VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Mengge Hu

Facial Emotional Recognition With Deep Learning On Pepper Robot

Technology and Communication
2019

FOREWORD

This is my final thesis about facial emotional recognition with deep learning on Pepper robot. I would like to express my appreciation to all the people who have helped me during the period I fulfilled my final project.

The first person I would like to express my appreciation is my tutor Dr. Yang Liu. During the whole period, he helped me to choose the topic of my project and discussed with me patiently to ensure the structure and plan which contained the whole step that I need to follow to fulfill my final project. It is him that gave me the confidence to learn and design my project with deep learning knowledge. Under his help and guidance, I have already learned much about the deep learning and the process to design a complete project.

I am still grateful to my parents who gave me the economic and mental supports to let me study and live in Finland. Also, I would like to show my appreciation to my boyfriend who gave me much support to be confident and happy.

Finally, thanks to all the staff in VAMK for their patient guidance and help.

Mengge Hu

Vaasa, Finland

25.05.2019

VAASAN AMMATTIKORKEAKOULU

VAASA UNIVERSITY OF APPLIED SCIENCES

Degree Program in Information Technology

ABSTRACT

Author	Mengge Hu
Title	Facial Emotional Recognition With Deep Learning On Pepper Robot
Year	2019
Language	English
Pages	59
Name of Supervisor	Yang Liu

There are millions of people in the world, who suffer from life-threatening diseases each year. It has been reported that the magnitude of the disparities of healthcare quality worldwide is still significant. In particular, things can lead to high mortality, such as inadequate doctor-patient ratio and lack of intelligent medical facilities. This situation, however, has been largely alleviated by the rapid developments in fields of Computer Aided Design (CAD) and Robotics. For example, the computer-aided machine can help doctors analyze radiological images and make diagnosis more quickly and accurately; the medical automatic system offers patients a broad and wide access to medical resources in a more effective way. This project is the use of robots for capturing and identifying the patient's expression, through the deep learning of the computer to determine the patient's mental state.

The main research works concludes six contents, face detected by photos, resolved by Open Source Computer Vision Library (OpenCV), facial dataset, resolved by Kaggle’s Fer2013, the processing training and testing the facial dataset, resolved by the model Convolutional Neural Network (CNN), showing the result by a bar chart is the next one, resolved by Matplotlib and taking pictures for patients, resolved by one of Pepper’s modules, ALPhotoCapture and face detected by the robot, resolved by one of Pepper’s modules, ALFaceDetection.

Keywords: PEPPER robot, Convolutional Neural Network, Facial Emotional Recognition

CONTENTS

LIST OF FIGURES AND TABLES

1 INTRODUCTION

With the rapid development of society, people are under increasing pressure. Nowadays, people, whether children, teenagers, adults or the elderly, are generally in a bad state of mind, usually ignored by themselves or ignored by those around them. As a Yle survey reported, around 90,000 kids who are all under 13 years old suffer from some mental health issues, while a shortage of care providers was witnessed through the whole the country. According to words from experts, there is a growing group which has been diagnosed and treated for minor or moderate mental health issues. At the same time, with the popularization and application of artificial intelligence, the method, using this technology reasonably, begins popularity to ease the needs for human resources. This project aims at recognizing the patients' mental change, especially at the children's mental change, as a result of it, people can realize the emotional change that they may ignore sometimes. So in this project, the module of Convolutional Neural Network is built for training, testing and detecting patients' facial emotion changes, then identifying and recognizing the patients' mental statues with the Pepper Robot.

1.1 Related Technologies and Development Environment

1.1.1 Deep Learning

Deep Learning, a learning algorithm (Learning Algorithm), is an important branch of the field of artificial intelligence. From rapid development to practical application, in just a few years, deep learning has upended the algorithm design ideas in many fields, such as speech recognition, image classification and text comprehension, and gradually formed a new model from the training data, through an end-to-end (End-to-end), and then directly output the final results.

The essence of deep learning is to learn useful features via constructing machine learning models with many hidden layers and lots of training or testing data, so as to improve the accuracy of classification or prediction. Distinguished from the traditional shallow learning, the difference in deep learning lies in:

i. The depth of the model structure is emphasized, usually, there are 5 layers, 6 layers, or even 10 layers of hidden layer nodes;
ii. The importance of feature learning is clearly highlighted. The characteristic transformation of the sample in the original space is transformed into a new feature space, which makes classification or prediction easier.

Compared with the method of constructing characteristics of artificial rules, using big data to learn features and describe the intrinsic information rich in data, at present, deep learning has a wide range of applications in search advertising CRT prediction, natural language processing, image recognition, speech recognition, and unmanned driving.

1.1.2 Artificial Neural Network's History

In 1943, neurologists and neuron anatomy McCulloch and mathematician Pitts published articles in the Journal of Biophysics presenting mathematical descriptions and structures of neurons. It is also proved that any computational function (M-P model) can be simulated as long as there are enough simple neurons that are connected to each other and run synchronously. The pioneering work they do is considered to be the starting point of artificial neural networks (ANN). In 1949, physiologist Hebb published behavioral histology, describing the Hebb adjustment rules for neuronal weights. He points out that in neural networks, information is stored in connection weights. It is suggested that the connection right of neuron *A* to neuron *B* is the same as the connection right from *B* to *A*.

In 1958, Rosenblatt, a computer scientist, proposed a neural network structure with three layers of network characteristics, called a "*perception*". The perceptron he proposed may be the first real artificial neural network in the world. After the sensor was put forward, the first craze of neural network research was set off in the 60s. Many people think that as long as they use thousands of neurons, they can solve all the problems. In 1969, Minsky and Papert, who are among the founders of Artificial Intelligence, published a book called "Perceptron," which states that simple neural networks can only be used to solve linear problems, and that networks capable of solving nonlinear problems should have hidden layers, In theory, it is not proved that it is meaningful to extend the Perceptron model to a multi-layer network.

In June 1987, the first International Neural Network Academic conference was held in Santiago, California, USA, where more than 1,600 people were represented. This was followed by an annual international academic conference organized jointly by the International Society for Neural Networks and the International Institute of Electrical Engineers and Electronic Engineers (IEEE). Later than 1987, the neural network flourished. Especially in recent years, showing an outbreak trend, neural networks began to be used in all areas of life, and a variety of new neural network models are constantly proposed. For instance, a variety of image recognition, speech recognition records are constantly refreshed. Artificial intelligence has now become a popular topic. [1]

1.1.3 Simple Artificial Neural

Artificial neurons are simple simulations of human nerve cells using a mathematical model. Human nerve cells have multiple dendrites and an elongated axon, shown in Figure 1. The axons of one neuron connect to the dendrites of other neurons and transmit nerve pulses to them. A neuron determines whether a nerve pulse is emitted from its axons to other neurons based on the signals of several dendrites from it.

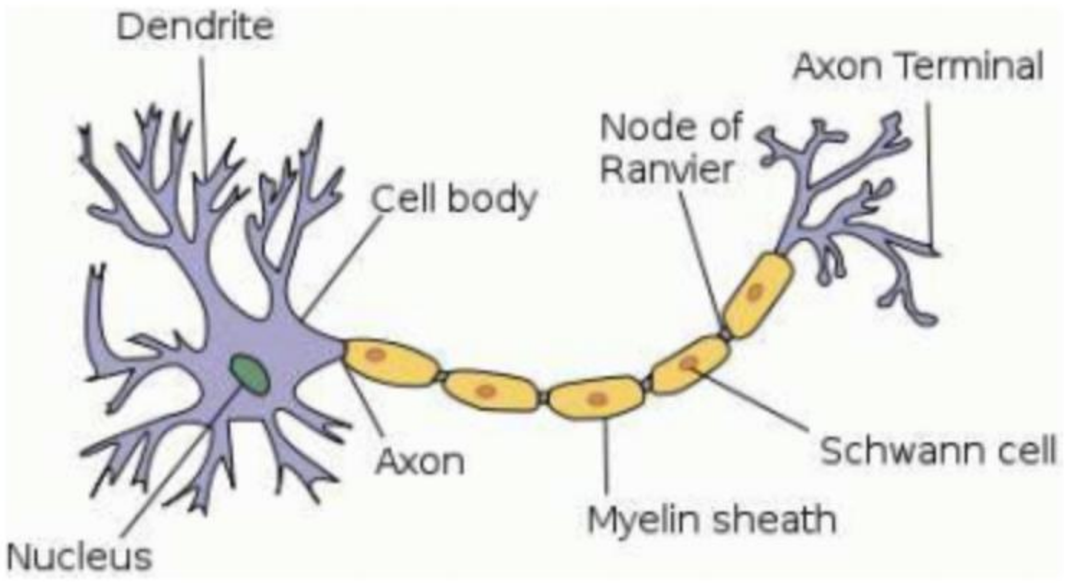


Figure 1. Biological Neural [2]

An artificial neuron is the mathematical modeling of biological neurons. See Figure 2.

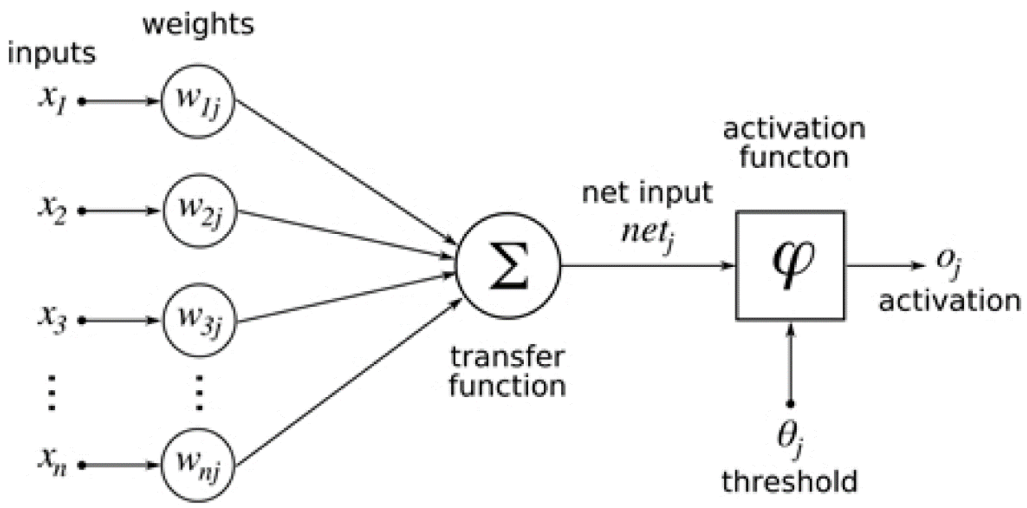


Figure 2. Artificial Neural [3]

$P_1, P_2, \cdots P_n$ is the input of artificial neurons. a is the output of an artificial nerve element. The artificial nerve element will be entered $P_1, P_2, \cdots P_n$, weighted summation is followed by a bias value *b*, finally applied to a function *f*, that is:

$$\begin{aligned} a = f(n) = f(\sum_{i=1}^{n} p_i w_i + b) &= f((w_1, w_2, \cdots w_n)\begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix} + b) \\ &= f(W^T P + b) \end{aligned} \tag{1}$$

The upper formula is finally the vector form of the formula. *P* is the input vector, *W* is the weight vector, and *B* is the bias value scalar, and *f* is called an "activation function." Activation functions can take many forms.

1.1.4 Artificial Neural Network

- Single Layer Neural Network

In this kind of neural network, the input-output mapping relationship is logical regression, as shown in Figure 1.3.

i. p_i is the input value and *b* is the bias.

ii. The connection strength is represented by the weight value $w_i \in \mathcal{R}$. If $w_i >$

0, the connection is activated, else closed.

iii. Σ is represented by the weighted sum of each input signal.

iv. *f* is represented by a nonlinear transfer function, nonlinear mapping which means the output amplitude will be limited to a certain range.

$$\begin{cases} n = \sum_{i=1}^{n} w_i p_i \\ output = f(n) \end{cases} \tag{2}$$

The threshold function:

$$\mathrm{f} = \mathrm{sng}(p - \theta) \tag{3}$$

The commonly used nonlinear activation functions are *Sigmoid*, *Tanh*, *Relu* and so on, the sigmoid and *tanh* functions are more common in the full connection layer, the latter *Relu* is common in the convolution layer. The *Sigmoid* function:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

The function of the *Sigmoid* function is to compress a real number between 0 and 1. When x is a very large positive number, $f(x)$ will approach 1, and when x is a very small negative number, $f(x)$ will approach 0. The use of this compression is to think of the activation function as a "probability of classification", for example, if the output of the activation function is 0.9, it can be explained that the probability of 90% is a positive sample. Therefore, the graph of the *Sigmoid* function $f(x)$ is represented as following in Figure 3. (the horizontal axis represents the definition domain x, and the longitudinal axis represents the value field $f(x)$):

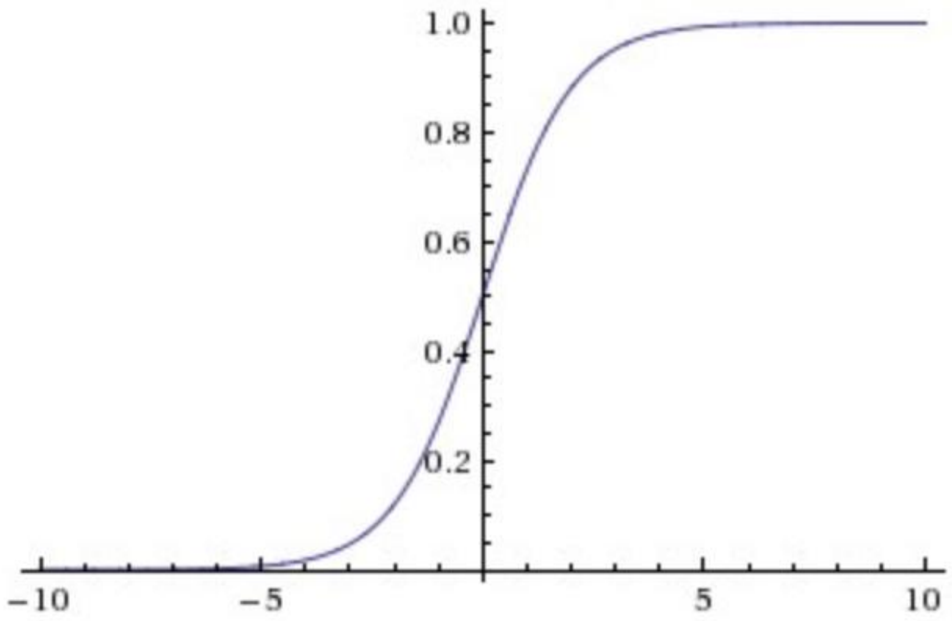


Figure 3. The Graph of the Sigmoid Function [4]

It was proposed in 1958 that, unlike the MP model, weights were variable so that learning could be carried out. It contains a linear accumulator and a two-valued threshold element (the activation function is a threshold function), and also includes an external deviation *b*. Single-layer perceptron is designed to classify inputs two, when the sensor output + 1 o'clock, the input is a class, and when the output is-1, the input is another category. Then there is the single-layer perceptron that applies the *LMS* algorithm, shown in Figure 4.

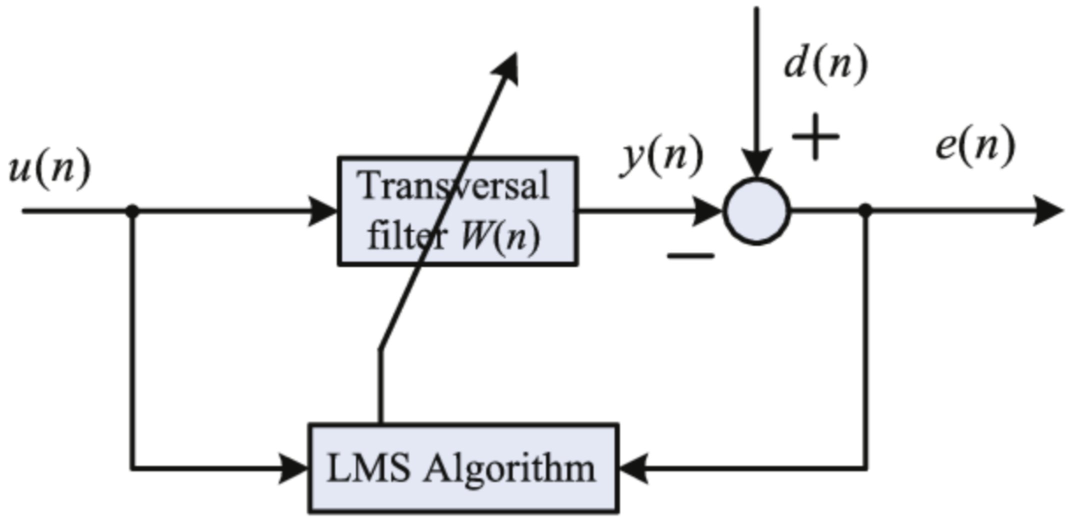


Figure 4. LMS algorithm [5]

- Multilayer Perceptron (MLP)

Multilayer perception is to add several hidden layers between the input layer and the output layer Because their values cannot be observed in the training sample, it is called the hidden layer, shown as Figure 5.

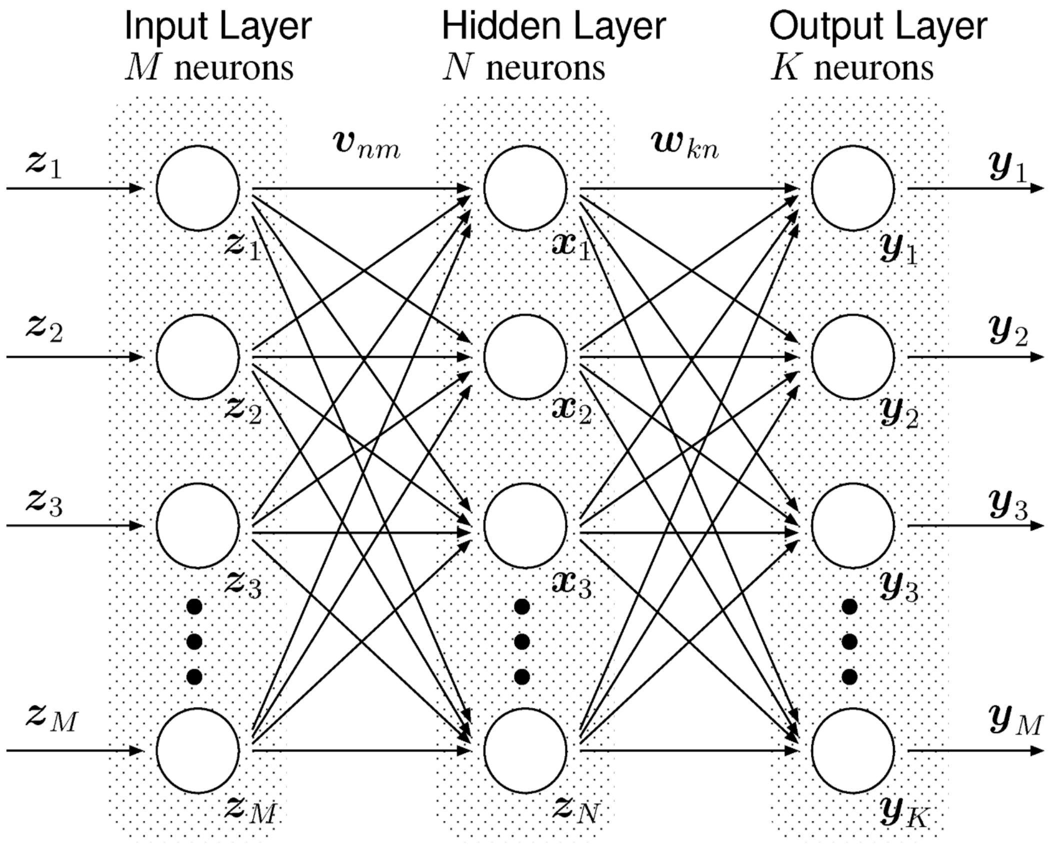


Figure 5. Multilayer Perceptron [6]

As shown in Figure 5, the output of one neuron is input to another. Neural networks can have a wide variety of topologies. One of the simplest is "multi-layer fully connected forward neural network". Its input is connected to each neuron in the first layer of the network. The output of each neuron in the previous layer is connected to the input of each neuron in the next layer. The output of the last layer of neurons is the output of the entire neural network. The calculation of the whole neural network can be given in matrix type. A single-layer is given to formula for artificial neural networks. The number of neurons per layer is different, the input/output dimensions are not the same, the number of matrices and vectors in the formula is not the same, but the form is consistent. Suppose this layer is the first layer. It accepts input and has a neuron (output), so the calculation of this layer is shown in the following formula:

$$O^i = \begin{pmatrix} o_1^i \\ \vdots \\ o_n^i \end{pmatrix} = f(\begin{pmatrix} w_{11}^i & \cdots & w_{1m}^i \\ \vdots & \ddots & \vdots \\ w_{n1}^i & \cdots & w_{nm}^i \end{pmatrix} \begin{pmatrix} o_1^{i-1} \\ \vdots \\ o_m^{i-1} \end{pmatrix} + \begin{pmatrix} b_1^i \\ \vdots \\ b_n^i \end{pmatrix}) \tag{5}$$

In the training of MLP, the random gradient drop of minibatch is generally used, and the inverse error conduction algorithm is used to realize the training of parameters after the gradient is obtained. The classical neural network training algorithm is the reverse propagation algorithm (BP, Back Propagation). BP algorithm belongs to the gradient descent method (Gradient Descend) in optimization theory. The error *e* is used as a function of all weights *W* and all bias values *B*. The purpose of the algorithm is to find the global minimum point of *e* in the independent variable space.

Shown in Figure 6, this illustrates the process about how to train a very simple neural network with MATLAB. It has only a single input single output. There are two neurons in the input layer and one neuron in the output layer. The entire network has 4 weights plus 3 bias. The figure shows the fixed other weights, the MSE surface when variables are made only for the weights $\omega_{(1,1)}^1$ and bias b_1^1 of the first neuron of the first layer, and the trajectory of the solution as the algorithm iterates. [7]

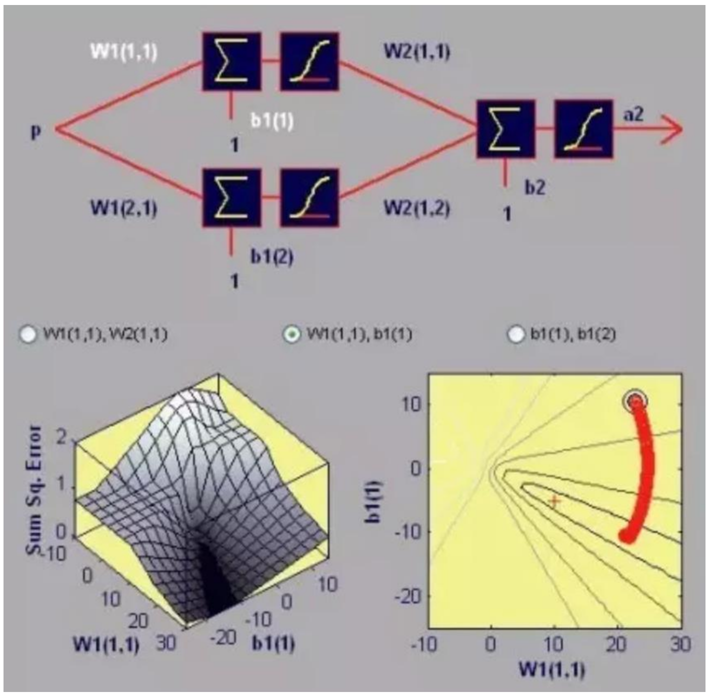


Figure 6. A Neural Network with MATLAB [7]

1.1.5 Convolution Neural Network (ConvNet/CNN)

CNN is proposed as a framework for deep learning to minimize data processing requirements. The biggest feature of CNN is sparse connectivity and weight sharing. Convolution Neural Network is a deep feedforward neural network with a local connection, weight sharing, and other characteristics. Convolution Neural Network includes convolution layers (conv), pooling layers (pool) and fully-connected layers (FC). The classical model of convolution neural network is the forward conduction process of LeNet-5 implementation to classify handwritten digits like shown in Figure 7.

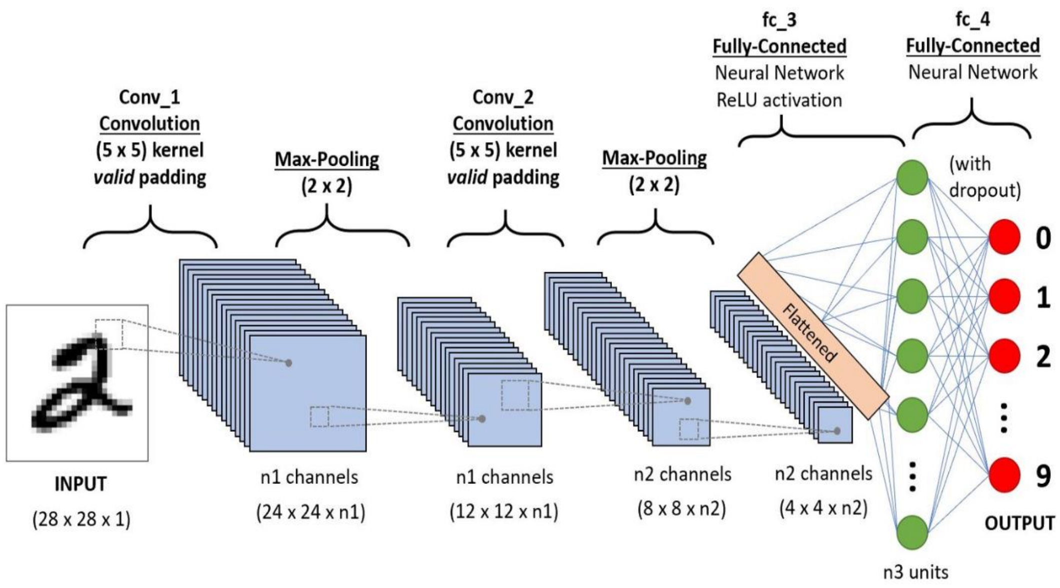


Figure 7. CNN Sequence to Classify Handwritten Digits [8]

- Convolution

Convolution is a mathematical operator that generates a third function through two functions *f* and *g*, characterizing the area of the overlapping part of the function *f* and *g* that has been flipped and transferred peacefully. The mean of convolution is weighted overlay. For instance, the physical meaning of convolution is the weighted superposition of one function (for example, Unit response) on another function, such as an input signal.

On the two-dimensional, there are two $\mathcal{R}^2 \to \mathcal{R}$ functions $f(x, y)$ and $g(x, y)$. The convolution of f and f is a new $\mathcal{R}^2 \to \mathcal{R}$ function. Get through the following formula:

$$c(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(s, t) * g(x - s, y - t)\, ds\, dt \qquad (6)$$

The meaning of this formula is: Traverse all *s* and *t* values from negative infinity to positive infinity and multiply the value of *g* at position $(x - s, y - t)$ with the value of *f* at the (*s*, *t*), "plus" (in the integral sense) together, that is the value of *c* in (*x*, *y*). In other words, convolution is a kind of "weighted summation". With (*x*, *y*) as the center, multiply the value of *g* from the center (-*s*, -*t*) position on the value of *f* at (s, t) and add it together at last. It is clearer to write the convolution formula in the discrete form:

$$\mathcal{C}(x, y) = \sum_{-\infty}^{\infty}\sum_{-\infty}^{\infty} \mathcal{F}(s, t) * \mathcal{G}(x - s, y - t) \tag{7}$$

The convolution process shows in Figure 8.

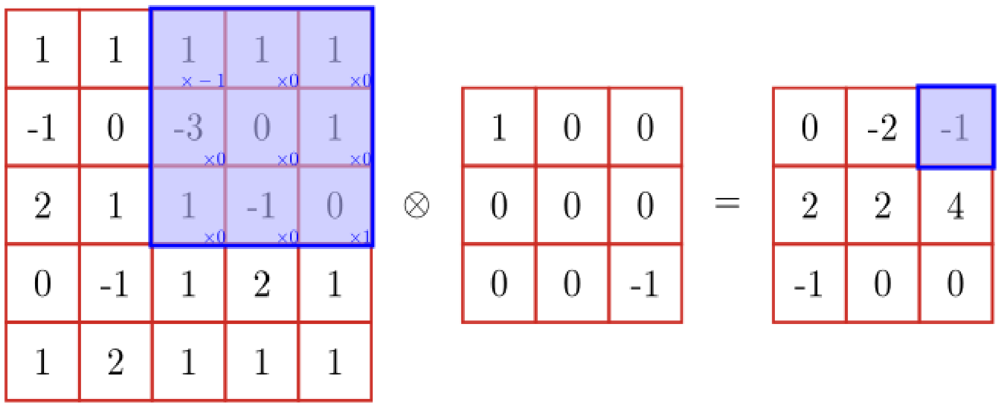
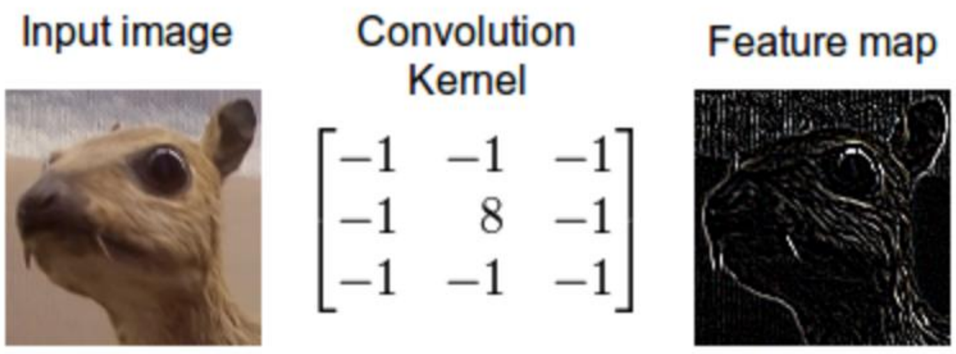


Figure 8. A Step of Convolution Process [9]

The above operation is a discrete convolution operation of digital images, also known as filtering, called convolution nuclei or filters. Different filters play different roles. Imagine that if the size of $\mathcal{F}$ is 3x3, the value in each lattice is $\frac{1}{9}$. Then the filter is equivalent to calculating the grayscale average of 9 image points in the 3x3 range around it for each point of the original image. This should be a blur, shown in Figure 9.



Input image
Convolution Kernel
$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
Feature map

Figure 9. Example of Convolution Process in Picture [10]

- Pooling

Pooling is the image of simple drawback. CNN's pooling (image sampling) methods are many: *Mean pooling*, *Max pooling* (maximum sampling), *Overlapping*, *L2 pooling*, *Local Contrast Normalization* (normalized sampling), *Stochasticpooling*, *Def-pooling* (deformation constraint sampling). One of the most classic is the maximum pool shown in Figure 10.

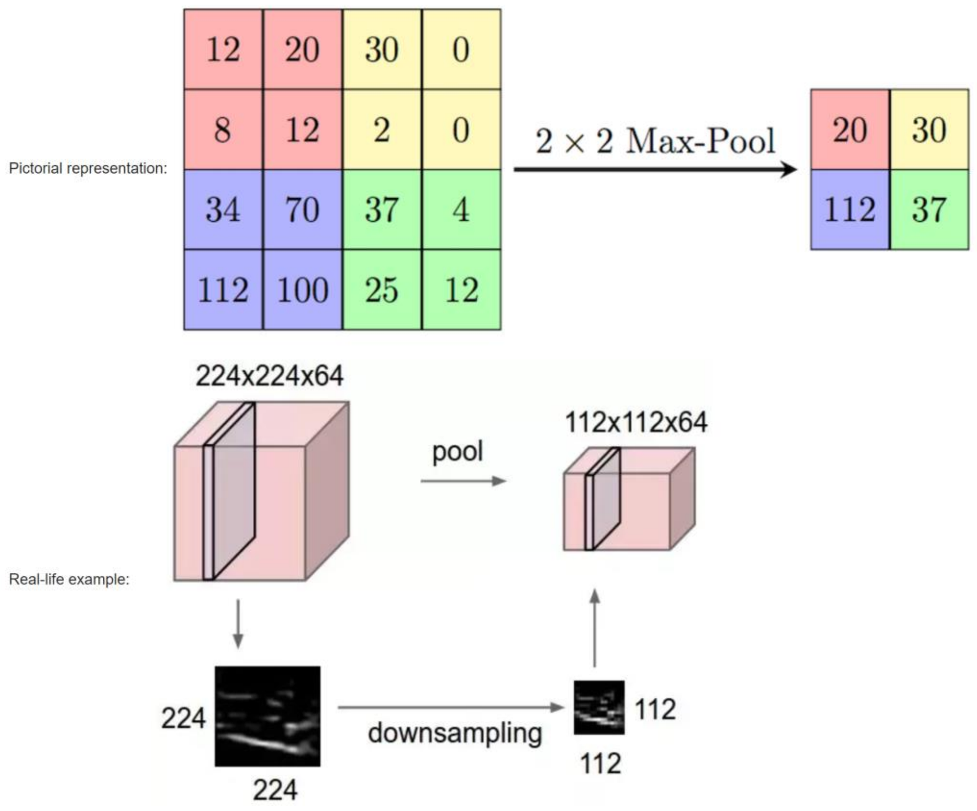


Figure 10. Example of Pooling [11]

- Fully-Connected

The full-connection layer plays the role of "classifier" in the whole convolution neural network. The full-connection layer acts as the "distributed feature representation" that is learned to map to the sample markup space. In practical use, the full connection layer can be realized by convolution operation: The full connection layer with the front layer is fully connected can be converted into convolution core 1x1, while the front layer is the whole connection layer of the convolution layer can be transformed into the global convolution with convolution core as *h* * *w*, and *h* and *w* are the high and wide of the convolution results of the front layer respectively.

The core operation of full connection is the product of the matrix vector:

$$\mathcal{Y} = \mathcal{W}x \tag{8}$$

In CNN, full connectivity often appears in the last few layers and is used to weighting and characterize the features of the previous design. For example, LeNet, the front convolution, and pooling are equivalent to doing feature engineering, the back of the full connection is equivalent to doing feature weighting. [9]

1.2 TensorFlow

TensorFlow is used to quickly implement various algorithm formulas such as DL and CNN. Its name itself describes its own execution principle: tensor means an n-dimensional array, flow (stream) means a calculation based on a stream diagram. The graphs in the flow diagram are all the directed graphs. There are two basic elements in this data structure: nodes and edges. These two elements have their own functions in this data flow diagram. Node is used to represent the mathematical operation to be performed. All operation has its own input/output, so it can also represent the end of the input or the entry of the output. Edges are represented by the relationship between input and output in nodes. A special type of data passes along with these edges. This particular type of data is called tensor in TensorFlow, a multidimensional array. When a tensor is put into this diagram, the actions represented by the node are assigned to the computing device to complete the calculation, shown in Figure 11.

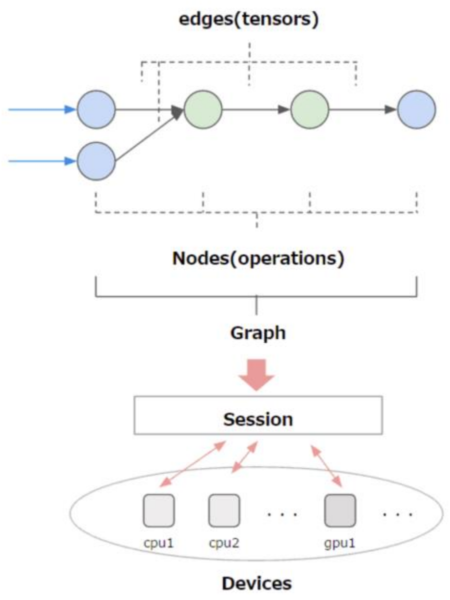


Figure 11. TensorFlow Core Graph [12]

1.3 Keras

Keras is a high-level neural network's API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

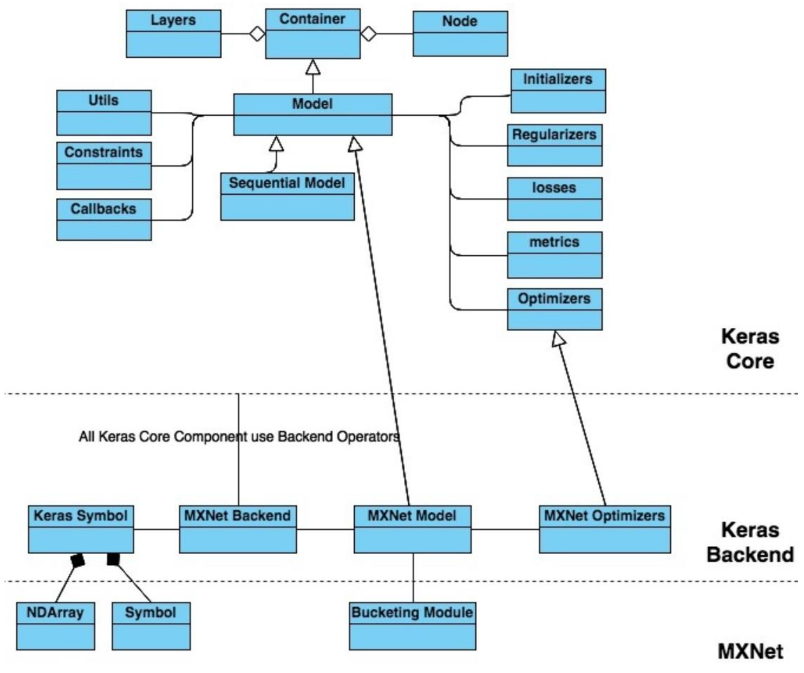


Figure 12. Keras Modules Structure [14]

1.4 Pepper Robot

Pepper is a social humanoid robot, designed for people through friendly conversations and its tablet, manufactured by SoftBank Robotics, which can be used for businesses and education. Pepper can create its own experiences and form real relationships shown in Figure 13, and its dimensions' parameters are shown in Figure 14 and Figure 15.

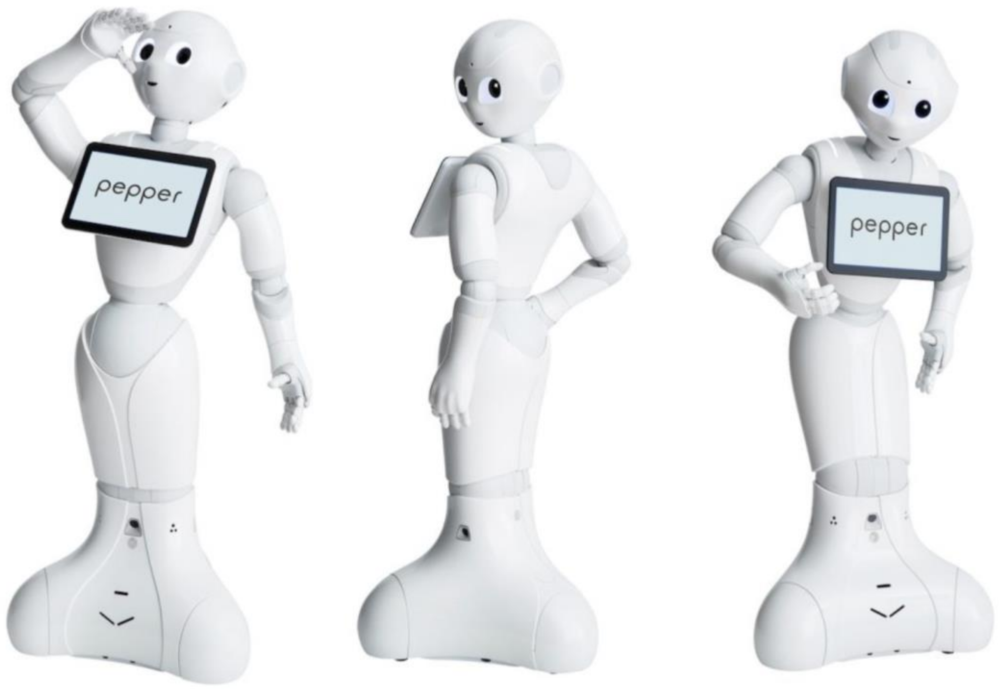


Figure 13. Pepper Robot [15]

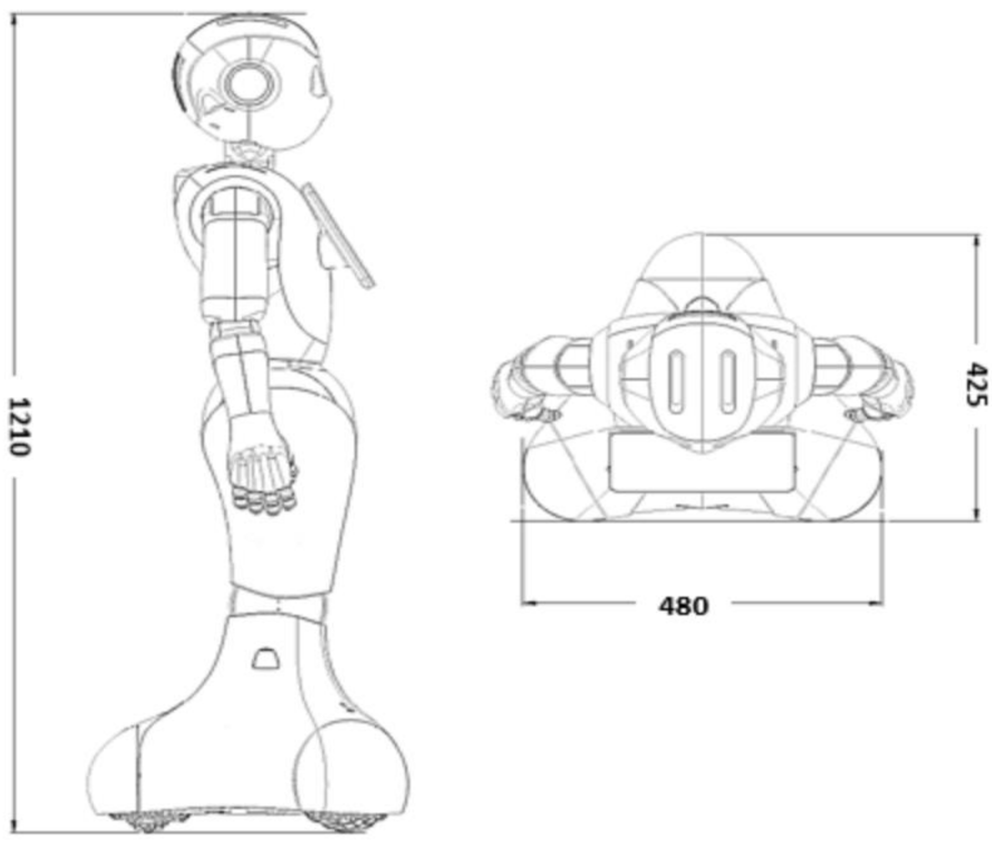


Figure 14. Pepper Dimensions [16]

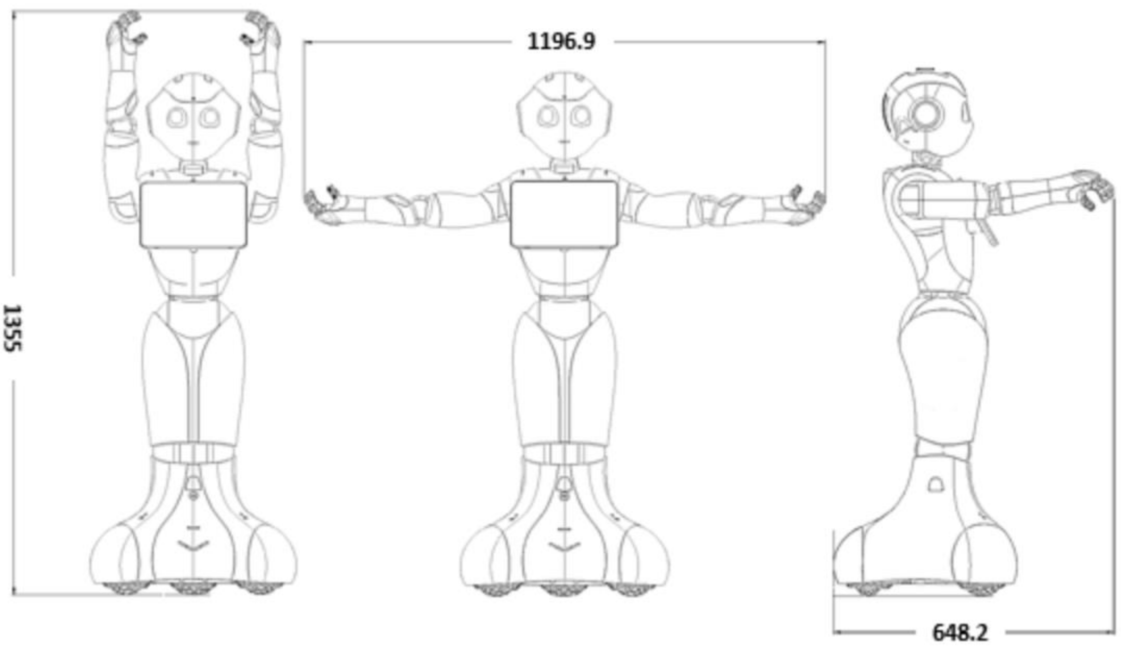


Figure 15. Pepper Dimensions [16]

1.4.1 NAOqi Python API

NAOqi is the main software to run projects on the robot, under OpenNAO distribution or computers via many software languages like Python, C++. The NAOqi Framework is the programming used to develop projects on NAO or PEEPER robots using all kinds of software languages through different platforms. NAOqi answers to common robotics need such as parallelism, resources, synchronization, events.

This framework supports all kinds of communication between different modules and parts like motion, vision, microphone, video, homogeneous programming, and homogeneous information sharing. The NAOqi framework can cross many platforms so that it is able to develop all kinds of projects on the robot with it on Windows, Linux or Mac. The NAOqi framework also supports many software languages through an identical API for both C++ and Python to benefit developers. The NAOqi framework still supports introspection, to know which functions are available in the different modules and where the functions are.

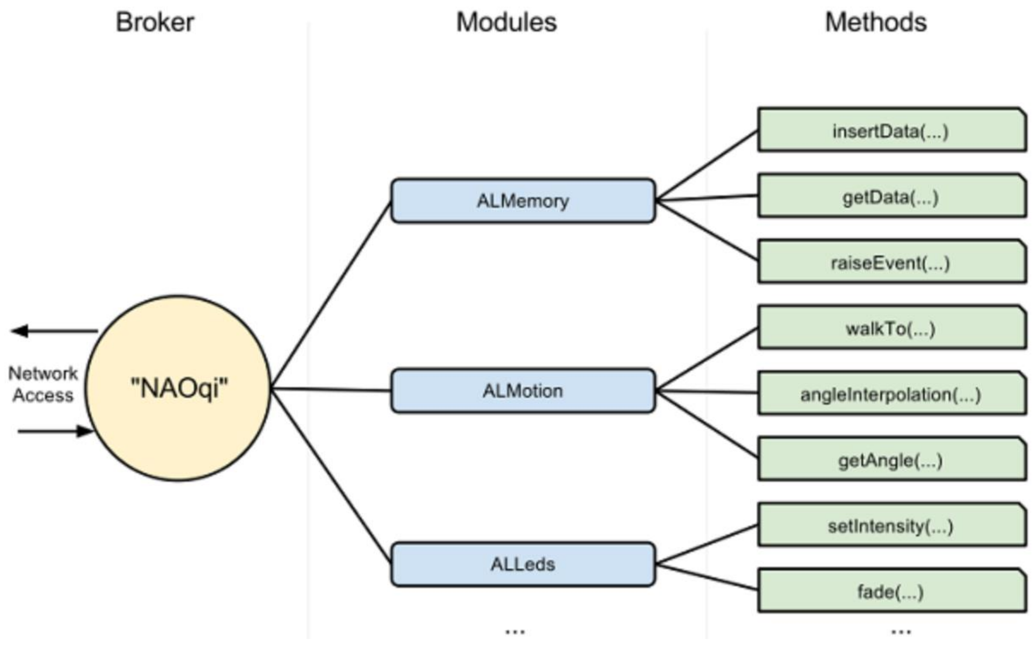
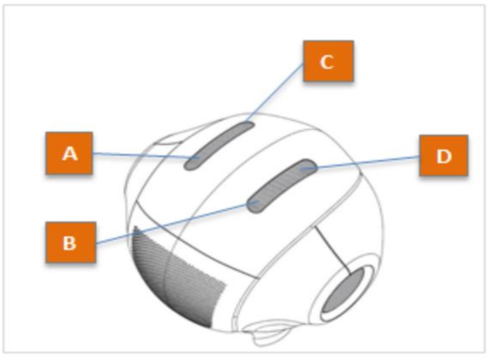


Figure 16. The NAOqi process [17]

1.4.2 ALTextToSpeech

Pepper has four microphones on the head, shown in Figure 17, and the type of Pepper's microphone is omnidirectional. These microphones' sensitivity is 250mV/Pa +/-3dB at 1kHz, while their frequency range is 100Hz to 10kHz (-10dB relative to 1kHz).



See the URDF files.

Part	Name
A	MicroRL_sensor
B	MicroRR_sensor
C	MicroFL_sensor
D	MicroFR_sensor

Figure 17. Pepper's Microphones [16]

The ALTextToSpeech module is one of the NAOqi APIs to let the robot to speak. The commands about ALTextToSpeech is sent to text-to-speech engine, and then robot machine will authorize the voice customization, then the result of the synthesis will be sent to the robot's loudspeakers to let robot speak the defined words.

1.4.3 ALFaceDetection

Three cameras are composed in Pepper's head, two 2D cameras on the top and the bottom and one 3D sensor in Pepper's eyes. The 2D cameras are two identical video cameras which are composed in robot's forehead, supporting a resolution up to 2560*1920 at 1 frames fps (per second) or 640*480 at 30 fps (per second), shown in Figure 18.

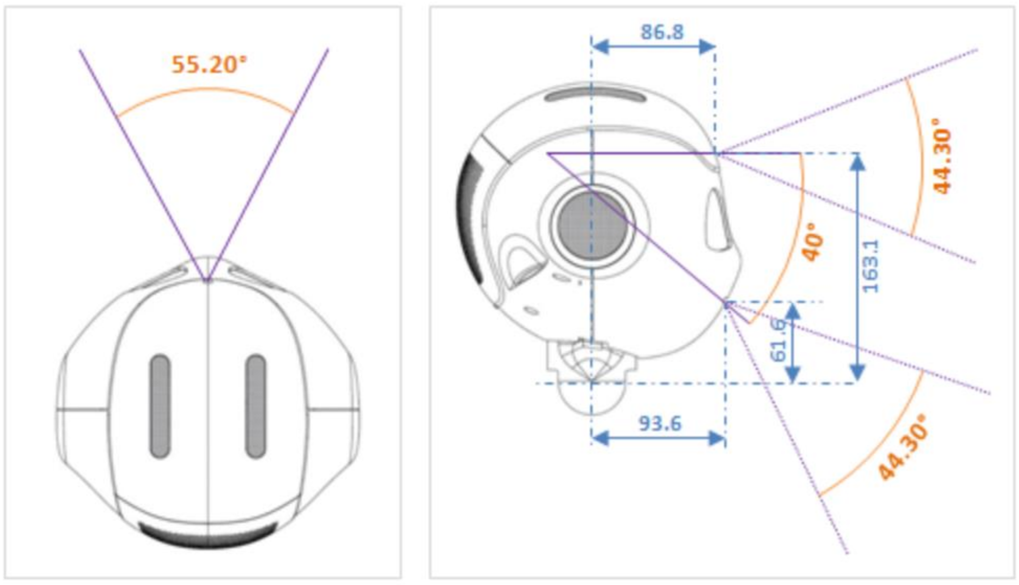


Figure 18. 2D Cameras in Pepper [16]

The ASUS Xtion 3D sensor is still located in Pepper’s forehead, supporting image resolution up to 320*240 at frames fps (per second), shown in Figure 19.

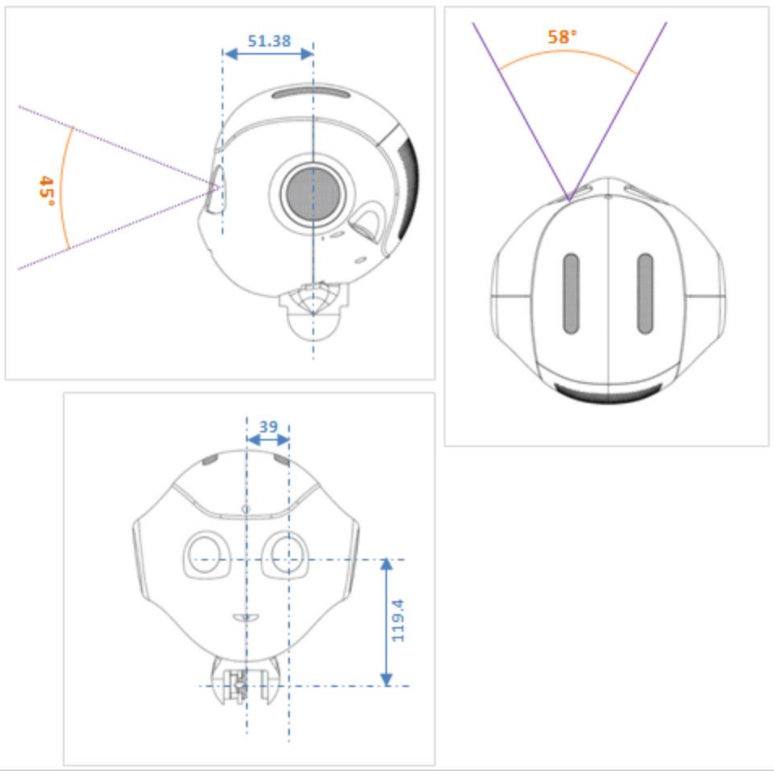


Figure 19. 3D Sensor in Pepper [16]

1.5 OpenCV-Python

OpenCV is a widely used open source toolkit in the field of computer vision, based on C++, crossing many platforms like Linux, Windows, MacOS, Android, and iOS, and supporting interfaces in languages such as Python, MATLAB. OpenCV has rich interfaces, excellent performance and commercially friendly licensing, both in academia and in the industry, which is very popular. OpenCV first originated from an Intel Corporation research project in 1998, when Gary Bradski, an engineer at Intel who worked in computer vision. When Guery Bradsky was visiting some universities and research groups, it was found that computer vision algorithms were used between students to implement internal code or libraries in their respective labs, so that new lab students could quickly get started based on the basic functions written by their predecessors. So OpenCV aims to provide a high-performance general library for scientific research and commercial applications for computer vision.

1.6 NumPy

NumPy (Numerical Python) is an extension library of the Python language that supports a large number of dimension groups and matrix operations, and also provides a large library of mathematical functions for array operations.

NumPy is a fast-running mathematical library, mainly used for array computing and having a powerful N-dimensional array, a broadcast functional function, some tools of consolidating C, C++ or Fortran code and the functions to create linear algebra, Fourier transform, random number and so on.

1.7 Matplotlib

Matplotlib is Python's most famous drawing library, which provides a complete set of command APIs similar to MATLAB and is ideal for interactive mapping. Among them, Matplotlib's Pyplot module is generally the most commonly used, which can make it easy for users to quickly draw two-dimensional charts.

2 FEASIBILITY STUDY

Feasibility Study refers to the research and evaluation of the projects to be carried out prior to the study of the project through a large number of market surveys, including market analysis, market research, target population surveys, technical surveys, operational guidelines and knowledge of the relevant laws.

2.1 Technical Feasibility Study

Today, the Python language has gradually become one of the mainstream languages of the editing program, and PyCharm is also the mainstream IDE for Python language programming. With the popularization of artificial intelligence, machine learning and deep learning have gradually attracted people’s attention. The development of robots is also one of the hottest topics today, and robots are used in a wide range of industries, with the primary aim of reducing unnecessary human resources and improving the efficiency of life and industries. Technically, through the platform of the PEPPER robot, the programming design of the Python language and the construction of the CNN framework in deep learning are applied to the medical system to analyze the patient’s condition and judge the patient’s mental state through emotional changes. TensorFlow is also a common package to build a convolutional neural network.
About the TensorFlow, there are four features of TensorFlow:

- Flexibility. TensorFlow is not a strict neural network library which means that any calculations can be used as long as they can be represented by a flow diagram.
- Portability. The underlying core codes of TensorFlow are C++, which can be run on many devices and provide distributed supports. This feature can quickly build a deep learning structure.
- Multilingual support. TensorFlow supports Python, C, C++, Java and Go.
- Efficiently. TensorFlow provides the supports for threads, queues and asynchronous operations, also supporting running on the CPU and GPU to realize the full potential of the hardware.

TensorFlow uses Graph to describe computer tasks, and the nodes in the diagram are called op. An op can accept 0 or more tensors as input or 0 or more tensors as output. Any Graph needs to be with the help of context Session to run. Start Graph through Session and distribute op in Graph to the CPU or GPU, and then with the help of Session to perform these op. These ops which are executed will return the resulting tensor. With the feed and fetch operations provided by Session, these ops are assigned values or get data. In the calculation process, the calculation state is maintained by variable.

Table 1. Related Concepts in TensorFlow

Type	Description	Use
Session	Session	The diagram must be executed in a context called a "session." The session distributes the OP of the diagram to calculations such as the CPU or GPU
Graph	Describe the calculation process	Must be started in Session
Tensor	Data	One of the data types that represent a multidimensional array
op	Operation	The nodes in the figure are called op, and an op gets 0 or more Tensor, then performs calculations, and then produces 0 or more Tensor
Variable	Variable	One of the data types that can be changed during operation to maintain state
feed	Assign a value	Assign a value to the tensor of op
fetch	Take a value	Take a value from the tensor of op
Constant	Constant	One of the data types, immutable

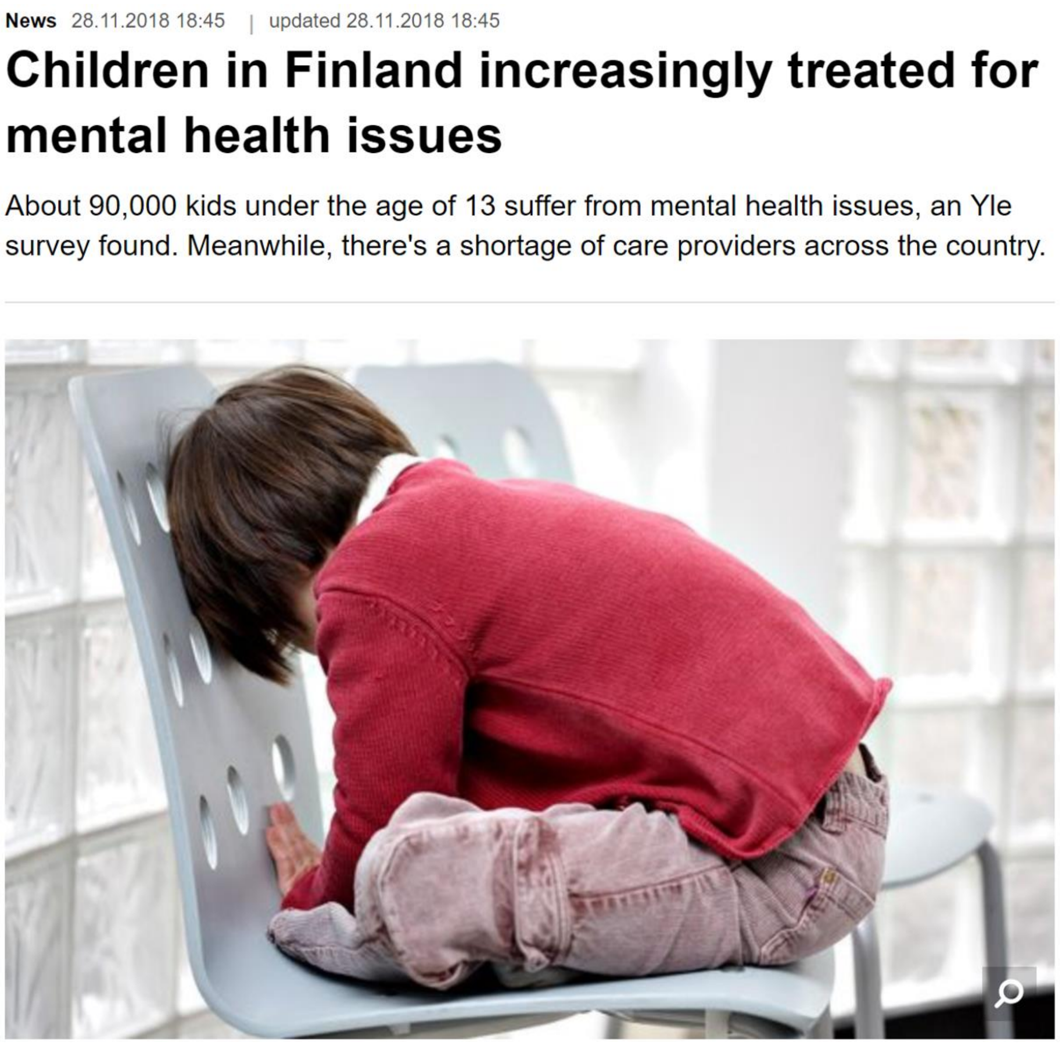
As for Keras, it is the simplified interface to TensorFlow. The guiding principles of Keras:

- User friendliness. Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.
- Modularity. A model is understood as a sequence or a graph of standalone, fully configurable modules that can be plugged together with as few restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, and regularization schemes are all standalone modules that you can combine to create new models.

- Easy extensibility. New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.
- Work with Python. No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility. [13]

2.2 Economic Feasibility Study

With the improvement of economic development, people’s demand for medical care is becoming more and more popular, the limited resources of medical institutions and people’s demand for medical care began to gradually create a gap. According to the report, shown in Figure 20, there is an increasing number of children in Finland who need treatment for mental health problems, but analytical support for the mental state has not increased synchronously. At this time, the use of an artificial intelligence robot instead of manpower to carry out people’s emotional changes to make judgment and recognition is particularly important. The extensive use of this design can alleviate a part of the medical institutions of human resources, so economic is also feasible.



News 28.11.2018 18:45 | updated 28.11.2018 18:45

Children in Finland increasingly treated for mental health issues

About 90,000 kids under the age of 13 suffer from mental health issues, an Yle survey found. Meanwhile, there's a shortage of care providers across the country.

Figure 20. Finland Mental Health Report [18]

2.3 Operational Feasibility Study

This design uses the humanoid robot PEPPER as an analysis of the mental condition of patients or children. It is done through face recognition of the patient and a series of captures of the facial expression, then the results are analyzed and summarized to determine the patient’s current mental state and visualize the results. Therefore, it is also feasible to operate.

2.4 Legal Feasibility Study

There is no link in this design that violates the provisions of the national law, so this design is also feasible in law.

3 SOFTWARE REQUIREMENT ANALYSIS

Software requirement analysis is a summary of a series of design analyses such as this project’s functional analysis, processing of data analysis, system flow charts and so on.

3.1 Functional Analysis

In this project,

- the robot first locates the patient’s face and captures 25 pictures of the patient’s changing facial expression as “.jpg” type,
- then data of the captured image goes into the native computer.
- CNN is used to build 4 layers of convolution and 2 full-connected layers, Kaggle’s “fer2013.csv” is used as facial emotional dataset and trained through the CNN structure, as a result of this, a file of “.h5” type was created named “model_4layer_2_2_pool” which contains all the special features of the emotional dataset.
- After receiving the pictures from a robot, these pictures will be trained and tested through the CNN structure and will be analyzed and recognized via the pre-trained file.
- Finally, all the analysis of the results will be summarised and visualize of the display as a bar chart in the values of angry, disgust, fear, happy, sad, surprise and neutral.

In this project, TensorFlow was used as backend and the Keras was used as the interface to call the functions of TensorFlow to build the CNN structure.

3.2 System Flowchart

The patient faces the robot with the robot’s voice prompts, and the robot begins a continuous shot of the facial expression after capturing the human face. When the local computer accepts the pictures, pictures will be transferred into CNN for training and recognition, and then visualize the results, shown in Figure 21.

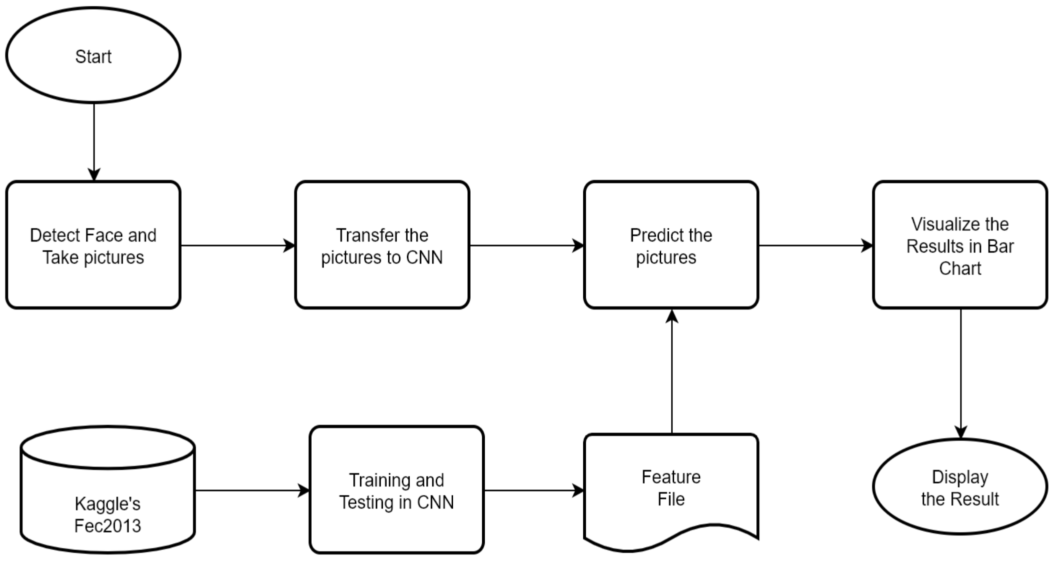

Start
Detect Face and Take pictures
Transfer the pictures to CNN
Predict the pictures
Visualize the Results in Bar Chart
Kaggle's Fec2013
Training and Testing in CNN
Feature File
Display the Result

Figure 21. System Flowchart

3.3 Data Flowchart

3.3.1 Input Data Flowchart

When the patient's face is identified, a series of the patient's emotional pictures are captured, a total of 25 pictures as ".jpg" type. Then a connection with the local computer is built and these pictures are transferred, as shown in Figure 22.

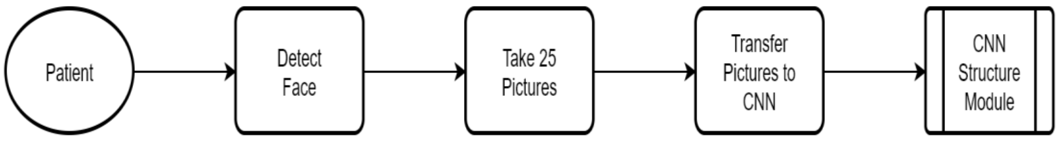


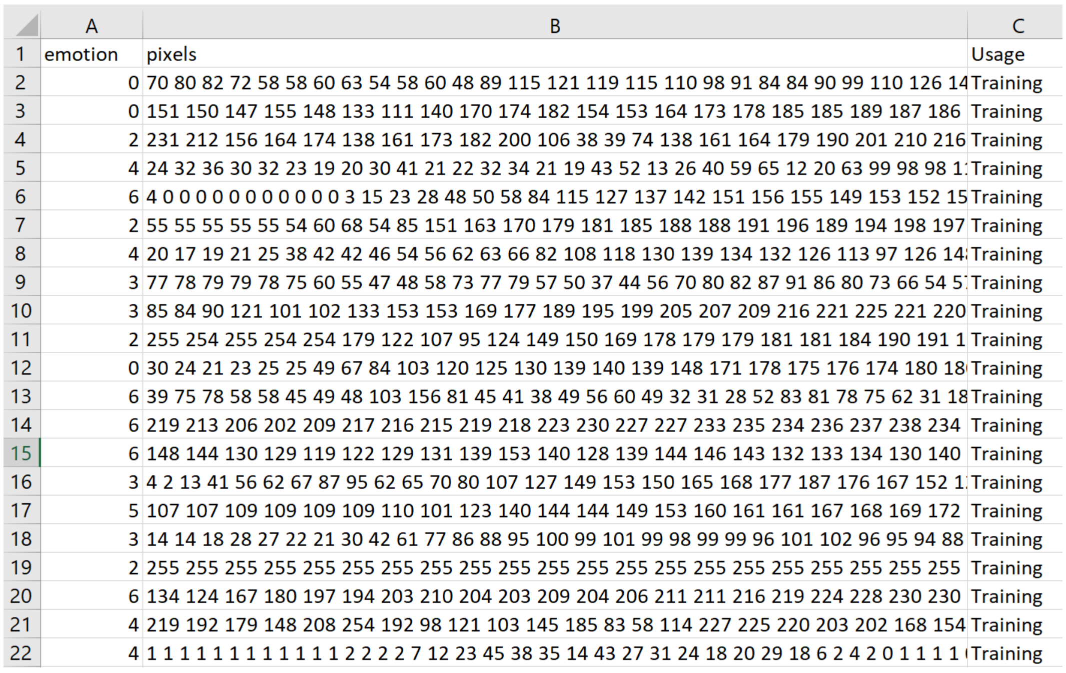
Figure 22. Input Date Flowchart

3.3.2 Dataset Training Flowchart

In this project, the facial emotional dataset is Kaggle's Fec2013, which is compressed at a size of 92 M and the uncompressed version size is 295 M. The dataset contains 28,000 training photos and 30,000 test photos, shown as Figure 23. Each photo is stored as 48 x 48 pixels.

This dataset contains image pixels (48 x 48 = 2,304 values), facial expressions in each image, and usage types (used as training or tests). The data file is mainly "Fec2013.csv", a total of two columns: emotion, pixels, and usage. There are 7 tags: 0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral. The pixels column is 48 x 48 pixels, shown in Figure 24.

Figure 23. Examples Picture of Fer2013 [19]



	A	B	C
1	emotion	pixels	Usage
2	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121 119 115 110 98 91 84 84 90 99 110 126 14	Training
3	0	151 150 147 155 148 133 111 140 170 174 182 154 153 164 173 178 185 185 189 187 186	Training
4	2	231 212 156 164 174 138 161 173 182 200 106 38 39 74 138 161 164 179 190 201 210 216	Training
5	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 19 43 52 13 26 40 59 65 12 20 63 99 98 98 1	Training
6	6	4 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84 115 127 137 142 151 156 155 149 153 152 15	Training
7	2	55 55 55 55 55 54 60 68 54 85 151 163 170 179 181 185 188 188 191 196 189 194 198 197	Training
8	4	20 17 19 21 25 38 42 42 46 54 56 62 63 66 82 108 118 130 139 134 132 126 113 97 126 14	Training
9	3	77 78 79 79 78 75 60 55 47 48 58 73 77 79 57 50 37 44 56 70 80 82 87 91 86 80 73 66 54 5	Training
10	3	85 84 90 121 101 102 133 153 153 169 177 189 195 199 205 207 209 216 221 225 221 220	Training
11	2	255 254 255 254 254 179 122 107 95 124 149 150 169 178 179 179 181 181 184 190 191 1	Training
12	0	30 24 21 23 25 25 49 67 84 103 120 125 130 139 140 139 148 171 178 175 176 174 180 18	Training
13	6	39 75 78 58 58 45 49 48 103 156 81 45 41 38 49 56 60 49 32 31 28 52 83 81 78 75 62 31 18	Training
14	6	219 213 206 202 209 217 216 215 219 218 223 230 227 227 233 235 234 236 237 238 234	Training
15	6	148 144 130 129 119 122 129 131 139 153 140 128 139 144 146 143 132 133 134 130 140	Training
16	3	4 2 13 41 56 62 67 87 95 62 65 70 80 107 127 149 153 150 165 168 177 187 176 167 152 1	Training
17	5	107 107 109 109 109 109 110 101 123 140 144 144 149 153 160 161 161 167 168 169 172	Training
18	3	14 14 18 28 27 22 21 30 42 61 77 86 88 95 100 99 101 99 98 99 99 96 101 102 96 95 94 88	Training
19	2	255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255	Training
20	6	134 124 167 180 197 194 203 210 204 203 209 204 206 211 211 216 219 224 228 230 230	Training
21	4	219 192 179 148 208 254 192 98 121 103 145 185 83 58 114 227 225 220 203 202 168 154	Training
22	4	1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 7 12 23 45 38 35 14 43 27 31 24 18 20 29 18 6 2 4 2 0 1 1 1 1	Training

Figure 24. Fec2013.csv

"Fec2013.csv" will be transferred into CNN structure to training and testing, and then output a feature file in ".h5" type which contains all the facial emotional characteristics, named "model_4layer_2_2_pool". The flowchart is shown in Figure 25.

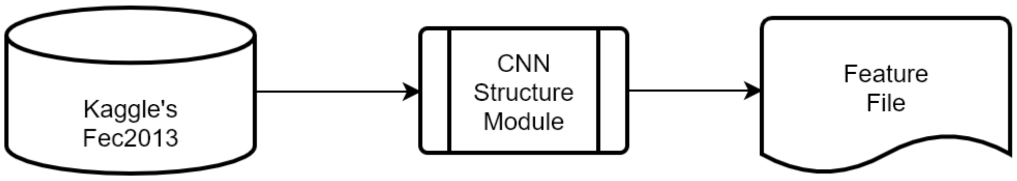


Figure 25. Dataset Training Flowchart

3.3.3 CNN Structure Module Flowchart

In this project, the CNN structure is like "input -> conv 1 -> pool 1 -> conv 2 -> pool 2 -> conv 3 -> pool 3 -> conv 4 -> pool 4 -> fc 1 -> fc 2 -> out", which contains four convolutional layers and two full-connected layers. The structure is shown in Figure 26.

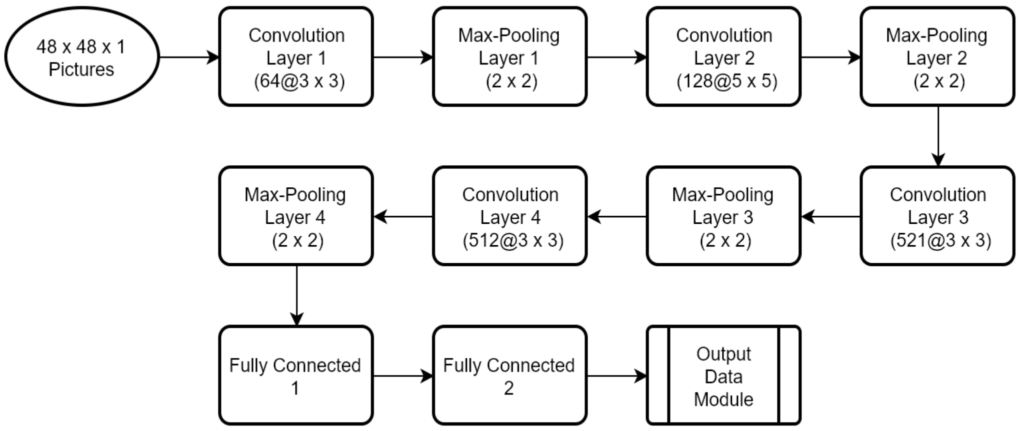

48 x 48 x 1 Pictures
Convolution Layer 1 (64@3 x 3)
Max-Pooling Layer 1 (2 x 2)
Convolution Layer 2 (128@5 x 5)
Max-Pooling Layer 2 (2 x 2)
Convolution Layer 3 (521@3 x 3)
Max-Pooling Layer 3 (2 x 2)
Convolution Layer 4 (512@3 x 3)
Max-Pooling Layer 4 (2 x 2)
Fully Connected 1
Fully Connected 2
Output Data Module

Figure 26. CNN Structure

3.3.4 Output Data Flowchart

After all the images captured by the robot having been identified by deep learning, all the results will be aggregated to the average, and finally, the aggregated results will be visualized in the form of a bar chart, shown in Figure 27.

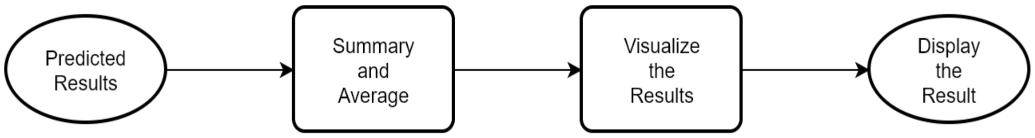


Figure 27. Output Data Flowchart

4 DETAILED DESIGN

The detailed design mainly contains some detailed description of the modules of this project, including the detailed operation process, the structure of each module and the core codes. This project is divided into three modules: CNN structure module, training module and predict module.

4.1 CNN Structure Module

This module mainly describes the detailed structure of the CNN framework in this project and the data analysis process. This CNN structure is built by Keras with TensorFlow backend, which contains tour convolution layers and two full-connected layers. The detailed structure is shown as Figure 28 which is written by python, illustrated in Figure 29, Figure 30 and Figure 31.

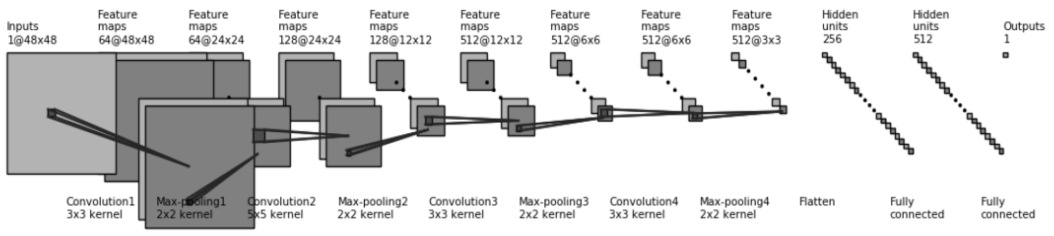
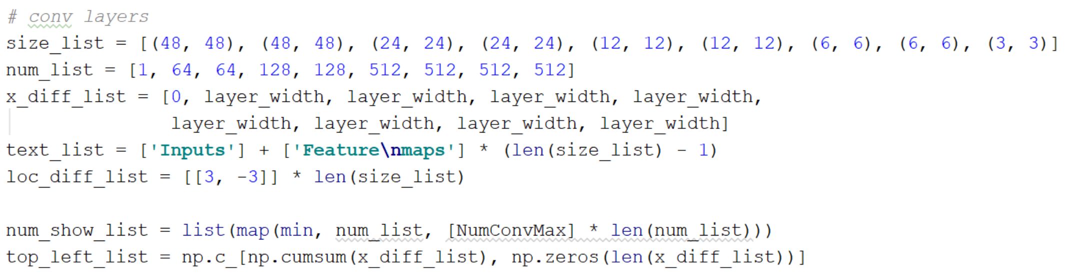


Figure 28. CNN Detailed Structure



```
# conv layers
size_list = [(48, 48), (48, 48), (24, 24), (24, 24), (12, 12), (12, 12), (6, 6), (6, 6), (3, 3)]
num_list = [1, 64, 64, 128, 128, 512, 512, 512, 512]
x_diff_list = [0, layer_width, layer_width, layer_width, layer_width,
               layer_width, layer_width, layer_width, layer_width]
text_list = ['Inputs'] + ['Feature\nmaps'] * (len(size_list) - 1)
loc_diff_list = [[3, -3]] * len(size_list)

num_show_list = list(map(min, num_list, [NumConvMax] * len(num_list)))
top_left_list = np.c_[np.cumsum(x_diff_list), np.zeros(len(x_diff_list))]
```

Figure 29. Example Codes for Displaying the CNN Structure about convolution layers

```
############################
# in between layers
start_ratio_list = [[0.4, 0.5], [0.4, 0.8], [0.4, 0.5], [0.4, 0.8],
                    [0.4, 0.5], [0.4, 0.8], [0.4, 0.5], [0.4, 0.8]]
end_ratio_list = [[0.4, 0.5], [0.4, 0.8], [0.4, 0.5], [0.4, 0.8],
                  [0.4, 0.5], [0.4, 0.8], [0.4, 0.5], [0.4, 0.8]]
patch_size_list = [(3, 3), (2, 2), (5, 5), (2, 2), (3, 3), (2, 2), (3, 3), (2, 2)]
ind_bgn_list = range(len(patch_size_list))
text_list = ['Convolution1', 'Max-pooling1', 'Convolution2', 'Max-pooling2',
             'Convolution3', 'Max-pooling3', 'Convolution4', 'Max-pooling4']

for ind in range(len(patch_size_list)):
    add_mapping(
        patches, colors, start_ratio_list[ind], end_ratio_list[ind],
        patch_size_list[ind], ind,
        top_left_list, loc_diff_list, num_show_list, size_list)
    label(top_left_list[ind], text_list[ind] + '\n{}x{} kernel'.format(
        patch_size_list[ind][0], patch_size_list[ind][1]), xy_off=[26, -65]
    )
```

Figure 30. Example Codes for Displaying the CNN Structure about convolution and pooling layers

```
# fully connected layers
size_list = [(fc_unit_size, fc_unit_size)] * 3
num_list = [256, 512, 1]
num_show_list = list(map(min, num_list, [NumFcMax] * len(num_list)))
x_diff_list = [sum(x_diff_list) + layer_width, layer_width, layer_width]
top_left_list = np.c_[np.cumsum(x_diff_list), np.zeros(len(x_diff_list))]
loc_diff_list = [[fc_unit_size, -fc_unit_size]] * len(top_left_list)
text_list = ['Hidden\nunits'] * (len(size_list) - 1) + ['Outputs']

for ind in range(len(size_list)):
    if flag_omit:
        add_layer_with_omission(patches, colors, size=size_list[ind],
                                num=num_list[ind],
                                num_max=NumFcMax,
                                num_dots=NumDots,
                                top_left=top_left_list[ind],
                                loc_diff=loc_diff_list[ind])
    else:
        add_layer(patches, colors, size=size_list[ind],
                  num=num_show_list[ind],
                  top_left=top_left_list[ind],
                  loc_diff=loc_diff_list[ind])
    label(top_left_list[ind], text_list[ind] + '\n{}'.format(
        num_list[ind]))

text_list = ['Flatten\n', 'Fully\nconnected', 'Fully\nconnected']

for ind in range(len(size_list)):
    label(top_left_list[ind], text_list[ind], xy_off=[-10, -65])
```

Figure 31. Example Codes for Displaying the CNN Structure about fully-connected layer

In this CNN structure, data will be transferred into CNN and then trained and tested in the CNN structure, shown in Figure 32.

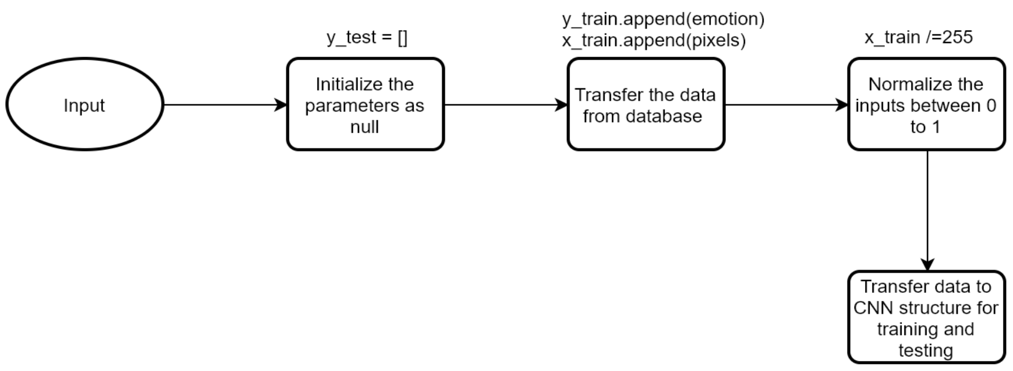
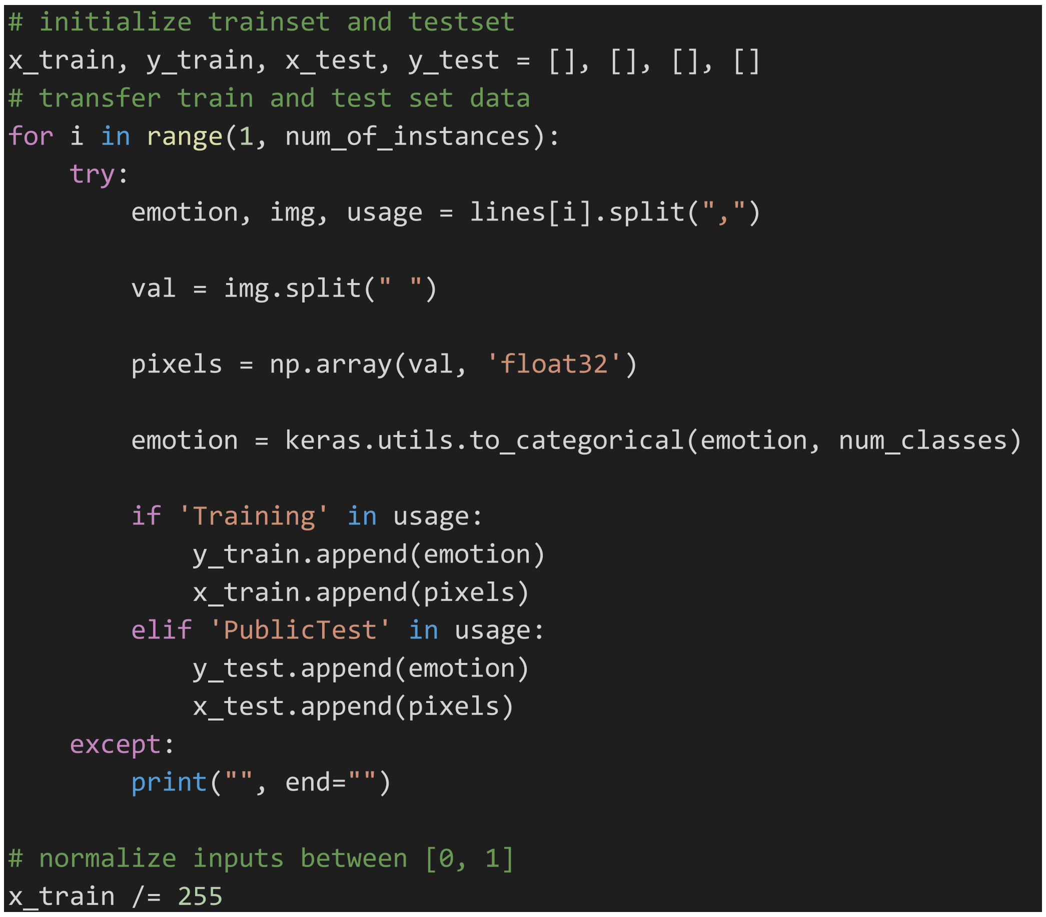

Input
y_test = []
Initialize the parameters as null
y_train.append(emotion)
x_train.append(pixels)
Transfer the data from database
x_train /=255
Normalize the inputs between 0 to 1
Transfer data to CNN structure for training and testing

Figure 32. Data Processing Flowchart

- it will firstly initialize the parameters and transfer the training and testing dataset from database and then normalize the inputs between 0 to 1.



```
# initialize trainset and testset
x_train, y_train, x_test, y_test = [], [], [], []
# transfer train and test set data
for i in range(1, num_of_instances):
    try:
        emotion, img, usage = lines[i].split(",")

        val = img.split(" ")

        pixels = np.array(val, 'float32')

        emotion = keras.utils.to_categorical(emotion, num_classes)

        if 'Training' in usage:
            y_train.append(emotion)
            x_train.append(pixels)
        elif 'PublicTest' in usage:
            y_test.append(emotion)
            x_test.append(pixels)
    except:
        print("", end="")

# normalize inputs between [0, 1]
x_train /= 255
```

```
x_test /= 255
```

- Then the data will be transfer into CNN structure, which means to first initialize the CNN and then begin to build four convolution and pooling layers and two fully connected layers.

```
#Main CNN model with four Convolution layer & two fully connected 
layer
def baseline_model():
    # Initialising the CNN
    model = Sequential()
```

In this project,

- the first convolution layer is built by the two-dimensional convolution layer Conv2D which contains 64 filters. Each filter's kernel_size is 3 x 3 and the border_mode is "same".

```
    # 1 - Convolution
    model.add(Conv2D(64,(3,3), border_mode='same', 
input_shape=(48, 48,1)))
```

  the activation layer of CNN structure is using the "relu" predefined activation function which means the activation layer exerts an activation function on the output of a layer.

```
    model.add(Activation('relu'))
```

  In order to prevent over-fitting, Dropout function is used to randomly disconnects a certain percentage (p, in this project $p = 0.25$) of the input neuron connection each time the parameter is updated during training.

```
    model.add(Dropout(0.25))
```

  The first pooling layer is built by the two-dimensional Man-Pooling layer MaxPooling2D, which the pool_size is 2 x 2. And the Pooling's role in CNN is:

  i. Invariance. This invariance includes translation, rotation, and scale.
  ii. Retain the main features while reducing the parameters (descending dimension, an effect similar to PCA) and computation, preventing over-fitting and improving the generalization ability of the mode.

```
    model.add(MaxPooling2D(pool_size=(2, 2)))
```

- The second convolution layer is built by the two-dimensional convolution layer Conv2D which contains 128 filters. Each filter's kernel_size is 5 x 5 and the border_mode is "same". "relu" activation function is used and the parameter of Dropout is 0.25. The second pooling layer is built by the two-dimensional Man-Pooling layer MaxPooling2D, which the pool_size is 2 x 2.

```
    # 2nd Convolution layer
    model.add(Conv2D(128,(5,5), border_mode='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
```

- The third convolution layer is built by the two-dimensional convolution layer Conv2D which contains 512 filters. Each filter's kernel_size is 3 x 3 and the border_mode is "same". "relu" activation function is used and the parameter of Dropout is 0.25. The second pooling layer is built by the two-dimensional Man-Pooling layer MaxPooling2D, which the pool_size is 2 x 2.

```
    # 3rd Convolution layer
    model.add(Conv2D(512,(3,3), border_mode='same'))
    model.add(BatchNormalization())
    model.add(Activation('relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
```

- The fourth convolution and pooling layer are the same as the third layer.
- Flatten ()'s input data is a bunch of feature diagrams, and the network after it accepts a one-dimensional feature, so the role of Flatten is to "squash" its input data into 1D form so that it can be successfully entered into the fully connected layer.

```
    # Flattening
    model.add(Flatten())
```

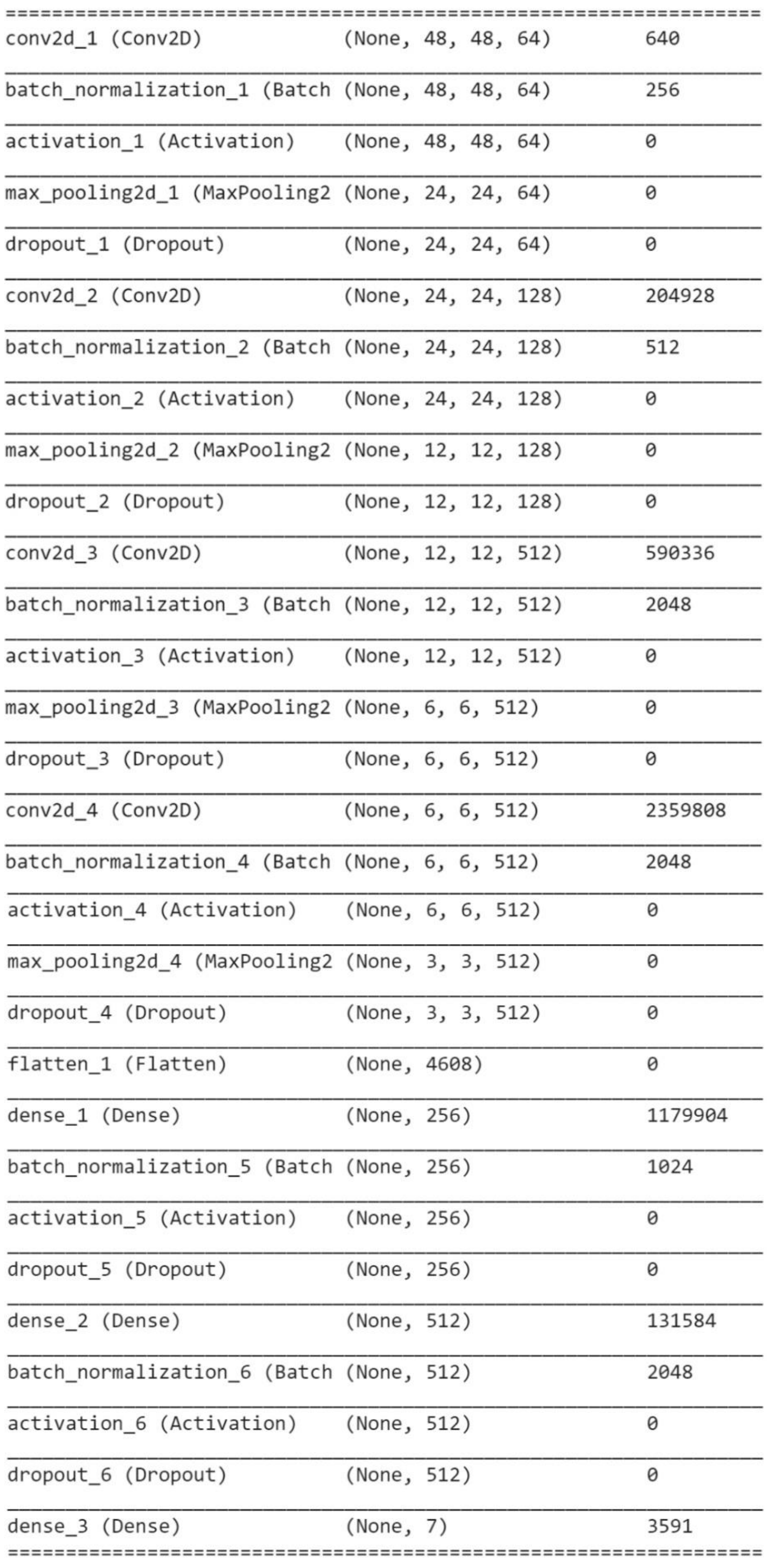
- The first fully connected layer. The fully connected layer in the Keras is called “Dense”. The connection here is very "dense" and the number of connections is very large. One of the required parameters of the dense layer is the number of neurons in the current layer, here are 256 neurons. The “relu” activation function is used and the parameter of Dropout is 0.25.

```
# Fully connected layer 1st layer
model.add(Dense(256))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.25))
```

- The second fully connected layer was built. The required parameters of the dense layer, the number of neurons in the current layer, here are 512 neurons. “relu” activation function is used and the parameter of Dropout is 0.25.
- The classification layer is to map the feature to the final category and then map it to the category probability through a “sigmoid” activation function. The 7 classifications are defined here, so the number of neurons in the final Dense layer is 7.

```
model.add(Dense(num_class, activation='sigmoid'))
```

After training the data, the CNN structure will be printed out and showed as Figure33.



```
=================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 64)        640
_________________________________________________________________
batch_normalization_1 (Batch (None, 48, 48, 64)        256
_________________________________________________________________
activation_1 (Activation)    (None, 48, 48, 64)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 24, 24, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 24, 24, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 128)       204928
_________________________________________________________________
batch_normalization_2 (Batch (None, 24, 24, 128)       512
_________________________________________________________________
activation_2 (Activation)    (None, 24, 24, 128)       0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 512)       590336
_________________________________________________________________
batch_normalization_3 (Batch (None, 12, 12, 512)       2048
_________________________________________________________________
activation_3 (Activation)    (None, 12, 12, 512)       0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 512)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 512)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 6, 6, 512)         2359808
_________________________________________________________________
batch_normalization_4 (Batch (None, 6, 6, 512)         2048
_________________________________________________________________
activation_4 (Activation)    (None, 6, 6, 512)         0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 3, 3, 512)         0
_________________________________________________________________
dropout_4 (Dropout)          (None, 3, 3, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               1179904
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024
_________________________________________________________________
activation_5 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               131584
_________________________________________________________________
batch_normalization_6 (Batch (None, 512)               2048
_________________________________________________________________
activation_6 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 7)                 3591
=================================================================
```

Figure 33. CNN Structure after Training

4.2 Training Module

This module is used to illustrate the process about training the data, shown in Figure 34.

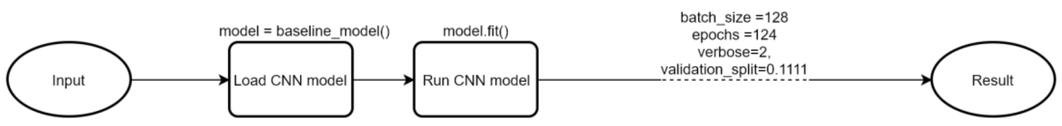


Figure 34. Training Module Flowchart

The compiled CNN model begins to prepare training data. There are two ways to train models in Keras, *model.fit()* and *model.fit_generator()*, and in this project, *model.fit()* function is selected for data training. The process of the *model.fit()* function can be simply understood as the process of determining the weight of connections between neurons through test data.

The test data is divided into two parts, the matrix type of input data *x*, and the corresponding array type of output *y*. Neural network training usually uses a reverse propagation (Backpropagation) algorithm, so the parameters that need to be specified are the training cycle *epochs* and the amount of data calculated each time *batch_size*. When the training is complete, the *history* will be used to save the relevant description after the model training.

The complete *model.fit()* function is:

```
fit( x, y, batch_size, epochs, verbose, callbacks=None,
validation_split=0.0, validation_data=None, shuffle=True,
class_weight=None, sample_weight=None, initial_epoch=0)
```

In this project, parameters of *x* = *X_train, y* = *y_train, batch_size* = *batch_size, epochs* = *epochs, verbose* and *validation_split* are used to define the testing function.

- *x*: Input data. If the model has only one input data, the type of *x* is numpy array, and if the model has more than one input data, the type of *x* will be a list which element is the numpy array to each input data.

```
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.1, random_state=0)
```

- *y*: Label. The type of *y* is numpy array.

```
y_train = (np.arange(num_class) == y_train[:,
None]).astype(np.float32)
```

- *batch_size*: The type is an integer. This parameter is the number of samples that each batch contains when a gradient drop is specified. During training, a *batch* sample is calculated as a gradient drop, allowing the objective function to be optimized one step at a time.

```
batch_size = 128
```

- *epochs*: The type is an integer. This parameter is the *epoch* value that represents the end of the training. The training will stop when the *epoch* value is reached, and when no *initial_epoch* is set, it is the total number of rounds trained, otherwise, the total number of rounds of training is *epochs* - *inital_epoch*.

```
epochs = 124
```

- *verbose*: This parameter is the control log information. *0* means that the log information is not output via the standard output stream, and *1* is represented as the output progress bar record, and *2* is represented as the output row record for each epoch.
- *validation_split*: This parameter type is a floating-point number between 0~1 and is used to specify a certain percentage of the training set as a validation set. Validation sets will not participate in training and test the indicators of the model after each *epoch* end, such as loss functions, accuracy, and so on. *validation_split* is divided before *shuffle*, so if the data itself is ordered, it needs to be manually disrupted before specifying *validation_split*, otherwise, the validation set sample may appear uneven.

```
    model.fit(X_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              verbose=2,
                validation_split=0.1111)
    model_json = model.to_json()
    with open("model_4layer_2_2_pool.json", "w") as json_file:
        json_file.write(model_json)
    # serialize weights to HDF5
    model.save_weights("model_4layer_2_2_pool.h5")
```

4.3 Predict Model

In this project, the model will predict the probability values for 7 labels for a test image: anger, disgust, fear, happy, sad, surprise and neutral, shown in Figure 35.

```
label_map = ['Anger', 'Disgust', 'Fear', 'Happy', 'Sad',
'Surprise', 'Neutral']
```

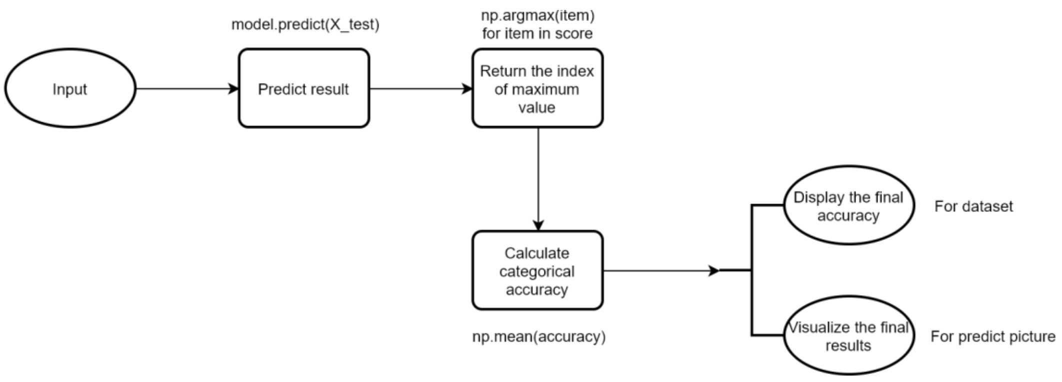
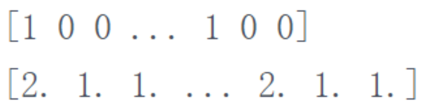


Figure 35. Predict Module Flowchart

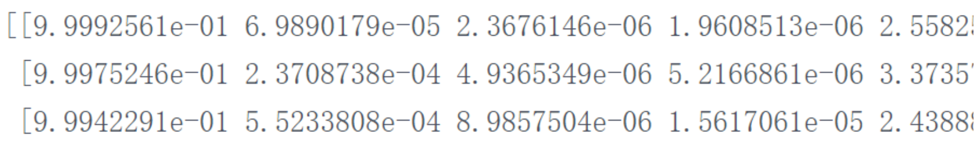
When making deep network predictions in Keras, there are two common predictive functions named *model.predic_classes(),* which predicts a category shown as Figure 35 and the printed value is the category number and can only be used for sequence models not for functional models, and *model.predict()*, which predicts some values and the output are numpy array of 7 encoded values in real number, shown as Figure 37. In this project, *model.predict()* is used to predict the result.

```
score = model.predict(X_test)
```



```
[1 0 0 ... 1 0 0]
[2. 1. 1. ... 2. 1. 1.]
```

Figure 36. Example Result of model.predic_classes()

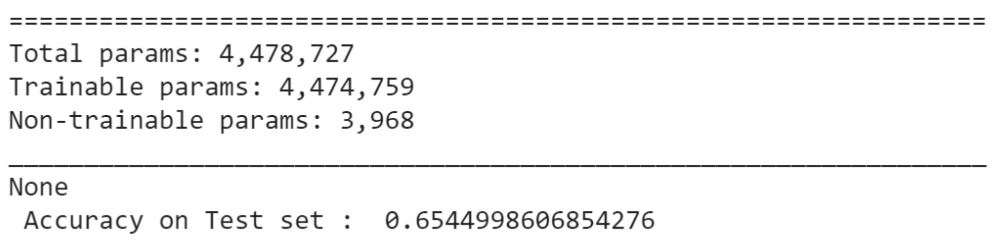


```
[[9.9992561e-01 6.9890179e-05 2.3676146e-06 1.9608513e-06 2.5582
 [9.9975246e-01 2.3708738e-04 4.9365349e-06 5.2166861e-06 3.3735
 [9.9942291e-01 5.5233808e-04 8.9857504e-06 1.5617061e-05 2.4388
```

Figure 37. Example Result of model.predict()

- After predicting the data, the final categorical accuracy will be calculated by taking labels of having highest probability via the function *np.argmax()*, which returns the index of the maximum value along the axis and *np.mean()*, and the result is about 0.654, which is shown in Figure 38.

```
New_X = [ np.argmax(item) for item in score ]
y_test2 = [ np.argmax(item) for item in y_test]
accuracy = [ (x==y) for x,y in zip(new_X,y_test2) ]
print(" Accuracy on Test set : " , np.mean(accuracy))
```



```
=================================================================
Total params: 4,478,727
Trainable params: 4,474,759
Non-trainable params: 3,968
_________________________________________________________________
None
 Accuracy on Test set :  0.6544998606854276
```

Figure 38. The Result of Final Accuracy

- After predicting the pictures taken from Pepper robot, the result will be transferred to the visualizing result.

```
Custom = model.predict(x)
emotion_analysis(custom[0],os.path.basename(imgPath))
```

5 IMPLEMENTATION

This part will illustrate the implementation of every detailed module in this project, including detecting and taking pictures, transferring the pictures, recognizing a face, predicting the pictures and visualizing the result, shown in Figure 39.

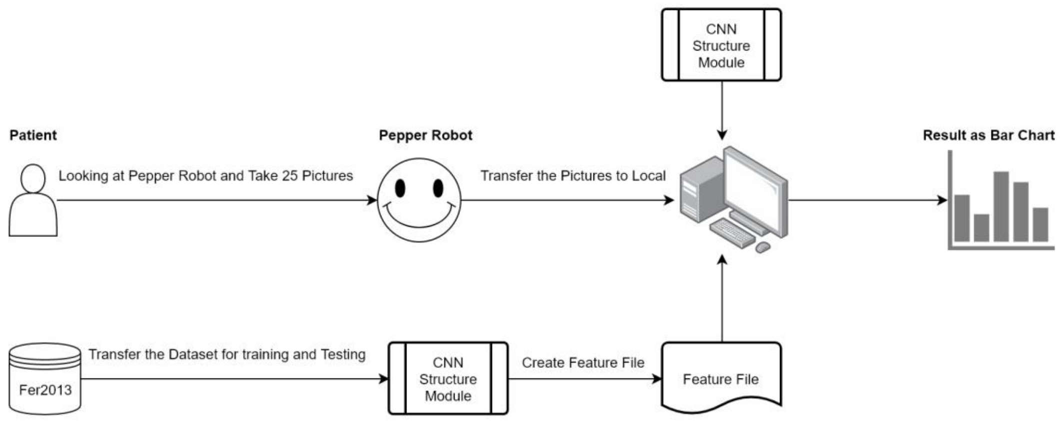

CNN Structure Module
Patient
Pepper Robot
Result as Bar Chart
Looking at Pepper Robot and Take 25 Pictures
Transfer the Pictures to Local
Fer2013
Transfer the Dataset for training and Testing
CNN Structure Module
Create Feature File
Feature File

Figure 39. Project Implementation Flowchart

5.1 Detecting and Taking Pictures Implementation

The first thing is to make the connection between the Pepper robot and the computer via the robot's ID and port, and then create a proxy on the text-to-speech module, named ALTextToSpeech, to give some information to patients to look at the robot. The connection between the computer and the Pepper Robot is Naoqi APIs. About Naoqi, a real-time application can only be a separate executable file or robot tree, process tree, module tree. No matter what you choose, the calling method is always the same. Use IP addresses and ports to connect the executable file to another robot, and all API methods in other executables are available in the same way as the local method. NAOqi is a choice between a quick direct call (LPC) and a remote call (RPC).

The NAOqi executable file running on the robot is a broker. After it starts, it loads a file named AutoLoad. Each library contains one or more modules that use proxies to invoke their methods, as shown in Figure 16. Proxies are used to provide a lookup service so that any module in the tree or on the network can find any method that has been advertised. The loading module forms a method tree attached to the module, as well as a module tree attached to the agent.

```
#create a proxy on the text-to-speech module
tts=ALProxy("ALTextToSpeech",robot_ID,PORT)

#modify the voice's speed
tts.setParameter("speed",200)
tts.resetSpeed()
tts.say("Hello, please look at me.")
```

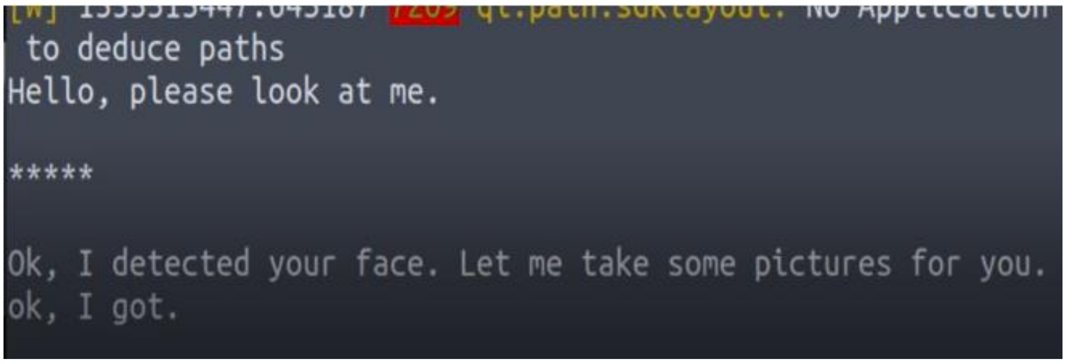
Then a proxy is created to let the robot detect patient's face, named ALFaceDetection, and subscribe to the ALFaceDetection proxy, which means that the module will write in ALMemory with the given period, finally make a valid output. After detecting the patient's face, ALPhotoCapture will be created to take 25 pictures.

```
#create a proxy to ALFaceDetection
faceProxy=ALProxy("ALFaceDetection",robot_ID, PORT)
```

```
# Subscribe to the ALFaceDetection proxy
period = 500
faceProxy.subscribe("Test_Face", period, 0.0 )
# Create a proxy to ALMemory
memoryProxy = ALProxy("ALMemory", robot_ID, PORT)
```

```
#crease a proxy to ALPhotoCapture
photoCaptureProxy = ALProxy("ALPhotoCapture", robot_ID, PORT)
photoCaptureProxy.setResolution(2)
photoCaptureProxy.setPictureFormat("jpg")
photoCaptureProxy.takePictures(25,"/home/nao/recordings/cameras/",
"image")
```

The implementation picture is shown in Figure 40.



```
[W] 1553513447.045187 7209 qt.path.sdklayout: No Application
 to deduce paths
Hello, please look at me.

*****

Ok, I detected your face. Let me take some pictures for you.
ok, I got.
```

Figure 40. Detecting and Taking Pictures Implementation

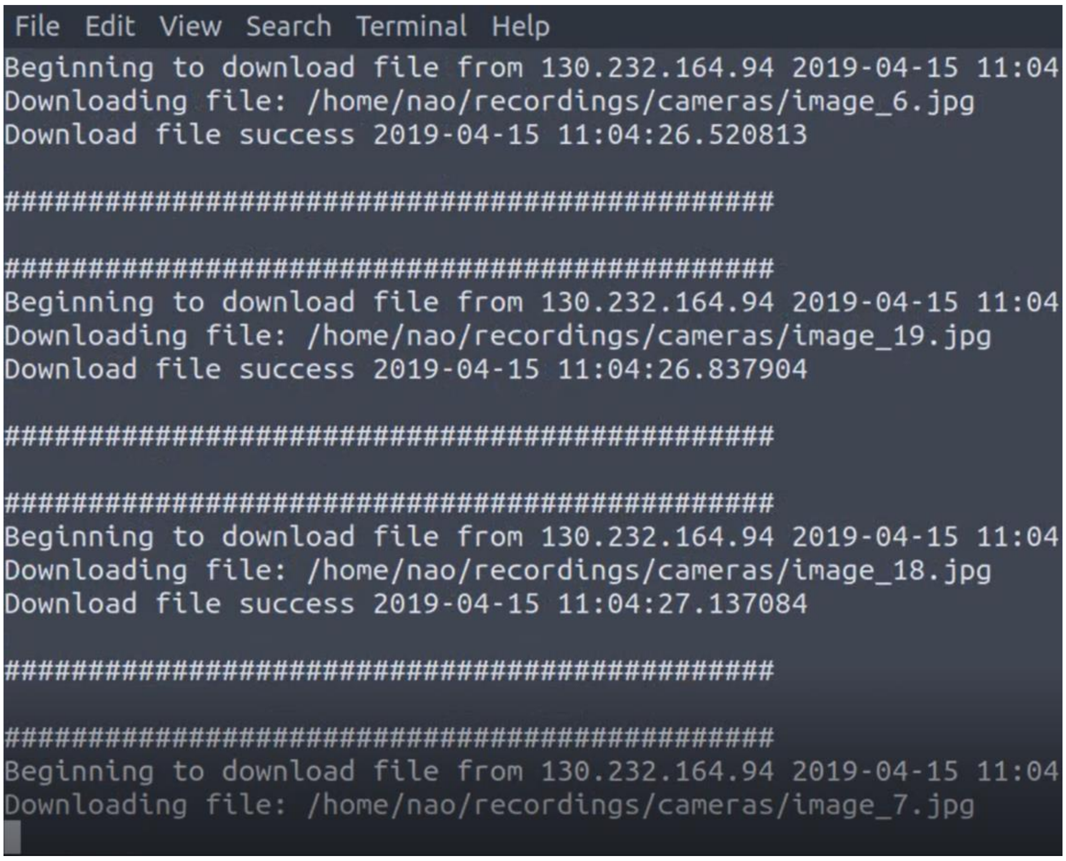
5.2 Transferring Pictures Implementation

A connection with Pepper robot is built via the robot's username and password to attain the right to handle the robot's files which are saved in robot's memory, and then the function is used i*paramiko.Transport()* to transfer the pictures to a local computer.

```
    t = paramiko.Transport(("130.232.164.94", 22))
    t.connect(username="nao", password="qwe123")
    sftp = paramiko.SFTPClient.from_transport(t)
    files=sftp.listdir(image_path)
    for f in files:
        print ('')
        print ('#############################################')
        print ('Beginning to download file from %s %s' %
        ("130.232.164.94",datetime.datetime.now()))
        print ('Downloading file:', os.path.join(image_path,f))

sftp.get(os.path.join(image_path,f),os.path.join(local_path,f))
        print ('Download file success %s'%datetime.datetime.now())
        print ('')
        print ('#############################################')
    t.close()
```

The implementation picture is shown in Figure 41.



```
Beginning to download file from 130.232.164.94 2019-04-15 11:04
Downloading file: /home/nao/recordings/cameras/image_6.jpg
Download file success 2019-04-15 11:04:26.520813

##############################################

##############################################
Beginning to download file from 130.232.164.94 2019-04-15 11:04
Downloading file: /home/nao/recordings/cameras/image_19.jpg
Download file success 2019-04-15 11:04:26.837904

##############################################

##############################################
Beginning to download file from 130.232.164.94 2019-04-15 11:04
Downloading file: /home/nao/recordings/cameras/image_18.jpg
Download file success 2019-04-15 11:04:27.137084

##############################################

##############################################
Beginning to download file from 130.232.164.94 2019-04-15 11:04
Downloading file: /home/nao/recordings/cameras/image_7.jpg
```

Figure 41. Transferring Pictures Implementation

5.3 Recognizing Face in Pictures Implementation

OpenCV-Python is used to recognize the face in pictures transferred from robot via the function *cv2.CascadeClassifier()* to read a “.xml” file named “haarcascade_frontalface_default”.

```
face_patterns=cv2.CascadeClassifier(r"./haarcascade_frontalface_de
fault.xml")
sample_img=cv2.imread(imgPath)
faces=face_patterns.detectMultiScale(sample_img, scaleFactor=1.1,
       minNeighbors=5, minSize=(100,100))
for(x,y,w,h) in faces:
    cv2.rectangle(sample_img, (x,y), (x+w,y+h), (0,255,0), 2)
cv2.imwrite('./detectedFaces/'+os.path.basename(imgPath),sample_im
g)
```

The implementation picture is shown in Figure 42.

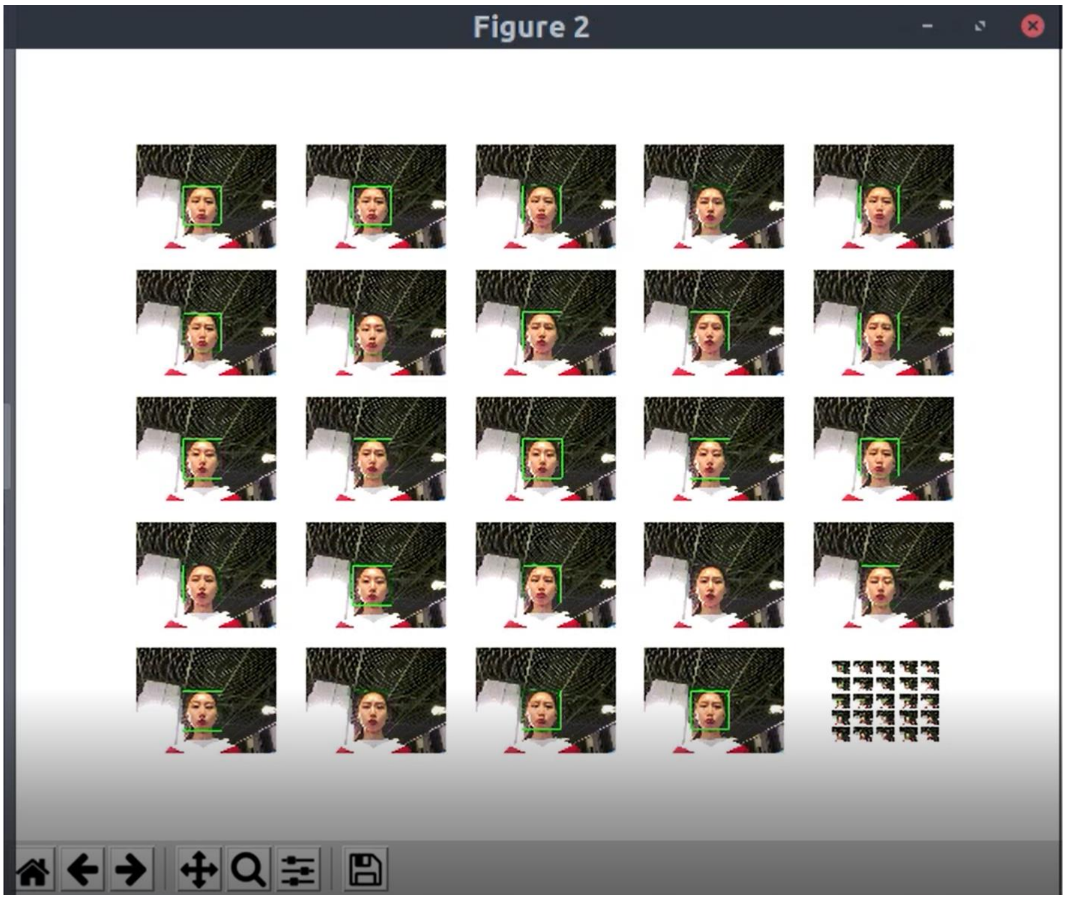


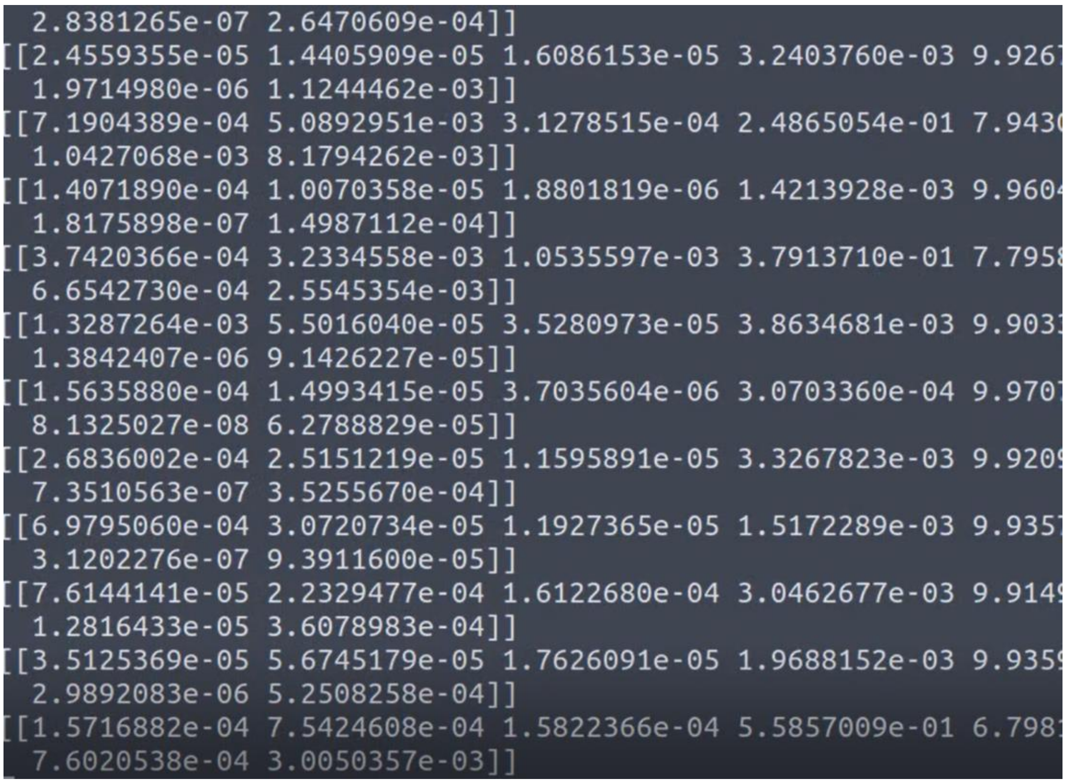
Figure 42. Recognizing Face in Pictures Implementation

5.4 Predicting the Pictures Implementation

The principle, process, and codes of this module have been described in the preceding article. The result of predicting the pictures will be printed as an array with seven elements which all the elements are 5 encoded values of real numbers. Every element represents a kind of emotion in the order as:

```
'angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral'
```

The implementation picture is shown in Figure 43.



```
 2.8381265e-07 2.6470609e-04]]
[[2.4559355e-05 1.4405909e-05 1.6086153e-05 3.2403760e-03 9.926
 1.9714980e-06 1.1244462e-03]]
[[7.1904389e-04 5.0892951e-03 3.1278515e-04 2.4865054e-01 7.943
 1.0427068e-03 8.1794262e-03]]
[[1.4071890e-04 1.0070358e-05 1.8801819e-06 1.4213928e-03 9.960
 1.8175898e-07 1.4987112e-04]]
[[3.7420366e-04 3.2334558e-03 1.0535597e-03 3.7913710e-01 7.795
 6.6542730e-04 2.5545354e-03]]
[[1.3287264e-03 5.5016040e-05 3.5280973e-05 3.8634681e-03 9.903
 1.3842407e-06 9.1426227e-05]]
[[1.5635880e-04 1.4993415e-05 3.7035604e-06 3.0703360e-04 9.970
 8.1325027e-08 6.2788829e-05]]
[[2.6836002e-04 2.5151219e-05 1.1595891e-05 3.3267823e-03 9.920
 7.3510563e-07 3.5255670e-04]]
[[6.9795060e-04 3.0720734e-05 1.1927365e-05 1.5172289e-03 9.935
 3.1202276e-07 9.3911600e-05]]
[[7.6144141e-05 2.2329477e-04 1.6122680e-04 3.0462677e-03 9.914
 1.2816433e-05 3.6078983e-04]]
[[3.5125369e-05 5.6745179e-05 1.7626091e-05 1.9688152e-03 9.935
 2.9892083e-06 5.2508258e-04]]
[[1.5716882e-04 7.5424608e-04 1.5822366e-04 5.5857009e-01 6.798
 7.6020538e-04 3.0050357e-03]]
```

Figure 43. Predicting the Pictures Implementation

5.5 Analyzing and Visualizing Result Implementation

All the predicting results will be collected together and then make the average value to improve the predicting accuracy.

```
sum=np.array(predictData)
averageList=np.mean(sum, axis=0)
```

The final result will be displayed as bar chart. Firstly a function is created for drawing bar chart for emotion predictions via *matplotlib.pyplot*. This function will receive the predicting results from CNN structure and then draw the bar chart as the data of predicting result, and then save the bar charts to local place.

```
objects = ('angry', 'disgust', 'fear', 'happy', 'sad', 'surprise',
'neutral')
y_pos = np.arange(len(objects))
plt.cla()
plt.bar(y_pos, emotions, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.ylabel('percentage')
plt.title('emotion')
```

```
savefig('./predictImg/'+predictImgName)
```

The implementation picture is shown in Figure 44 and Figure 45.

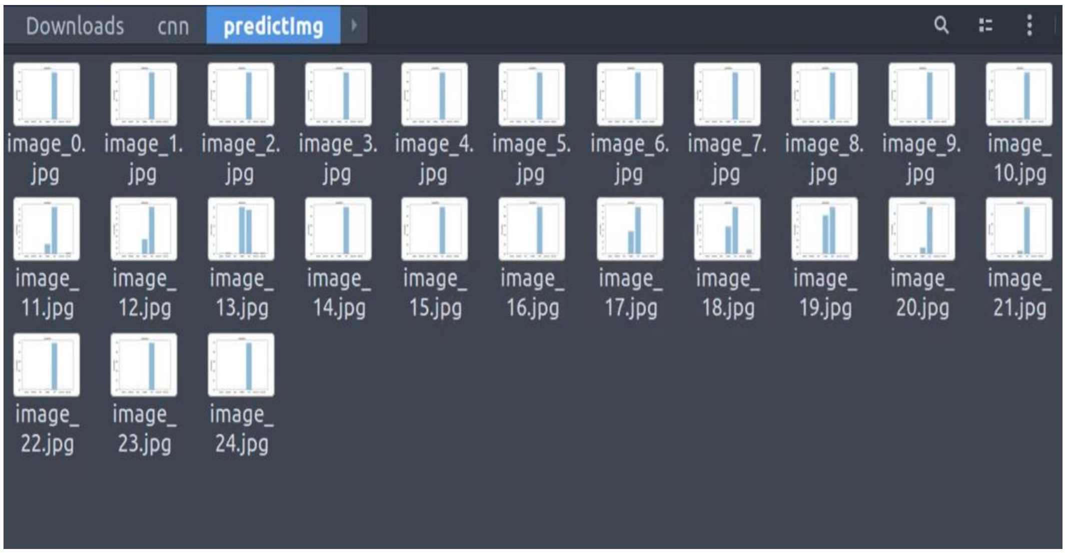


Figure 44. Visualizing Results in Bar Chart

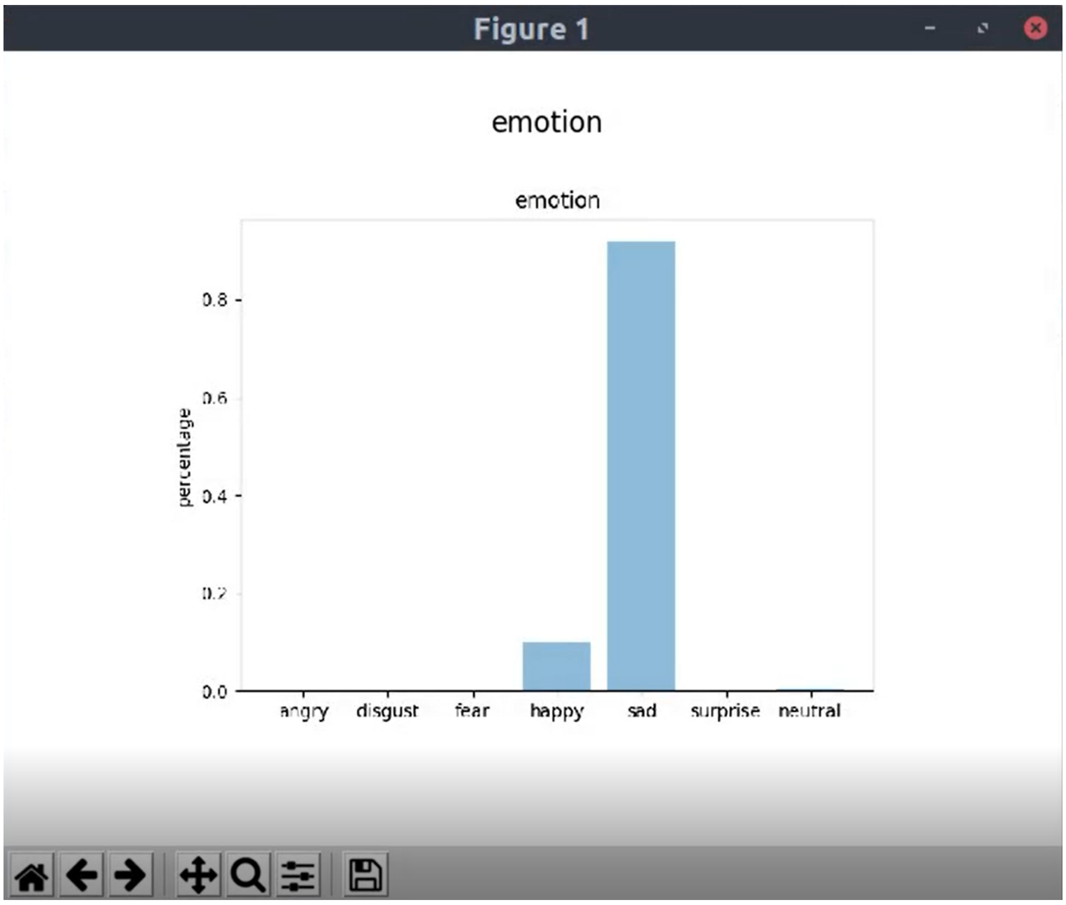


Figure 45. Final Result after Analyzing

The complete implementation video link is:
https://www.youtube.com/watch?v=pbwMU2AFOv0

6 IMPROVEMENT AND CONCLUSION

The main thing which needs to be improved in this project is to prevent overfitting. Overfitting refers to the problem in the process of fitting model parameters. Because the training data contains sampler errors, and complex model, which contains four convolution layers and two fully connected layers in this project, takes the sample errors in the training process, and the sample errors are also well fitted. The concrete performance is that the final model works well on the training set, and the effect on the test set is poor. The model generalization ability is poor. Dropout is used in this project to solve this problem, but the effect is not so good. There are some other improvement methods in the future:

i. Get more data.

   There are many ways to get more data. For instance, in this project, taking several pictures continuously (25) to get more data from the source of data has been taken to prevent overfitting. However, a significant increase in the number of taking pictures is not easy for a robot to fulfill. In addition, it is impossible to know how much data is enough in this project. The feasible method to get more data to prevent overfitting in the future is using the technology, Data Augmentation, which means to expand the data with certain rules.

ii. Use a suitable model

   Overfitting is mainly caused by two reasons, too little data, and too complex model. So using a model in the appropriate complexity to fit the real rules is also a way to avoid fitting too many sample errors, for instance, early stopping, regularization and increasing noise.

iii. Combine multiple models

   Train multiple models to collect all the output of each model and then the average output as the final result, for instance, Bagging, Boosting and Dropout, which has been used in this project.

In this project, the understanding of the structure and principle of artificial neural networks in deep learning, as well as the learning of related knowledge of convolutional neural network are all very core part. In the process to build a CNN structure, there are many factors that influence the accuracy of the deep learning result and the speed of training, like the number of convolution layers and fully connected layers, the activation function, the dropout and so on.

As for the Pepper robot, there is no doubt that the humanoid robot is the best choice to replace human resource to detect and take pictures for patients. With the rapid technological development, the research of fields related to Robotics and Computer Aided Design (CAD), such as healthcare robotics and relevant visual computing direction, becomes more and more popular to help hospital take care about patients. During the development of this project, it is difficult to find a suitable convolutional neural network to train and test the Kaggle's Fec2013 dataset so I trained several times for different structures from two convolution layers up to five convolution layers. Finally, four convolution layers had been chosen because of the highest accuracy. Excluding the overfitting, the limit of facial emotional categories is also a part which needs to be improved. The dataset, which this project used, only have 7 kinds of emotions to let computer study and predict. In the future, other kinds of facial emotion still need to be added and the size of the dataset also needs to be enriched.

In this project, most of the module has been implemented. Users can look at Pepper robot to let it take several pictures for him and these pictures can be transferred to the local computer successfully. Convolutional Neural Network can receive these pictures to predict the facial emotion by comparing with the feature file and the final accuracy is about 0.67. After averaging all the results, the final result is displayed as a bar chart to illustrate the user's mental stature with 7 labels: angry, disgust, fear, happy, sad, surprise, neutral.

In this thesis, the history and process of related technologies and development environments like a neural network, convolution neural network, and the Pepper robot are all introduced, also the whole working process of this project having been illustrated via several flowchart and codes.

REFERENCE

[1] M. J. Z. H.-d. Y. Jing-jing, "Application and prospect of Artificial Neural Network," *Electronic Design Engineering,* p. 4, 2011.

[2] L. Shu, "Popular Science: Artificial neural network VS biological neural network? " 24 05 2018. [Online]. Available: http://share.shautonews.com/blog/39.

[3] J. R. F. Manag, "Equalizing Seasonal Time Series Using Artificial Neural Networks in Predicting the Euro–Yuan Exchange Rate," *Journal of Risk and Financial Management— Open Access Journal,* vol. 12, no. 2, 2019.

[4] Y. .. Blog, "Speeding up sigmoid function by approximating exponential function," 20 03 2011. [Online]. Available: http://ybeernet.blogspot.com/2011/03/speeding-up-sigmoid-function-by.html.

[5] Y. W. Y. C. S. L. Y. W. Songsong Cheng, "A universal modified LMS algorithm with iteration order hybrid switching," *ISA Transactions,* vol. 67, pp. 67-75, 03 2017.

[6] H. N. a. N. M. Teijiro Isokawa, "Quaternionic Multilayer Perceptron with Local Analyticity," *Information — Open Access Journal,* vol. 3, no. 4, 2012.

[7] J. Zhang, "Introduction of Convolution Neural Network," Zhihu, 26 05 2018. [Online]. Available: https://zhuanlan.zhihu.com/p/25249694.

[8] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way," *Medium,* 2018.

[9] X. Qiu, "Neural Network and deep learning," 04 04 2019. [Online]. Available: https://nndl.github.io/chap-%E5%8D%B7%E7%A7%AF%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C.pdf.

[10] T. Dettmers, "Understanding Convolution in Deep Learning," 26 03 2015. [Online]. Available: https://timdettmers.com/2015/03/26/convolution-deep-

learning/.

[11] cs321n, "Convolutional Neural Networks (CNNs / ConvNets)," CS231n Convolutional Neural Networks for Visual Recognition, [Online]. Available: https://cs231n.github.io/convolutional-networks/.

[12] A. Sachan, "TensorFlow Tutorial: 10 minutes Practical TensorFlow lesson for quick learners," CV-Tricks.com, [Online]. Available: https://cv-tricks.com/artificial-intelligence/deep-learning/deep-learning-frameworks/tensorflow-tutorial/.

[13] “Keras Document,” [Online]. Available: https://keras.io/.

[14] S. Krishnamurthy, "MXNet - Keras Integration Design," Confluence, 08 06 2018. [Online]. Available: https://cwiki.apache.org/confluence/display/MXNET/MXNet+-+Keras+Integration+Design.

[15] J. Engel, "SoftBank Taps Affectiva to Boost Pepper Robot’s Emotional IQ," Xconomy, 28 08 2018. [Online]. Available: https://xconomy.com/boston/2018/08/28/softbank-taps-affectiva-to-boost-pepper-robots-emotional-iq/.

[16] "SOFTBANK ROBOTICS DOCUMENTATION," SOFTBANK ROBOTICS , [Online]. Available: http://doc.aldebaran.com/2-5/family/pepper_user_guide/index_pepper_user.html.

[17] "NAOqi Framework," [Online]. Available: http://doc.aldebaran.com/1-14/dev/naoqi/index.html.

[18] "Children in Finland increasingly treated for mental health issues," yle, 28 11 2018. [Online]. Available: https://yle.fi/uutiset/osasto/news/children_in_finland_increasingly_treated_for_mental_health_issues/10529653.

[19] J. Liang, "Design of an Automatic Facial Expression Detector," 26 01 2018. [Online]. Available: https://uwaterloo.ca/applied-mathematics/sites/ca.applied-mathematics/files/uploads/files/jian_liangs_essay_final_version.pdf.