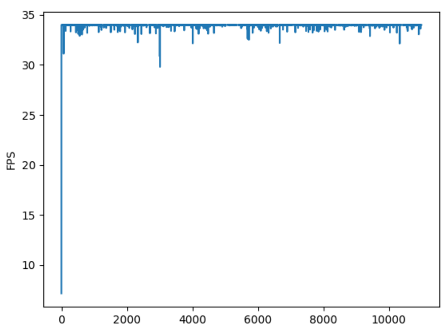
similar.

***Table 4.2** The inference time (per frame) of HRNet and ResNet*

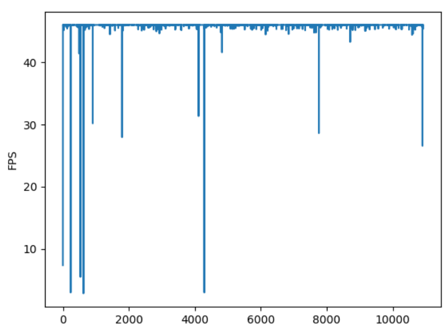
model	**time (1 s / frame)**
HRNet	1.20
ResNet	0.90

4.3 Evaluating hand models in real-time

This thesis built a simple application in Unreal Engine 4 (UE4) to test if the inferences from hand models run smoothly in real-time. The input image is the frame from webcam and each frame is processed by hand models to generate estimated hand joints. Then the Frames Per Second (FPS) is used to evaluate inference fluency of these models. The FPS is around 33 when using HRNet and 45 when using ResNet. FPS can be used to determine if they satisfy different applications' FPS requirements. Moreover, some new applications of hand models were proposed. They are using estimated hand joints to move object in the world and predicting digital number.



(a) FPS of the application using HRNet



(b) FPS of the application using ResNet

***Figure 4.19** FPS of two models*

The FPS is calculated by dividing the difference of last frame time (seconds) and the current frame time (seconds) into 1. For example, Figure 4.19(a) shows FPS of the application using HRNet. The x-axis shows about 10000 indices and the y-axis shows the FPS corresponding to each index. The most of FPS values are stable at 33, except for several lower values. Figure 4.19(b) shows FPS of the application using ResNet. The x-axis shows about 10000 indices and the y-axis shows the FPS corresponding to each index. The most of FPS values are stable at 45, except for several lower values. For example, the FPS of the first frame is 7.38 because loading models and initialization take much time.

The first application in real-time is ”moving objects in the world using estimated hand joints”. This functionality can be realized by using estimated 2D coordinates of the joint *forefinger4*. The estimated coordinates from hand models are normalized. Then normalized coordinates are multiplied with screen resolution, which would get the real 2D location on the user’s screen. The screenshot of moving a cube in UE4 can be seen in the Figure 4.20.

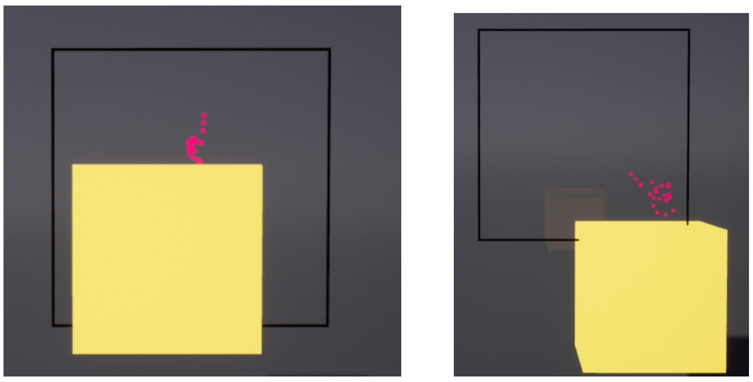
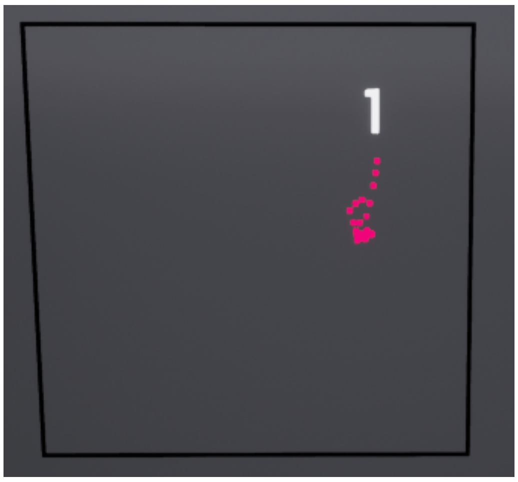
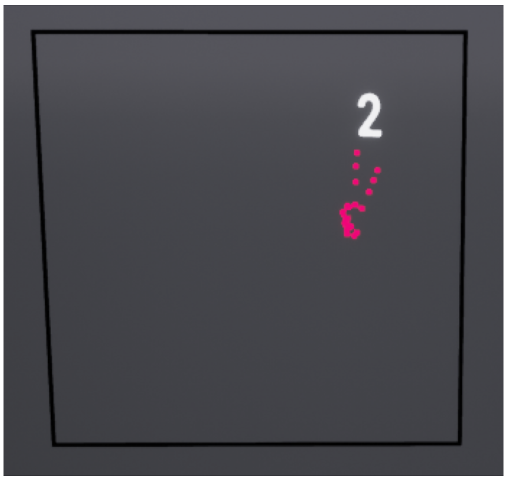

(a) Moving cube to left side (b) Moving cube to right side

Figure 4.20 *Visualization of moving a cude in UE4*

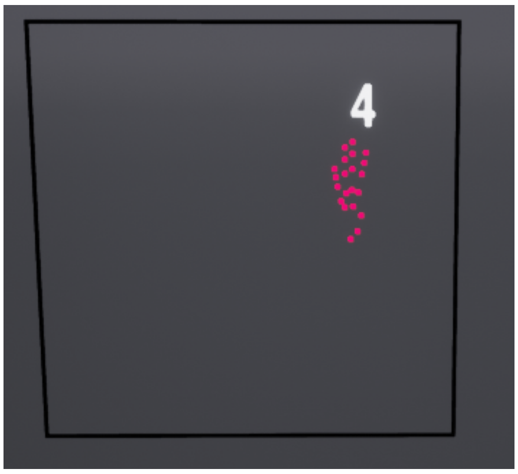
The second application in real-time is ”predicting digital number or pose from hand images”. Some simple poses can be predicted from the estimated hand joints. For example, the pose ”one” can be predicted by checking if 2D coordinates of 4 joints of forefinger are in a straight line. The pose ”2”, ”3”, ”5” can be predicted by testing if coordinates of joints of middle finger, ring finger and pinky finger are in the straight line. Moreover, the two digits and other simple pose, like ”thumb up” can be predicted by testing relations of coordinates of hand joints, like if they are in a straight line. The result of predicting of simple pose can be seen in the Figure 4.21.



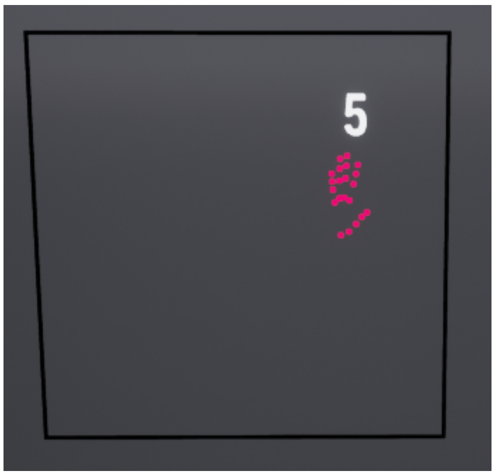
(a) Pose ”1”



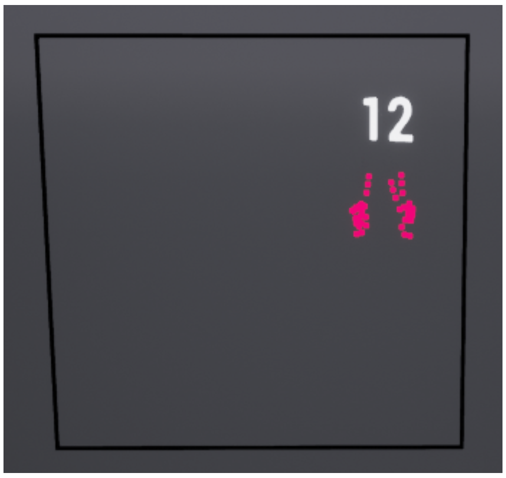
(b) Pose ”2”



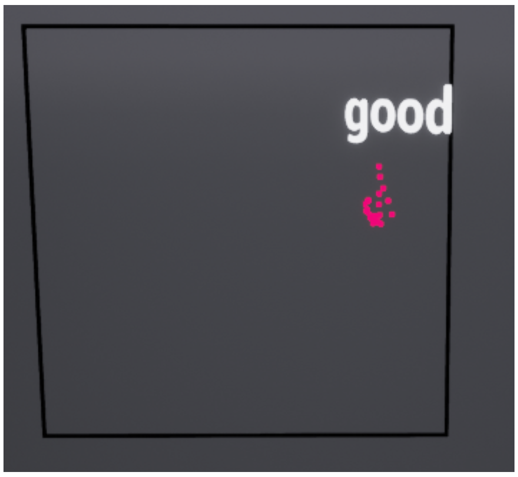
(c) Pose ”4”



(d) Pose ”5”



(e) Pose ”12”



(f) Pose ”Thumb Up”

Figure 4.21 *Visualization of predictions of poses in UE4*

5 CONCLUSION

In this thesis, two neural network architectures (HRNet and ResNet) were trained on four datasets (COCO-WholeBody, OneHand10K, panoptic2d and RHD) and generated eight models for hand pose estimation. Each model was evaluated by calculating distance and average distance on each datasets. Thus 32 values of distance and 8 values of average distance were obtained. Moreover, they were tested on self-taken photos and Unreal Engine 4.

In real-time using UE4, which hand pose estimation models should be chosen depends on the importance of prediction accuracy or prediction time. This thesis proves that HRNet trained on the selected four datasets had better performance than ResNet. The dataset OneHand10K is more suitable for HRNet and ResNet compared to those four training datasets. Therefore, if applications have high requirements on prediction accuracy, HRNet is better. However, the FPS of ResNet is little higher than HRNet, so ResNet is better if the prediction time is more important. If applications do not have strict requirements on prediction accuracy and time, HRNet has broader applicability than ResNet in general. For example, if one of two models is applied to boxing games, different hand pose prediction accuracy affects player's game experience much more than similar FPS.

The future work of this thesis includes two parts: increasing the hand poses variety and decreasing the prediction time of HRNet or ResNet. For hand poses variety, this thesis work can predict a few simple digital numbers and two simple hand gestures. However, some hand gestures of the numbers from ”six” to ”nine” are hard to be recognized by calculating hand keypoints' coordinates. One possible way is building another model to input those coordinates and output the name of hand gestures. For decreasing the prediction time of HRNet or ResNet, applications can react to users' operations quicker than before so the user experience will be improved.

References

Andriluka, Mykhaylo et al. (2014). “2D Human Pose Estimation: New Benchmark and State of the Art Analysis”. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3686–3693. DOI: 10.1109/CVPR.2014.471.

Bishop, C.M. (1995). *Neural networks for pattern recognition.* Oxford University Press, USA.

Bridle, John S. (1989). “Training Stochastic Model Recognition Algorithms as Networks Can Lead to Maximum Mutual Information Estimation of Parameters”. In: *Proceedings of the 2nd International Conference on Neural Information Processing Systems*. NIPS’89. Cambridge, MA, USA: MIT Press, pp. 211–217.

Cai, Zhaowei and Nuno Vasconcelos (2017). *Cascade R-CNN: Delving into High Quality Object Detection*. DOI: 10.48550/ARXIV.1712.00726. URL: https://arxiv.org/abs/1712.00726.

Chen, Xinghao et al. (June 2020). “Pose guided structured region ensemble network for cascaded hand pose estimation”. In: *Neurocomputing* 395, pp. 138–149. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2018.06.097. URL: http://dx.doi.org/10.1016/j.neucom.2018.06.097.

Contributors, MMPose (2020). *OpenMMLab Pose Estimation Toolbox and Benchmark*. https://github.com/open-mmlab/mmpose.

Deaf Studies Department of Victoria University of Wellington (2010). *NZ sign language exercises*. https://www.wgtn.ac.nz/llc/llc_resources/nzsl/, Last accessed on 2022-04-16.

Fourure, Damien et al. (2017). “Multi-task, multi-domain learning: Application to semantic segmentation and pose regression”. In: *Neurocomputing* 251, pp. 68–80. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2017.04.014. URL: https://www.sciencedirect.com/science/article/pii/S0925231217306847.

Ge, Liuhao, Yujun Cai, et al. (2018). “Hand PointNet: 3D Hand Pose Estimation using Point Sets”. In: *CVPR*.

Ge, Liuhao, Hui Liang, et al. (2018). “Robust 3D Hand Pose Estimation From Single Depth Images Using Multi-View CNNs”. In: *IEEE Transactions on Image Processing* 27.9, pp. 4422–4436. DOI: 10.1109/TIP.2018.2834824.

Ge, Liuhao, Zhou Ren, et al. (2019). “3D Hand Shape and Pose Estimation From a Single RGB Image”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10825–10834. DOI: 10.1109/CVPR.2019.01109.

Guo, Shaoxiang et al. (2021). “Graph-Based CNNs With Self-Supervised Module for 3D Hand Pose Estimation From Monocular RGB”. In: *IEEE Transactions on*

Circuits and Systems for Video Technology 31.4, pp. 1514–1525. DOI: 10.1109/TCSVT.2020.3004453.

He, Kaiming et al. (2015). “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385. arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

Ioffe, Sergey and Christian Szegedy (2015). *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. arXiv: 1502.03167 [cs.LG].

Jin, Sheng et al. (2020). “Whole-Body Human Pose Estimation in the Wild”. In: *CoRR* abs/2007.11858. arXiv: 2007.11858. URL: https://arxiv.org/abs/2007.11858.

Joo, Hanbyul et al. (2017). “Panoptic Studio: A Massively Multiview System for Social Interaction Capture”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Kanal, Laveen N. (2003). “Perceptron”. In: *Encyclopedia of Computer Science*. GBR: John Wiley and Sons Ltd., pp. 1383–1385. ISBN: 0470864125.

Kourbane, Ikram and Yakup Genc (Mar. 2022). “A graph-based approach for absolute 3D hand pose estimation using a single RGB image”. In: *Applied Intelligence*. DOI: 10.1007/s10489-022-03390-x.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Kyurkchiev, Nikolay (Dec. 2016). “A family of recurrence generated sigmoidal functions based on the Verhulst logistic function. Some approximation and modelling aspects”. In: *Biomath Communications* 3. DOI: 10.11145/bmc.2016.12.171.

Lecun, Y. et al. (1998). “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. DOI: 10.1109/5.726791.

Lemaréchal, Claude (2010). *Cauchy and the Gradient Method*. https://www.math.uni-bielefeld.de/documenta/vol-ismp/40_lemarechal-claude.pdf, Last accessed on 2022-06-23.

Lin, John, Ying Wu, and T.S. Huang (2000). “Modeling the constraints of human hand motion”. In: *Proceedings Workshop on Human Motion*, pp. 121–126. DOI: 10.1109/HUMO.2000.897381.

Liu, Wei et al. (2016). “SSD: Single Shot MultiBox Detector”. In: *Lecture Notes in Computer Science*, pp. 21–37. ISSN: 1611-3349. DOI: 10.1007/978-3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

Mcculloch, Warren and Walter Pitts (1943). “A Logical Calculus of Ideas Immanent in Nervous Activity”. In: *Bulletin of Mathematical Biophysics* 5, pp. 127–147.

Moon, Gyeongsik, Ju Yong Chang, and Kyoung Mu Lee (2018). *V2V-PoseNet: Voxel-to-Voxel Prediction Network for Accurate 3D Hand and Human Pose Estimation from a Single Depth Map.* arXiv: 1711.07399 [cs.CV].

Newell, Alejandro, Kaiyu Yang, and Jia Deng (2016). *Stacked Hourglass Networks for Human Pose Estimation.* arXiv: 1603.06937 [cs.CV].

Oberweger, Markus and Vincent Lepetit (2017). *DeepPrior++: Improving Fast and Accurate 3D Hand Pose Estimation.* arXiv: 1708.08325 [cs.CV].

Qi, Charles Ruizhongtai et al. (2016). “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation”. In: *CoRR* abs/1612.00593. arXiv: 1612.00593. URL: http://arxiv.org/abs/1612.00593.

Robbins, Herbert E. (2007). “A Stochastic Approximation Method”. In: *Annals of Mathematical Statistics* 22, pp. 400–407.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1988). “Learning Representations by Back-Propagating Errors”. In: *Neurocomputing: Foundations of Research.* Cambridge, MA, USA: MIT Press, pp. 696–699. ISBN: 0262010976.

Russell, Bryan C. et al. (2008). “LabelMe: A Database and Web-Based Tool for Image Annotation.” In: *Int. J. Comput. Vis.* 77.1-3, pp. 157–173. URL: http://dblp.uni-trier.de/db/journals/ijcv/ijcv77.html#RussellTMF08.

Simon, Tomas, Hanbyul Joo, Iain Matthews, et al. (July 2017). “Hand Keypoint Detection in Single Images Using Multiview Bootstrapping”. In: pp. 4645–4653. DOI: 10.1109/CVPR.2017.494.

Simon, Tomas, Hanbyul Joo, Iain A. Matthews, et al. (2017). “Hand Keypoint Detection in Single Images using Multiview Bootstrapping”. In: *CoRR* abs/1704.07809. arXiv: 1704.07809. URL: http://arxiv.org/abs/1704.07809.

Simonyan, Karen and Andrew Zisserman (Sept. 2014). “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: *arXiv 1409.1556.*

Sun, Xiao et al. (June 2015). “Cascaded Hand Pose Regression”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Unreal Engine 4.27 documentation (2021). *Skeleton Assets.* https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/Skeleton/, Last accessed on 2022-03-22.

Wan, Chengde et al. (2017). “Crossing Nets: Dual Generative Models with a Shared Latent Space for Hand Pose Estimation”. In: *CoRR* abs/1702.03431. arXiv: 1702.03431. URL: http://arxiv.org/abs/1702.03431.

Wang, Jingdong et al. (2019). “Deep High-Resolution Representation Learning for Visual Recognition”. In: *CoRR* abs/1908.07919. arXiv: 1908.07919. URL: http://arxiv.org/abs/1908.07919.

Xu, Bing et al. (2015). “Empirical Evaluation of Rectified Activations in Convolutional Network”. In: *CoRR* abs/1505.00853. arXiv: 1505.00853. URL: http://arxiv.org/abs/1505.00853.

Yangang Wang, Cong Peng and Yebin Liu (Nov. 2019). “Mask-pose Cascaded CNN for 2D Hand Pose Estimation from Single Color Images”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 29.11, pp. 3258–3268. DOI: 10.1109/TCSVT.2018.2879980. URL: http://yangangwang.com/papers/WANG-MCC-2018-10.html.

Ye, Qi, Shanxin Yuan, and Tae-Kyun Kim (2016). “Spatial Attention Deep Net with Partial PSO for Hierarchical Hybrid Hand Pose Estimation”. In: *CoRR* abs/1604.03334. arXiv: 1604.03334. URL: http://arxiv.org/abs/1604.03334.

Yuan, Shanxin et al. (2017). “The 2017 Hands in the Million Challenge on 3D Hand Pose Estimation”. In: *CoRR* abs/1707.02237. arXiv: 1707.02237. URL: http://arxiv.org/abs/1707.02237.

Zeiler, Matthew D. and Rob Fergus (2013). “Visualizing and Understanding Convolutional Networks”. In: *CoRR* abs/1311.2901. arXiv: 1311.2901. URL: http://arxiv.org/abs/1311.2901.

Zhang, Fan et al. (2020). “MediaPipe Hands: On-device Real-time Hand Tracking”. In: *CoRR* abs/2006.10214. arXiv: 2006.10214. URL: https://arxiv.org/abs/2006.10214.

Zimmermann, Christian and Thomas Brox (2017). “Learning to Estimate 3D Hand Pose from Single RGB Images”. In: *CoRR* abs/1705.01389. arXiv: 1705.01389. URL: http://arxiv.org/abs/1705.01389.