Hand Gesture Recognition based on LSTM Using Leap Motion

1st Hanqi SU
Department of Mechanical and Energy Engineering
Southern University of Science and Technology
Shenzhen, China
11912625@mail.sustech.edu.cn

***Abstract*—This paper introduces the background, development ,basic principle and application of dynamic hand gesture and RNN, LSTM, CNN neural networks, we uses LSTM neural network to construct U-LSTM, BI-LSTM, HBU-LSTM structures to achieve the classification task of dynamic gesture.**

***Index Terms*—Dynamic Hand Gesture, RNN, LSTM, HBU-LSTM, CNN**

I. Introduction

Over the years, there has been a lot of research into hand gesture recognition, because hand gesture recognition is a frequently used forms of communication humans use in addition to speaking and writing [1]. A hand gesture is a motion of the body that contains information. It has the characteristics that using to express a simple command since we can not contain a lot of massages in a single gestures. However, that is not a problem in hand gesture recognising. What most confusing is that the appearance of hand gestures has natural randomness [2]. That means people in different area will express different command with the same gesture. To solve this problem it exposes another characteristic that gestures which are used by a large amount of people are developed under some sort of agreement [3]. For example, when people use VR devices they all know that wiping means to turn pages or move the view to a certain direction. This is a significant problem for future industrialization of hand gesture recognition, as it relates to the ease of use, decrease misunderstand and dissemination of goods. Although several problems derives from the characteristics of hand gestures, if we solve them it will make a huge difference to nowadays world. The reason is 1.it can provide a new way for human-computer interaction; 2.it can improve communication efficiency as we can't always use voice or letter in some special circumstance; 3.it is a base of other technologies like VR, MR and other artificial intelligence technology [4]. With all these problems and significance of hand gestures research, obviously, there is still a long way to go before gesture recognition is widely used around the world.

Hand gesture recognition research started at the end of the 20th century, after nearly 20 years of development, it has been divided into about three research directions [5]. Specifically, they are detecting which includes detect color, shape, pixel values, 3D model and motion , tracking which includes methods for feature model, MeanShift, CamShift, particle filtering and optimal estimation and recognition which includes using algorithm like template based, support vector machine, hidden Markov model, neural network, dynamic time warping and finite state machine. In this work, we are focusing on neural network and have done deep research of LSTM in different types of neural network.

II. Background

A. Data Collecting

For our work, we need to start at data collecting. Progress in information technologies nowadays has brought us several commercial sensors [7], like the Leap Motion Controller, the Intel Realsense camera, Microsoft Kinect, the Myo armband, the gyroscope and the accelerometer, which are more and more utilized in the domain of computer vision . Sensors commonly used are shown in Fig. 1.

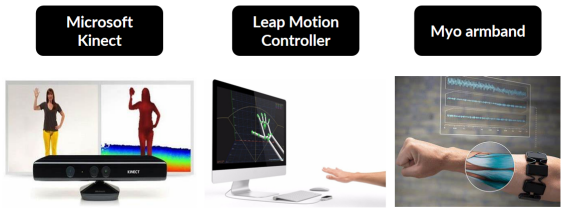


Fig. 1: Commonly used sensors

Among all of them, Leap Motion Controller is the most natural, accurate and low-cost device dedicated to capture hand motions in an intuitive way. This sensor has also the capacity to collect hand movements simultaneously, with precision higher than 0.01 mm. At the same time, Leap Motion Controller can provide researchers large amount of information in different types not only the position of hands or the outline of hands which is beneficial for hand gesture classifying. As a result, Leap Motion Controller is chosen for our research.

Leap Motion uses optical gesture tracking technology and a binocular camera to extract 3d position of hands through binocular stereo vision imaging principle. It simulates the bone joints of real hands and automatically tracks the hand

model generated by each hand and outputs a series of data frames refreshed in real time. A hand has 29 bones, 29 joints, 123 ligaments, 48 nerves and 30 arteries. The Leap Motion controller tracks identifiable information as the hand moves at more than 200 frames per second. When a hand enters Leap Motion's recognition zone, it automatically tracks it, sending out a series of data frames that refresh in real time. Frame data is the most important part of Leap Motion data. Each Frame contains all the information about hand movements, for example, hands, fingers, pointables, tools, gestures and their positions, velocity, direction, rotation, etc. The detailed data is shown in Fig. 2.

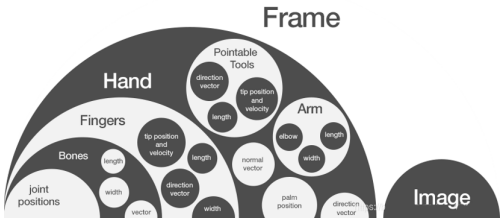


Fig. 2: Data provided by Leap Motion Controller

The location information is three-dimensional information, including plane position and depth position, which is conducive to hand gesture recognition [6]. Leap Motion Controller has the information of each finger in a hand. And for each finger, it is specifically expressed with the distal phalanges, intermediate bones, proximal bones, and metacarpal. Leap Motion simulates the bone joints of real hands and represents the skeleton of hands in the form of a stick model, in which the joints are represented by a round ball and the bones by a gray stick, as shown in the Fig. 3.

Fig. 3: Output hand model by Leap Motion Controller

Each frame information is updated and acquired in the same form, which can achieve real-time, fast and accurate hand tracking effect.

When Leap Motion Controller is used for actual work, the work frame shown in the following Fig. 1, which is the same as the process we use in work.

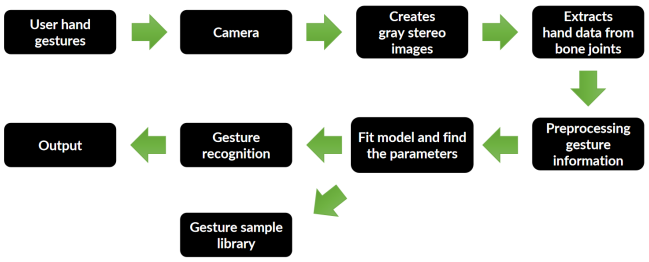


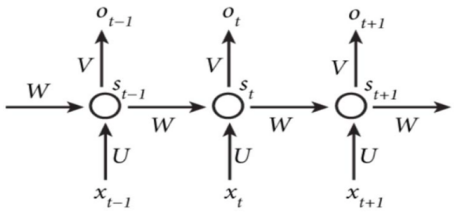
Fig. 4: Work frame of hand gesture recognition

First, the camera captures the user's gestures and forms a data frame. The gray stereo image is presented by the data frame, and the Leap Motion Controller extracts the skeleton information of the hand according to its own internal algorithm, so as to provide the position and information needed for gesture recognition. After this step, researchers need to build algorithms for recognition and classification, train the testing gestures, fit models, and find parameters that work best. Next, gesture recognition of experimental samples can be carried out, and the processed information can be stored to generate gesture sample library. The computer carries out the actual hand gesture recognition according to this process.

B. Development of neural network in hand gesture recognition

The classifiers' ability to discriminate a particular sign in any environment is crucial. Many classifiers have been used in previous studies and each classifier had its advantages and drawbacks. Many of the existing works for controlled environment tend to apply the Support Vector Machine (SVM) as a classifier. When we work with sensor data like the Leap Motion Controller and facing uncontrolled environment feature extraction becomes a critical step in the recognition tasks in order to learn relevant patterns, and neural network is more commonly used. People started from applying artificial neural network(ANN) which includes at least a hidden layer, each neuron is associated with all the neurons in the adjacent layers by sum of weight. R. Yasir. et al. [8] developed the system using ANN to recognise Two-handed hand gesture in 2014 and reach the accuracy of 70% for raw data and 100% after LDA processing. Later, convolutional neural network(CNN) is developed for hand gesture recognizing which its neurons in each layer of convolutional neural network are arranged in three dimensions: width, height and depth compared with ANN. Instead of being fully connected, neurons in the layer will only connect to a small area in the previous layer. ElBadawy et al. [9] developed the system using 3D CNN. The system used depth map data and achieved more than 90% accuracy. Yang and Zhu [10] had used the CNN for Chinese Sign Language and achieve an accuracy of 99%. The three convolutional layers CNN model extracts the upper body images from videos directly, and it is also able to recognize the gestures in the images. These are some traditional algorithm of neural network used in hand gesture recognition. Although they reached a considerable accuracy, they cannot consider previous information when dealing with a current task.

To solve this problem, people apply recurrent neural network(RNN) which memorizes the preceding information and applies it to the calculation of the current output, that is, the input of the hidden layer includes not only the output of the input layer but also the output of the hidden layer at the previous moment and its structure is shown in Fig. 5.



$$O_t = g(V \cdot S_t)$$
$$S_t = f(U \cdot X_t + W \cdot S_{t-1})$$

Fig. 5: Structure and principle of RNN

K. Lai and S. N. Yanushkevich [11] had combine RNN with CNN and this improves the accuracy from 33.24% to 75.79%. As RNN will face the problem of gradient extinction and gradient explosion, people develop long short-term memory(LSTM) which is a special kind of RNN and performs better in long sequences. As Fig. 6 shown below it add a state c to hold the long-term state and have more gates for different functions.

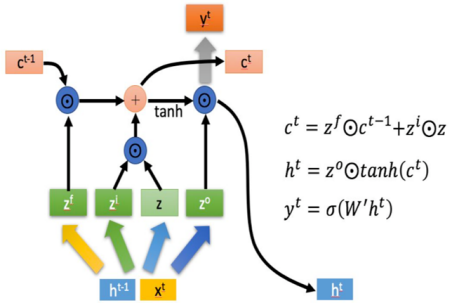


Fig. 6: Structure and principle of LSTM

C. R. Naguri and R. C. Bunescu [12] set models for dynamic hand gestures with 3D motion data in 2018. They also compared LSTM with CNN and get the average accuracy of 98.4% which is 1.2% higher than using CNN. Recent years people are still working on LSTM, and LSTM has developed more than ten variants. For exmaple, Deep LSTM, Tree-LSTM, conv-LSTM, Peephole LSTM, LSTM-CNN, GS-GLSTM, Bi-directional LSTM and Hybrid Bidirectional Unidirectional LSTM. Among them, Bi-directional LSTM and Hybrid Bidirectional Unidirectional LSTM are newly developed and we will discuss them more specifically later.

III. Algorithm Principle

A. Introduction To RNN Algorithm Principle

Parameters

h_0	Initial input value
h_i	The value of i hidden layer
x_i	Input value of layer i
y_i	Output value of layer i

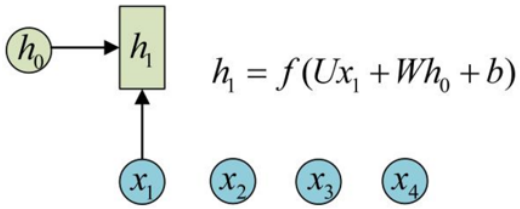


Fig. 7: Structure and principle of RNN

Variables in the circle and square are vectors, and U, W and b are the same in each step of calculation. h_0 is the initial value entered by the hidden layer. x_1 is the input value of the first layer nerve. h_1 is the hidden layer value of the first layer nerve, and the calculation method is shown in the figure.

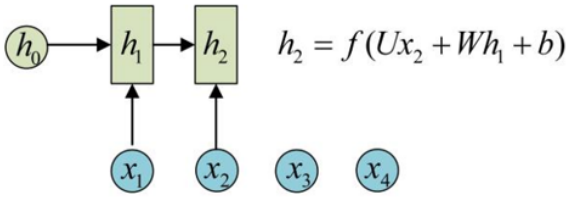


Fig. 8: Structure and principle of RNN

Later, the calculation result of h_1 is used as the input of the hidden layer of the second layer nerve, and x_2 is the input value of the second layer nerve. h_2 can be calculated by using the same method.

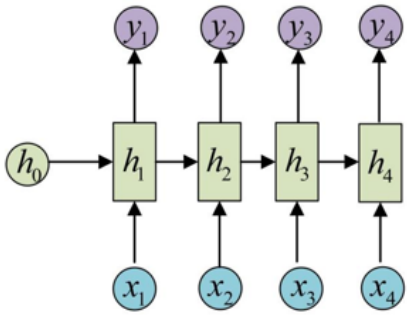


Fig. 9: Structure and principle of RNN

The hidden layer value of the following sequence can be calculated according to the above iteration method (h_3, h_4.....h_n) finally, the output of the neural network, and the output value can be directly obtained by calculating h.At this point, a classical RNN structure is established.

B. Limitations of traditional RNN constructs: Long-Term Dependency Problem.

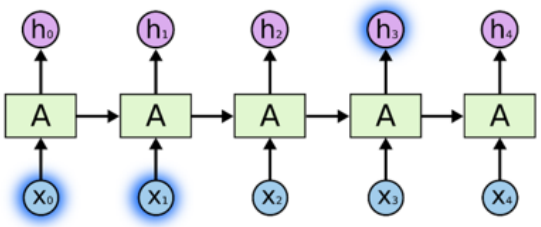


Fig. 10: Short-term memory structure of RNN

RNN can be used to connect the previous information to the current task. When the previous information is very close to the current task, the original information can be well used and accurate prediction can be made.

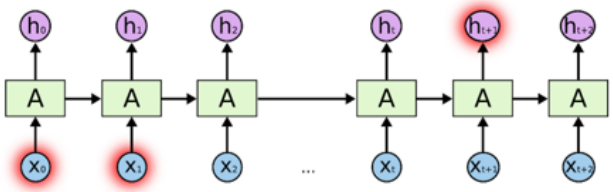


Fig. 11: Long-term memory structure of RNN

However, when the previous information is far away from the current task, RNN will dilute the weight of the information we need because it remembers too much input information. As a result, RNN will lose the ability to connect to the previous information when performing a distant task.RNN is affected by short-term memory.If a sequence is long enough, it will be difficult for them to transfer information from an earlier time step to a later time step.

C. Introduction To LSTM Algorithm Principle

Parameters	
X_t	Current cell input variable
h_{t-1}	The last cell output variable
h_t	Current cell output variable
C_{t-1}	Initial cell state
C_t	Final cell state
Forget Gate	
W_f	Forget gate operation matrix
b_f	Forget gate offset matrix
f_t	Forget gate calculation results
Input Gate	
W_i	Input gate operation matrix
b_i	Input gate offset matrix
i_t	Input gate calculation results
W_C	Input gate state change matrix
b_C	Input gate state change offset matrix
$\hat{C}_t$	Input gate state calculation results
Output Gate	
W_o	Output gate operation matrix
b_o	Output gate offset matrix
o_t	Output gate calculation results

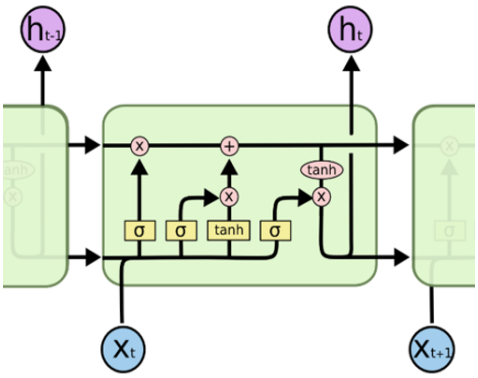


Fig. 12: LSTM neuron structure diagram

Each green module is a cell in the LSTM neural network. The cell input is the cell state C_{t-1} and h_{t-1} of the previous cell output, as well as the current cell input x_t. The output is the current cell state C_t and h_{t-1} of the hidden state.Each cell has three kinds of operation gates with different functions (Forget gate, Input gate and Output gate), which can realize the memory and forgetting of input data. Due to the limited memory capacity, remember important and forget irrelevant, which can improve the short-term memory defect of RNN network, so as to realize long-term memory.The plus sign in the figure means adding elements and the multiply sign means multiplying elements. The core of LSTM is the cell state, which runs as a horizontal line running above the

cell, connecting all cells in series, and interacting with the current cell when passing through each cell (the forgetting gate multiplies with the cell state to eliminate information and achieve the purpose of forgetting. The input gate interacts with the cell state in addition to provide new information. The output gate interacts with the cell state passively. The cell state enters the output gate as one of the inputs and outputs to the next cell through the output gate after calculation.) It is used to forget unimportant information and input new information, and then continue to flow into the next cell. The function of the three gates in the cell is realized through sigmoid and tanh mapping functions and a series of specific delicate operations.The activation function tanh helps to regulate the values flowing through the network so that they are always limited to between -1 and 1. The Sigmoid activation function represented by sigma is similar to the tanh function, except that the Sigmoid is compressed between 0 and 1 instead of -1 to 1. This helps update or forget information: because any number multiplied by 0 is 0, the information is removed; Similarly, any number multiplied by 1 yields itself, and that piece of information is perfectly preserved.

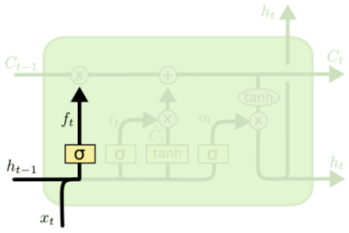


Fig. 13: Forget Gate

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$$

Forget gate reads h_{t-1} and input vector x_t, do a Sigmoid mapping, and then output a vector, the vector of each dimension value between 0 and 1, 1 is completely retained, 0 means completely abandoned, equivalent to forget the important information in the memory is not important information, the C_{t-1} multiplication and cell state.

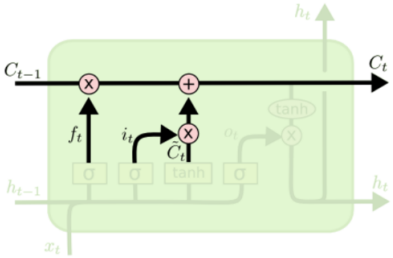


Fig. 14: Input Gate

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$

$$\hat{C}_t = tanh(W_C.[h_{t-1}, x_t] + b_C)$$

The input gate determines what new information will be stored in the cell state. First, the Sigmoid layer determines what values to update. The tanh layer creates a new vector, $\hat{C}_t$ which is added to the cell state after being multiplied by it.

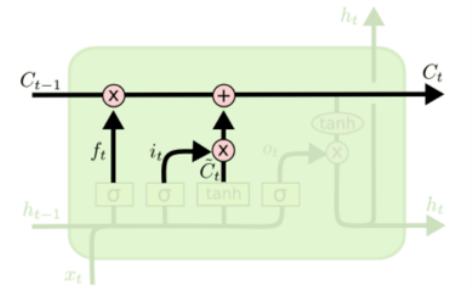


Fig. 15: Cell state change

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t$$

The old state C_{t-1} is multiplied by f_t to forget the information we need to discard, and then i_t * $\hat{C}_t$ is added to obtain the latest cell state C_t after adding new information.

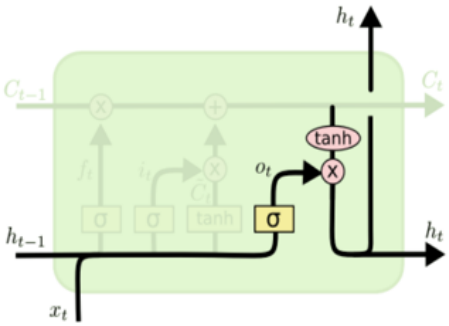


Fig. 16: Output Gate

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$$

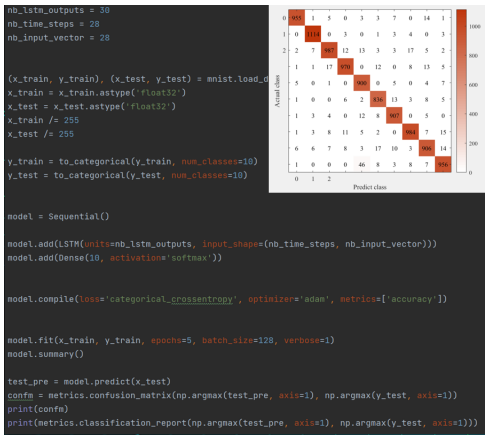
$$h_t = o_t * tanh(C_t)$$

We need to output a value to the next cell, which is calculated based on our current cell state. First, a sigmoid layer is used to determine which part of the cell state to output, then the current cell state is tanh processed and multiplied by O_t to determine the output part, and finally the output is carried out.

D. Use LSTM To Classify Handwritten Numbers

First import the official Minist data set. Next add the LSTM cycle layer and set the parameters. Then Compile the network and define loss functions and optimizer types. At last, set the number of iteration rounds and data batch size for training.

Here is the code and test results. The accuracy of classification is about 95%.



```
nb_lstm_outputs = 30
nb_time_steps = 28
nb_input_vector = 28


(x_train, y_train), (x_test, y_test) = mnist.load_d
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)


model = Sequential()

model.add(LSTM(units=nb_lstm_outputs, input_shape=(nb_time_steps, nb_input_vector)))
model.add(Dense(10, activation='softmax'))


model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])


model.fit(x_train, y_train, epochs=5, batch_size=128, verbose=1)
model.summary()

test_pre = model.predict(x_test)
confm = metrics.confusion_matrix(np.argmax(test_pre, axis=1), np.argmax(y_test, axis=1))
print(confm)
print(metrics.classification_report(np.argmax(test_pre, axis=1), np.argmax(y_test, axis=1)))
```

Fig. 17: LSTM classify handwritten numbers

E. Single Neuron Code

In this project, an LSTM neuron was reproduced by hand-written Python code, sigmoid and tanh functions, as well as forget gate, input gate, cell state change, output gate and other functions were realized by writing methods. Given the input variables and the operation matrix specific value, it can be used to simulate the workflow of a single LSTM cell.

```
# The basic parameters
h_t_1 = array([0, 0, 0, 0])_# The last cell output variable
x_t = array([1, 0, 0, 1])_# Current cell input variable
c_t_1 = array([1, 1, 0, 0])_# Initial cell state
# Forget the gate parameters
w_fh = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])_# Forget gate operation matrix
w_fx = np.array([[100, 0, 0, 0], [-100, 0, 0, 0], [0, 0, 0, 100], [0, 0, 0, 100]])_# Forget gate operation matrix
b_f = np.array([0, 0, 0, 0])_# Forget gate offset matrix
# Input gate parameters
w_ih = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])_# Input gate operation matrix
w_ix = np.array([[100, 0, 0, 0], [100, 0, 0, 0], [0, 0, 0, 100], [0, 0, 0, 100]])_# Input gate operation matrix
b_i = np.array([0, 0, 0, 0])_# Input gate offset matrix
w_ch = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])_# Input gate state change matrix
w_cx = np.array([[0, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 100]])_# Input gate state change matrix
b_c = np.array([0, 0, 0, 0])_# Input gate state change offset matrix
# Output gate parameters
w_oh = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])_# Output gate operation matrix
w_ox = np.array([[-100, 0, 0, 0], [-100, 0, 0, 0], [0, 0, 0, -100], [0, 0, 0, -100]])_# Output gate operation matrix
b_o = np.array([0, 0, 0, 0])_# Output gate offset matrix
```

```
def tanh(z):
    return (np.exp(z)-np.exp(-z)) / (np.exp(z)+np.exp(-z))


def sigmoid(z):
    return 1 / (1+np.exp(-z))


def forget_gate(W_fh, W_fx, B_f, H_t_1, X_t):
    F_t = sigmoid(np.dot(W_fh, H_t_1) + np.dot(W_fx, X_t) + B_f)
    return F_t


def input_gate(W_ih, W_ix, B_i, W_ch, W_cx, B_c, H_t_1, X_t):
    I_t = sigmoid(np.dot(W_ih, H_t_1) + np.dot(W_ix, X_t) + B_i)
    C_t0 = tanh(np.dot(W_ch, H_t_1) + np.dot(W_cx, X_t) + B_c)
    Input_vector = I_t * C_t0
    return I_t, C_t0, Input_vector


def cell_change(C_t_1, F_t, Input_vector):
    C_t = C_t_1 * F_t + Input_vector
    return C_t


def output_gate(W_oh, W_ox, B_o, C_t, H_t_1, X_t):
    O_t = sigmoid(np.dot(W_oh, H_t_1) + np.dot(W_ox, X_t) + B_o)
    H_t = O_t * tanh(C_t)
    return H_t, O_t
```

Fig. 18: Single LSTM neuron code

IV. Algorithm Architecture

In this section, we will focus on the architecture and logic of the algorithm. In this section, we discuss the existing LSTM models and its variations. Therefore, we start from the basic LSTM classification model, then the Bi-LSTM model and an extension of the basic models – HBU-LSTM model; a hybridization between the existing LSTM architectures. Some of the following information is reffered to the paper [7].

A. Unidirectional-LSTM

Considering the basic LSTM unit, it is worth mentioning that LSTM is very efficient in dealing with time series problems. As shown in Figure X, LSTM's input gate, forgetting gate and output gate are designed to control the update, forgetting and memory of information. With a gated approach, the required information is selectively passed through sigmoid σ and an element-level multiplication function

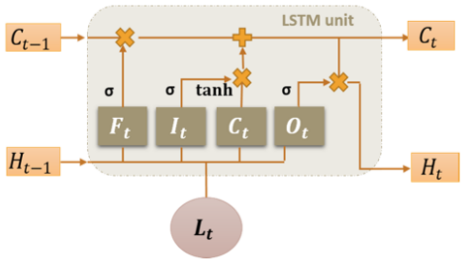


Fig. 19: Architecture of LSTM unit

In fact, the LSTM layer is made up of several LSTM cells that are connected in such a way that the output of one cell becomes the input of the next. Figure X shows a basic LSTM network, including sequence input layer, LSTM layer, full connection layer, Softmax layer, and classification output layer.

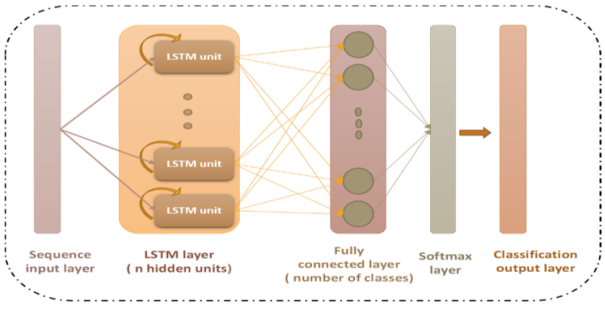


Fig. 20: Architecture of simple LSTM network

B. Bidirectional LSTM Network

In time series data, the future context is as important as the past one. Whereas, U-LSTM network gets the input sequences only from the past, Bi-LSTM network takes also the upcoming frames as described in figure x.

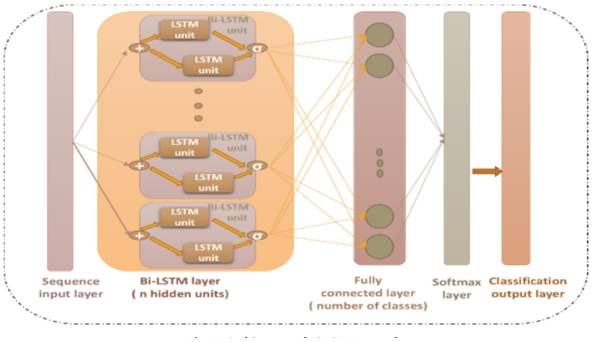


Fig. 21: Architecture of Bi-LSTM network

The BI-LSTM model was introduced as an extension of the basic LSTM model. It consists of a forward layer and a back layer that update the hidden state in reverse chronological order, as shown in figure x. In fact, BI-LSTM relies on previous frames and upcoming frames, which allows for a complete understanding of the data sequence.

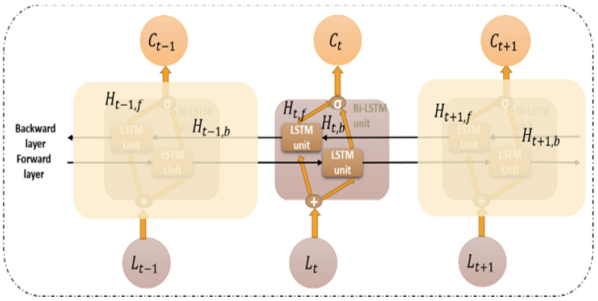


Fig. 22: General architecture of Bi-LSTM shown unfolded in time for three time steps

C. HBU-LSTM Network

we study our proposed HBU-LSTM model; a hybridization between the existing LSTM architectures. The idea came from the combination of u-LSTM and BI-LSTM layers. Therefore, we take advantage of the large dependency of past and future frames when performing sequential modeling. The output of the BI-LSTM layer serves as the input of the next LSTM layer. Figure x illustrates a novel deep architecture for handling hand gesture classification. We propose an HBU-LSTM network in which a Bi-LSTM layer is selected as the first layer of the model due to the forward and backward dependencies described above. The first feature learning layer enables us to learn more useful information from the input layer, which consists of a spatial time series of Leap Motion data, and consider the time dependence of the next layer. When classifying gestures, the last layer of the network needs to iteratively calculate weights and give output values along the forward pass using features learned from the lower layers. Therefore, as the last layer of the model, the ULSTM layer is a suitable choice because of the basic structure of the LSTM. In addition, we use two dropout layers to avoid potential overfitting problems. The active function in the output layer is Softmax. Thus, we can get the probability that the recorded gesture falls into each different category.

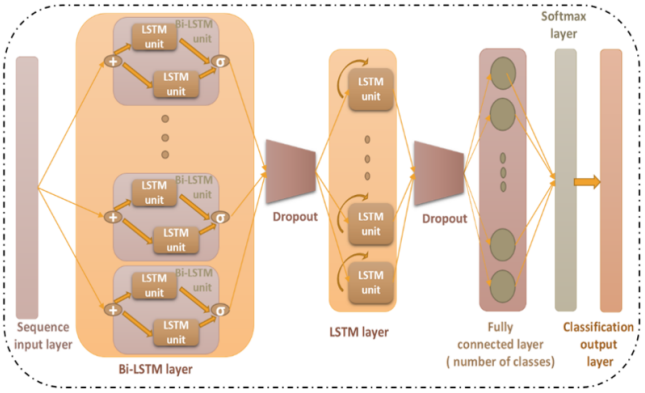


Fig. 23: Architecture of HBU-LSTM network

V. Results and Analysis

In this part, it will introduce how to obtain data, feature extraction and data structure. In addition, it will introduce how to use the three algorithms introduced in the previous part to test hand gesture classification and get the classification results and tuning results. Finally, the performance of different algorithms will be compared.

A. Data Collection and Data Structure

For this project, data is acquired through the Leap Motion Controller, as shown in figure x. The Leap Motion Controller has a good ability to obtain the skeleton information of gestures, and the position information of finger nodes is shown in figure x. Therefore, the initial data collection takes into account the 3d coordinates of the palm (x,y,z), the normal vector coordinates (x,y,z), and the coordinates of each finger node (x,y,z). Thus, the data structure of the original data is a single frame 1x93 matrix.

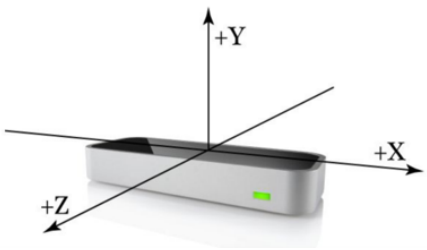


Fig. 24: Leap Motion Controller

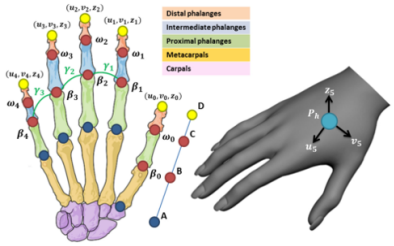


Fig. 25: finger joint of hand

In addition, in order to verify whether the data is true and effective, MATLAB was used to draw the image. As shown in figure x below, the restoration of gestures is very good, indicating the validity of our data.

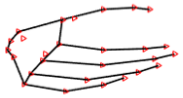


Fig. 26: MATLAB restoration hand gesture

Due to the high dimension of data, it is necessary to reduce the dimension of data. Feature extraction is a very good method. Consider the following characteristics: Fingertip distance, Fingertip Angle, Fingertip height, The distance of adjacent fingertips, The Angle of adjacent fingertips, Coordinate of the palm. Thus, the data can be dimensionally reduced to a 1x23 matrix per frame.

The following five gestures in figure x are the five gestures to be classified in this project, including pause, straight driving, left turn, right turn, and continuous right turn.

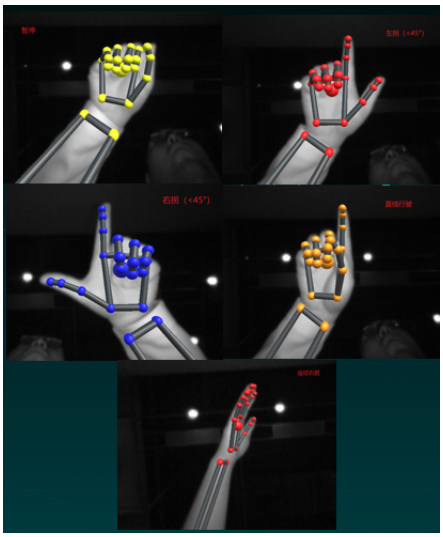


Fig. 27: hand gesture classification

B. Parameter Tuning

Consider about parameter tuning, the most important thing is hyper-parameters for different LSTM algorithms.

(1) $sequence-length*input-size$ represents single frame data;

(2) $num-layers$ represents the number of LSTM layers;

(3) $num-epochs$ means trainning once using all the samples in the training set;

(4) $hidden-size$ means the dimension size of hidden layer set;

(5) $batch-size$ means how many samples are taken for training at one time;

(6) $learning-rate$ determines whether and when the objective function converges to the local minimum;

In the process of parameter adjustment, we strictly controlled the variables and calculated the accuracy of the experiment by adopting the principle of single variable and the principle of repeated average value.

Figure X shows the influence of different $hidden-size$ on accuracy. We find that the accuracy rate is higher when $hiddern-size$ is 20-30, and the effect is best when $hiddern-size$ is 25.

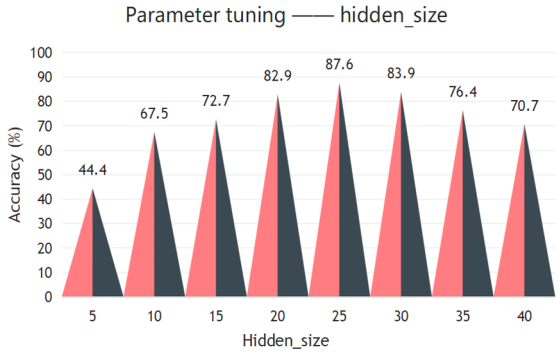


Fig. 28: Parameter tuning - $hidden-size$

Figure X shows the influence of different $learning-rate$ on accuracy. We find that the accuracy rate is higher when $learning-rate$ is 0.0001-0.01, and the effect is best when $learning-rate$ is 0.0001.

Fig. 29: Parameter tuning - $learning-rate$

Figure X shows the influence of different $num-epochs$ on accuracy. We find that the accuracy rate is stable when $num-epochs$ is from 1-10, but with the $num-epochs$ increase, it will cost more time to compute the results. So in order to balance the accuracy and time cost, we find the effect is best when $num-epochs$ is 2.

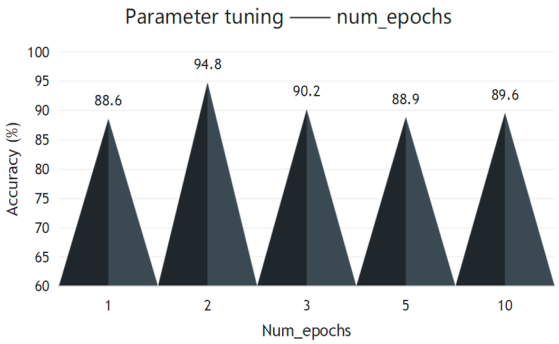


Fig. 30: Parameter tuning - $num-epochs$

Figure X shows the influence of different $batch-size$ on accuracy. We find that the accuracy rate is higher when $batch_size$ is 30-60, and the effect is best when $batch-size$ is 50.

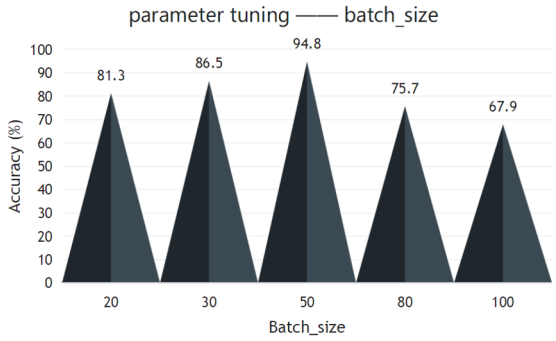


Fig. 31: Parameter tuning - $batch-size$

C. Accuracy of Results

In figure x, it shows the accuracy and the time cost of three algorithms.

Algorithm	Average Accuracy (%)	Time consuming (s)
U-LSTM	85.26%	1.08
BI-LSTM	90.37%	1.39
HBU-LSTM	94.82%	2.13

Fig. 32: accuracy and the time cost of three algorithms

We can find that HBU-LSTM algorithm has the best performance on average accuracy although its time cost is twice than U-LSTM. Comparison of algorithm effects:

(1)if we consider LSTM algorithm accuracy: we can find that BI-LSTM is better than U-LSTM and HBU-LSTM is better than BI-LSTM. HBU-LSTM do the best and it can achieve average accuracy as 94.82%

(2) Consider the degree to which the accuracy fluctuate: The accuracy of the U-LSTM varies wildly the most, occasionally approaching 90 % at best and 70 % at worst. By comparison, bi-LSTM and HBU-LSTM have much better accuracy stability than U-LSTM, and HBU-LSTM has the best stability.

When we do the analysis, we have some general ideas. (1)A single U-LSTM layer may not be efficient for capturing pertinent features. (2) A combination of U-LSTM and BI-LSTM models will lead to better results for it will exploit the forward and backward states to calculate the current state with deep training. (3) More complex model will cost more time to compute and classify. At this point, the results and correlation analysis are ended.

VI. Other LSTM Application

Besides hand gesture recognition, LSTM has a lot more applications in different areas, including translate languages, control bots, image analysis, document summaries, speech recognition image recognition, predict diseases, click-through rates and stocks, synthesize music. Some well-known companies like Google, uses LSTM to do text translation, and Google Translate uses a 7-8 layer LSTM model. Apple also uses LSTM to optimize Siri for speech recognition. The products are shown in Fig. 33.

Fig. 33: Apllication on Google and siri

LSTM is also used in weather prediction. K. Moharm et al. [13] did wind speed forecast using LSTM and Bi-LSTM algorithms. Using different gate and state activation functions they did eight experiment with different models. Compared the data with NASA in 2020-2022, prediction is shown in Fig. 34, they reached a best accuracy of 94%. And three of the models has accuracy higher than 80%. That is a really good prediction as weather is influenced by large amount of factors and usually uncertain.

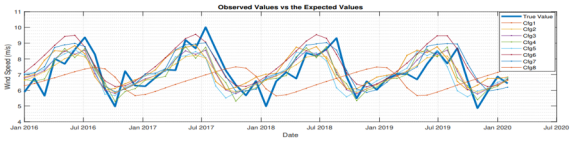


Fig. 34: Weather predicting using LSTM

In economy area, researchers uses LSTM to predict stock price. Using LSTM and Bi-Directional LSTM Model M. A. Istiake Sunny et al. [14] had done this work. The result of their predict is shown in Fig. 35. It can foresee the future value of stock price with exceptionally nonlinear demonstrating data. It

can map, and highlight the information that is required to be familiar with a function and thus, achieve a better forecasted output. For stock price prediction, LSTM can generate the highest level of accuracy compared to any other regression models. Among different Deep Learning models, both LSTM, and BI-LSTM can be used for stock price prediction with proper adjustment of different parameters. To develop any kind of prediction model, adjustment of these parameters is very important as the accuracy in prediction depends significantly upon these parameters.

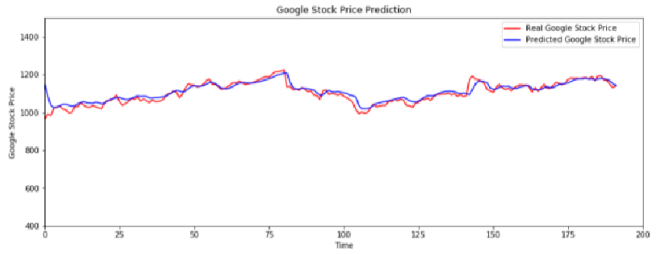


Fig. 35: Stock price predicting using LSTM

Moreover, LSTM can also be applied in medical technologies. K. Xie and Y. Wen et al. [15] uses LSTM-MA method with multi-modality and adjacency constraint for brain image segmentaion of MRI. A model for MRI is created by them and compared with several traditional mothods which are shown in Fig. 36. Moreover, they tested with pixel-wise and superpixel-wise constraint on different levels of noise in comparison with other five segmentation methods. LSTM-MA/BiLSTM-MA performs much better results when noise ratio becomes larger under each noise.

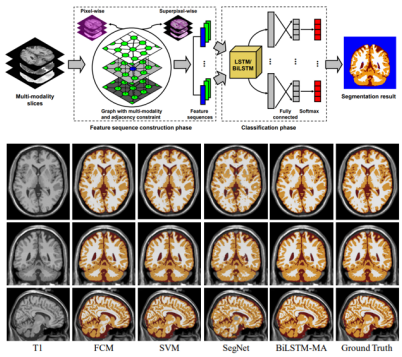


Fig. 36: MRI optimizing using LSTM

The result shows that it has great computational efficiency as well as robustness to noise and the problem of inhomogeneous intensity in image could be substantially improved.
All these applications shows a great effect on long term fairs.

VII. Extra Part: CNN

Another approach to dynamic hand gesture detection is convolutional neural network, aka CNN.

Basic CNN Algorithm Architecture

A typical CNN architecture is shown in Fig. 37. The basic CNN architecture involves five layers.

- Input layer processes the data. For example, zero-centering, normalization and PCA. These operations can reduce the noise and minish the dimension of input data, lightening the training burden of the model.
- Convolutional layer is the filter which detect the features. The reason model use convolution instead of fully connection is the parameters can be shared by multiple scalars of one example data. Imagining a 4×4 input image, the output contains 4 scalars. There will be 16×4 parameters to be trained if fully connection is used. But if convolution is applied, only a 2×2 filter is needed which just contains 4 parameters. Another reason is the model will have stronger robustness. A specific scalar in the output is decided only by a part of the input. The output value will not change if some pixels of the input shifts.
- Relu layer, or the nonlinear layer map the output nonlinearly.
- The pooling layer compresses the data further. The dimension of data is minished while the features are preserved. Two common used pooling layers are max pooling and average pooling.
- Fully connected layer collect all the features and identify the category of the data.

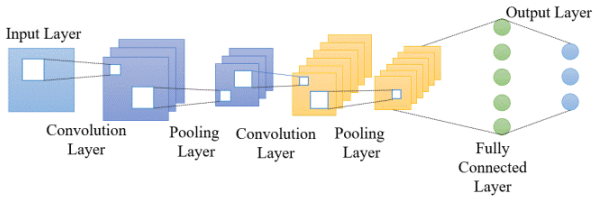


Fig. 37: The algorithm architecture of CNN.

The development of CNN

In 1989, LeCun [16] invented the convolutional neural network by combining the convolutional neural layer with the back propagation algorithm and weight sharing, and successfully applied the convolutional neural network to the handwritten character recognition system of the US Post Office for the first time.

In 1998, LeCun [17] proposed LeNet-5, the classical network model of convolutional neural network, and again improved the accuracy rate of handwritten character recognition. The architecture of LeNet-5 is shown in Fig. 38. Each unit in a layer receives inputs from a set of units located in a small neighborhood in the previous layer. After several convolutional layers, the number of units are greatly reduced.

Krizhevsky and Hinton [18] brought out AlexNet in 2012. Because of the limit of hardware, they design a double GPU structure. Thus, the AlexNet have the ability to acquire more and smaller convolutional layers. For that, the architecture of

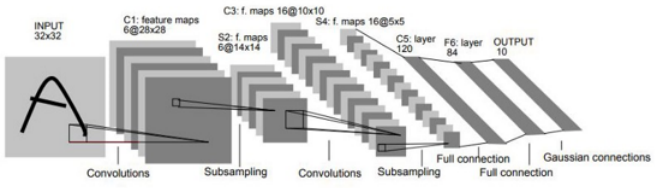


Fig. 38: Architecture of LeNet-5.

AlexNet is deeper than LeNet, as Fig. 39 shows. They won the ImageNet Large-Scale Visual Recognition Challenge of that year by a whopping 10.9 percentage points over the runner-up.

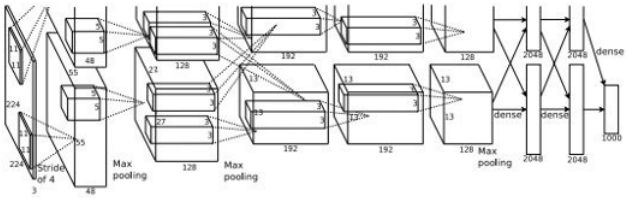


Fig. 39: Architecture of AlexNet.

Later, according to different needs, the structure of CNN developed in different directions. VGG [19] uses much deeper networks for the fitting ability of neural network model increases with the model size in theory. On the basis of VGG, GoogLeNet [20] carefully designed the local structure in the model and discarded the full connection layer because of the limitation of computing resources. R-CNN [21] complete the tasks from classification to detection. InceptionV2 [22] is a advanced version of GoogLeNet. In InceptionV2, the batch normalization layer is added and the large convolutional kernel is replaced by a series of smaller convolutional kernels. Fig. 40 shows the development process of CNN network briefly.

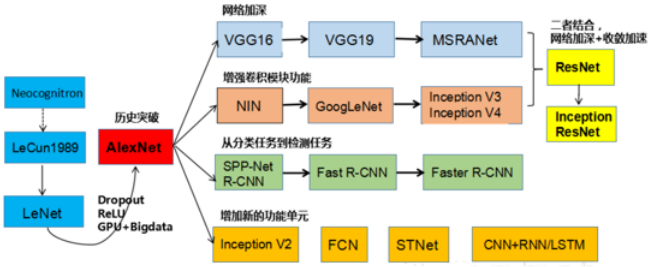


Fig. 40: CNN development sketch.

In post ImageNet era, residual learning structure are intr duced in Xception [23] and the performance are improved by more efficient use of model parameters. In 2018, MobileNetV [24] improves the state of the art performance of mobile models on multiple tasks and benchmarks as well as across a spectrum of different model sizes. It's architecture is based on an inverted residual structure where the input and output of the residual block are thin bottleneck layers opposite to traditional residual models which use expanded representations in the input an MobileNetV2 uses lightweight depthwise convolutions to filter features in the intermediate expansion layer.

Application of CNN on gesture detection

In 2022, Jimin Yu [25] propose a recognition method based on a strategy combining 2D convolutional neural networks with feature fusion. The original keyframes and optical flow keyframes are used to represent spatial and temporal features respectively, which are then sent to the 2D convolutional neural network for feature fusion and final recognition. The data processing flow is quite similar to the recognition framework mentioned before and the flow is shown in Fig. 41. To ensure the quality of the extracted optical flow graph without increasing the complexity of the network, they use the fractional-order method in Fig. 42 to extract the optical flow graph, creatively combine fractional calculus and deep learning. The structure of the algorithm is demonstrated in Fig. 43. Finally, they use Cambridge Hand Gesture dataset and Northwestern University Hand Gesture dataset to verify the effectiveness of the algorithm. The experimental results are shown in Fig. 44. The results indicate the algorithm has a high accuracy while ensuring low network complexity.

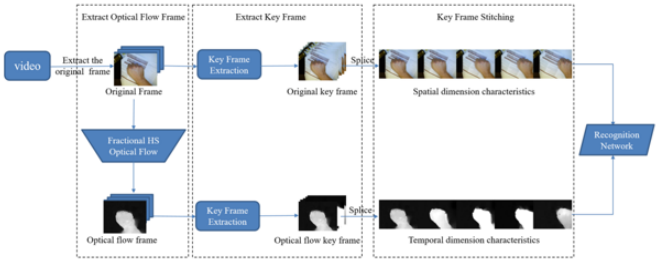


Fig. 41: Overview of the data processing flow.

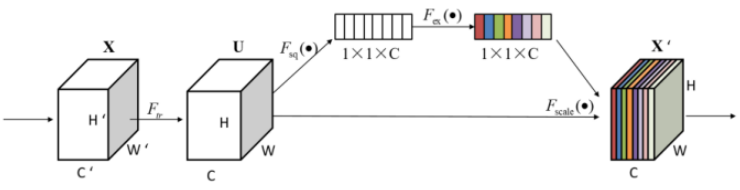


Fig. 42: A squeeze-and-excitation module.

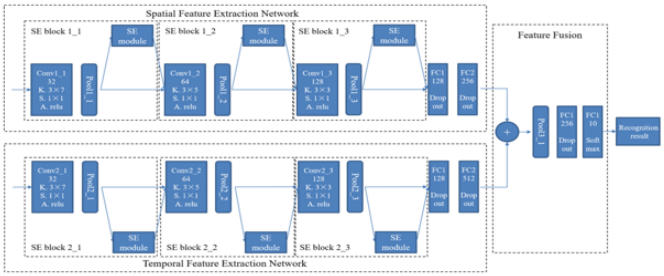


Fig. 43: Recognition network structure.

Method	SE module	Top-1 accuracy	
		Northwestern	Cambridge
Original frame + Optical flow frame	✓	**97.64%**	**98.62%**
Original frame + Optical flow frame		96.52%	97.23%
Original frame	✓	83.67%	88.47%
Original frame		81.44%	87.18%
Optical flow frame	✓	87.21%	79.13%
Optical flow frame		85.79%	78.07%

Fig. 44: Accuracy comparison of ablation experiments.

Other applications and dilemma

Besides hand gesture detection, CNN can be applied to many other areas due to the advantage on processing the large images. For example facial recognition and medical image computing.

Although many CNN algorithms have been proposed and applied to numerous applications, dilemmas still exist.

About the algorithm, CNN need too many labeled training data and the training time is too long and cost expensive computing resources. Besides, the hyperparameters, especially around nodes and layers, remain mysterious. The actual physical meaning of these parameters is unclear, resulting in no clear goal when designing the network framework.

As for the gesture detection, the image background in real world is far way complex than that in data library. The tracing and matching of dynamic gesture in real world are still difficult.

Use CNN to classify handwritten numbers

To better understand the CNN network, we use a simple 2-d convolutional network to recognize MNIST handwritten digits. We will use a very popular framework PyTorch to train the network.

We load the MNIST dataset and set the batch size to 64 for training and 1000 for testing. Then a normalization is performed to the data.

```
train_loader = torch.utils.data.DataLoader(
  torchvision.datasets.MNIST('/files/', train=True, download=True,
                             transform=torchvision.transforms.Compose([
                               torchvision.transforms.ToTensor(),
                               torchvision.transforms.Normalize(
                                 (0.1307,), (0.3081,))
                             ])),
  batch_size=batch_size_train, shuffle=True)

test_loader = torch.utils.data.DataLoader(
  torchvision.datasets.MNIST('/files/', train=False, download=True,
                             transform=torchvision.transforms.Compose([
                               torchvision.transforms.ToTensor(),
                               torchvision.transforms.Normalize(
                                 (0.1307,), (0.3081,))
                             ])),
  batch_size=batch_size_test, shuffle=True)
```

Some test examples of one batch are shown in Fig. 45

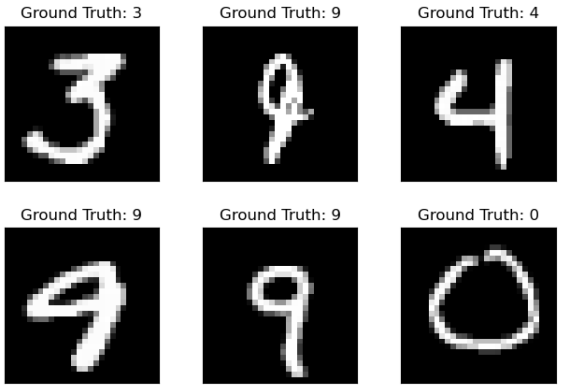


Fig. 45: Test examples of one batch.

We use two 2-D convolutional layers followed by two fully-connected layers. As activation function we choose ReLUs and as a means of regularization we use two dropout layers.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
        self.conv2_drop = nn.Dropout2d()
        self.fc1 = nn.Linear(320, 50)
        self.fc2 = nn.Linear(50, 10)

    def forward(self, x):
        x = F.relu(F.max_pool2d(self.conv1(x), 2))
        x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
        x = x.view(-1, 320)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

We iterate over all training data once per epoch and calculate the accuracy for test data after each epoch. As we can see, We started out with randomly initialized parameters and as expected only got about 10% accuracy on the test set before starting the training. With just 3 epochs of training, we managed to achieve 97% accuracy on the test set. the accuracy

is improved after each epoch. The loss with respect to the number of samples being trained are shown in Fig. 46.

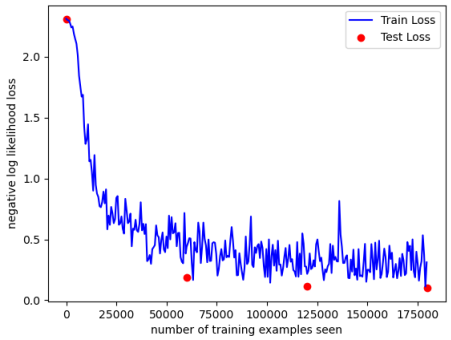


Fig. 46: Loss with respect to number of training examples seen.

The prediction of the example data given by the model is shown in Fig. 47

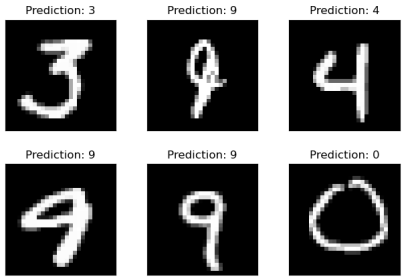


Fig. 47: Prediction of the example data.

VIII. Conclusion

This paper discusses the process of hand gesture recognition, the technical structure, the use of sensors for hand gesture recognition and the development of neural networks in hand gesture recognition. Also, it introduces the basic principles and neuronal structure of RNN, and further focus on the LSTM algorithms in RNN. Following the guidance of a paper, members in our team deeply studyed unidirectional LSTM network, bidirectional LSTM network and HBU-LSTM network. Finished code writing, algorithm reproduction, parameter tuning and achieved considerable accuracy. Moreover, CNN is also explored in the last part of paper as another approach to dynamic hand gesture recognition. The architecture, development and application of CNN are discussed and it is applied to classify handwritten numbers. Finally we completed the tasks very well and deeply realized the importance of doing research on hand gesture recognition!

References

[1] S. Jiang, P. Kang, X. Song, B. P. L. Lo and P. B. Shull, "Emerging Wearable Interfaces and Algorithms for Hand Gesture Recognition: A Survey," in IEEE Reviews in Biomedical Engineering, vol. 15, pp. 85-102, 2022, doi: 10.1109/RBME.2021.3078190.

[2] S. M. Mankar and S. A. Chhabria, "Review on hand gesture based mobile control application," 2015 International Conference on Pervasive Computing (ICPC), 2015, pp. 1-2, doi: 10.1109/PERVASIVE.2015.7087125.

[3] S. K. Keshari, S. Tyagi, N. Tomar and S. Goel, "Aphonic's Voice: A Hand Gesture Based Approach to Convert Sign Language to Speech," 2019 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT), 2019, pp. 1-4, doi: 10.1109/ICICT46931.2019.8977690.

[4] N. Mohamed, M. B. Mustafa and N. Jomhari, "A Review of the Hand Gesture Recognition System: Current Progress and Future Directions," in IEEE Access, vol. 9, pp. 157422-157436, 2021, doi: 10.1109/ACCESS.2021.3129650.

[5] J. S. Sonkusare, N. B. Chopade, R. Sor and S. L. Tade, "A Review on Hand Gesture Recognition System," 2015 International Conference on Computing Communication Control and Automation, 2015, pp. 790-794, doi: 10.1109/ICCUBEA.2015.158.

[6] M. Kavian and A. Nadian-Ghomsheh, "Monitoring Wrist and Fingers Range of Motion using Leap Motion Camera for Physical Rehabilitation," 2020 International Conference on Machine Vision and Image Processing (MVIP), 2020, pp. 1-6, doi: 10.1109/MVIP49855.2020.9116876.

[7] Ameur, S., Ben Khalifa, A. and Bouhlel, M.. "A novel hybrid bidirectional unidirectional LSTM network for dynamic hand gesture recognition with Leap Motion," Entertainment Computing, 35, p.100373, 2020.

[8] R. Yasir and R. A. Khan, "Two-handed hand gesture recognition for Bangla sign language using LDA and ANN," The 8th International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2014), 2014, pp. 1-5, doi: 10.1109/SKIMA.2014.7083527.

[9] M. ElBadawy, A. S. Elons, H. A. Shedeed, and M. F. Tolba, "Arabic sign language recognition with 3D convolutional neural networks," in Proc. 8th Int. Conf. Intell. Comput. Inf. Syst. (ICICIS), Dec. 2017, pp. 66–71.

[10] S. Yang and Q. Zhu, "Video-based Chinese sign language recognition using convolutional neural network," in Proc. IEEE 9th Int. Conf. Commun. Softw. Netw. (ICCSN), May 2017, pp. 929–934.

[11] K. Lai and S. N. Yanushkevich, "CNN+RNN Depth and Skeleton based Dynamic Hand Gesture Recognition," 2018 24th International Conference on Pattern Recognition (ICPR), 2018, pp. 3451-3456, doi: 10.1109/ICPR.2018.8545718.

[12] C. R. Naguri and R. C. Bunescu, "Recognition of Dynamic Hand Gestures from 3D Motion Data Using LSTM and CNN Architectures," 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), 2017, pp. 1130-1133, doi: 10.1109/ICMLA.2017.00013.

[13] K. Moharm, M. Eltahan and E. Elsaadany, "Wind Speed Forecast using LSTM and Bi-LSTM Algorithms over Gabal El-Zayt Wind Farm," 2020 International Conference on Smart Grids and Energy Systems (SGES), 2020, pp. 922-927, doi: 10.1109/SGES51519.2020.00169.

[14] M. A. Istiake Sunny, M. M. S. Maswood and A. G. Alharbi, "Deep Learning-Based Stock Price Prediction Using LSTM and Bi-Directional LSTM Model," 2020 2nd Novel Intelligent and Leading Emerging Sciences Conference (NILES), 2020, pp. 87-92, doi: 10.1109/NILES50944.2020.9257950.

[15] K. Xie and Y. Wen, "LSTM-MA: A LSTM Method with Multi-Modality and Adjacency Constraint for Brain Image Segmentation," 2019 IEEE International Conference on Image Processing (ICIP), 2019, pp. 240-244, doi: 10.1109/ICIP.2019.8802959

[16] L. Yann, B. Boser, D. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation applied to handwritten zip code recognition," Neural computation, vol. 1, no. 4, pp. 541–551, 1989.

[17] L. Yann, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks," Communications of the ACM, vol. 60, no. 6, pp. 84–90, 2017.

[19] K. Simonyan, and A. Zisserman."Very Deep Convolutional Network for Large-scale Image Recognition," Computer Vision and Pattern Recognition, 2014.

[20] C. Szegedy et al. "Going deeper with convolutions," in IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9, doi: 10.1109/CVPR.2015.7298594.

[21] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 580–587, doi: 10.1109/CVPR.2014.81.

[22] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, "Rethinking the Inception Architecture for Computer Vision," in IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826, doi: 10.1109/CVPR.2016.308.

[23] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," in IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1800–1807, doi: 10.1109/CVPR.2017.195.

[24] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L. -C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520, doi: 10.1109/CVPR.2018.00474.

[25] J. Yu, M. Qin and S. Zhou. "Dynamic gesture recognition based on 2D convolutional neural network and feature fusion," Scientific reports, vol. 12, no. 1, pp. 4345–4345, 2022.