VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Wenhao Zhu

CAPTCHA Recognition System

Information Technology
2019

FOREWARD

I have been studying in Vaasa Ammattikorkeakoulu (Vaasa University of Applied Sciences) for about 3 years since 2016. All of these great experiences came from a unique exchange opportunity provided by the Department of International Relations and HMT Affairs Office, Hubei University of Technology. It is a memorable experience during my whole life and I would like to express my appreciation to everyone who helped me.

First of all, I would like to show my great appreciation to my supervisor, Dr. Yang Liu, who provides a great platform and provides many opportunities for me and many IT students. I had a big delay in my project and thesis process, but Dr. Yang Liu is very tolerant and offered me the chance to finish my thesis. Without him, it would be hard for me to finish it.

Also, I would like to thank all the staffs in VAMK, especially lecturers Dr. Chao Gao, Dr. Ghodrat Moghadampour, Mr. Santiago Chavez Vega, and Dr. Seppo Makinen. They offer high-quality lectures and have high professionalism, which helps me understand deeply in the fields where they have research for many years.

Also, I would like to express my great gratitude to my parents who always encourage me to do whatever I decided to and support me financially and emotionally. And I would thank all my friends who study and work with me in Technobothnia RoboCup Laboratory, I wish you very good luck in the future.

Wenhao Zhu

Vaasa, Finland

19.05.2019

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Program of Information Technology

ABSTRACT

Author	Wenhao Zhu
Title	CAPTCHA Recognition System
Year	2019
Language	English
Pages	59
Name of Supervisor	Yang Liu

Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA), is a public fully automatic program that distinguishes users from computers or people. This thesis developed a CAPTCHA recognition system that can be deployed on the NAO robot, a humanoid robot to pass the Turing test and also can be deployed on web services in order to provide a recognition service.

The recognition system uses convolutional neural network to extract features in CAPTCHA image and encode data with one-hot encoding system which is widely used in multiclassification. Python is the programming language used in developing this project, TensorFlow and Keras library are used to easily establish a neural network. NAO robot version is v5 and code testing is on Ubuntu 16.04 release.

The final recognition model showed about 99.67% accuracy on train dataset and 98.10% accuracy on test dataset with suitable optimizer and loss function. According to the one-hot encoding features when regulated data, the accuracy is a bit high than it performed in real applications. Due to a large amount of CAPTCHA data for the combination of numbers and letters, the CAPTCHA in this thesis dataset consists only of numbers, which could be improved by using datasets contains numbers and letters CAPTCHA.

Keywords: NAO Robot, CAPTCHA, Convolutional Neural Network

CONTENTS

LIST OF FIGURES AND TABLES

1 INTRODUCTION

1.1 Background

As artificial intelligence and deep learning have been gaining more and more attention since 2010, people began again to talk about where the boundaries between human beings and machines are. CAPTCHA, an acronym for "Completely Automated Public Turing test to tell Computers and Humans Apart", is a fully automatic program that distinguishes users from computer or people. A commonly used CAPTCHA test is to let the user input the text or number displayed on a distorted picture if user types the correct text inside of CAPTCHA, he or she are believed a human (distortion is to avoid being recognized by a computer program such as Optical Character Recognition (OCR)), otherwise, the program will generate a new CAPTCHA to do the test again within limited chances. If the input text and CAPTCHA text do not match more than a certain number of times, then it will be considered a robot and the previous work will not be recognized. Recognizing CAPTCHA is consider a hard work in the past, however, thanks to the deep learning algorithm, it can be possible in today's technology.

1.2 Purpose

The goal of this thesis is to build a CAPTCHA recognition system, which could recognize what texts contained in the CAPTCHA image, with an over 50% accuracy and universal applicability which could be deployed on NAO robot in a real situation. To reach the goal, a series of researches and works are done to achieve the purpose. First, two datasets should be prepared for training and testing the model. Secondly, CAPTCHA images are regulated and processed in order to fit the training format and train the model more quickly. After these pre-processes for data, a convolutional neural network would be established to train the model with multiple optimizers and loss functions. Then we analyze the result, choose the model performed best as our final model. Finally, we connect to the NAO robot and deploy our recognition system on it. We also provide a recognition interface for web services that can be invoked by certain command.

1.3 Overall Structure

This thesis consists of ten chapters. The first chapter is the introduction part, which includes the relevant background knowledge, the purpose and the structure of this thesis. Chapter two mainly introduces the technologies and development tools used in this project, including programming language, neural network library and instruction about NAO robot. In Chapter three, overall design of the whole project is given, with flow chart and documental explanations. Chapter four illustrates how data is been processed before training in the convolutional neural network by some methods, in order to suit the data format in the Keras library. Chapter five and six are the model designing and training parts, during these two chapters, how CNN training structure is shown by several figures which can clearly explain the CNN structure. Besides, how our model is trained is detailed described. After our model has been trained, an analysis of the result is given by chapter seven, includes the choice of optimizers and loss functions by analyzing the output results. Chapter eight is about deployment, in this chapter the suitable trained model is deployed on the NAO robot to establish a CAPTCHA recognition system in a real situation. Besides, a CAPTCHA-recognize web service is also provided through the Flask web framework in this chapter. The final chapter is the conclusion part, in which some recommendation improvements for future research are illustrated as well as the limitations.

2 TECHNOLOGIES AND DEVELOPMENT TOOLS

2.1 Technologies

2.1.1 Artificial Intelligence

Artificial Intelligence (AI) has gradually appeared in many news reports recently. However, it started far early since the 1950s. AI has 3 development periods. From 1956, when the concept of AI was first presented at the Dartmouth meeting, to 1980s is the first development period, scientists were starting to do some researches in this field, mainly focus on voice recognition, cryptography, and expert system. IBM Deep Blue was invented in this period and beat a world champion chess player. After that, machine learning and simple perceptron model was introduced to this field which brought the second big development. Advertisements Blocker is a typical application based on a machine learning method. Time comes to the twenty-first century, people realized that deep learning has a good application when the amount of data and computing capability are sufficient. In 2010, Dr. Fei-fei Li established ImageNet with many other researchers brought AI to the third development period.

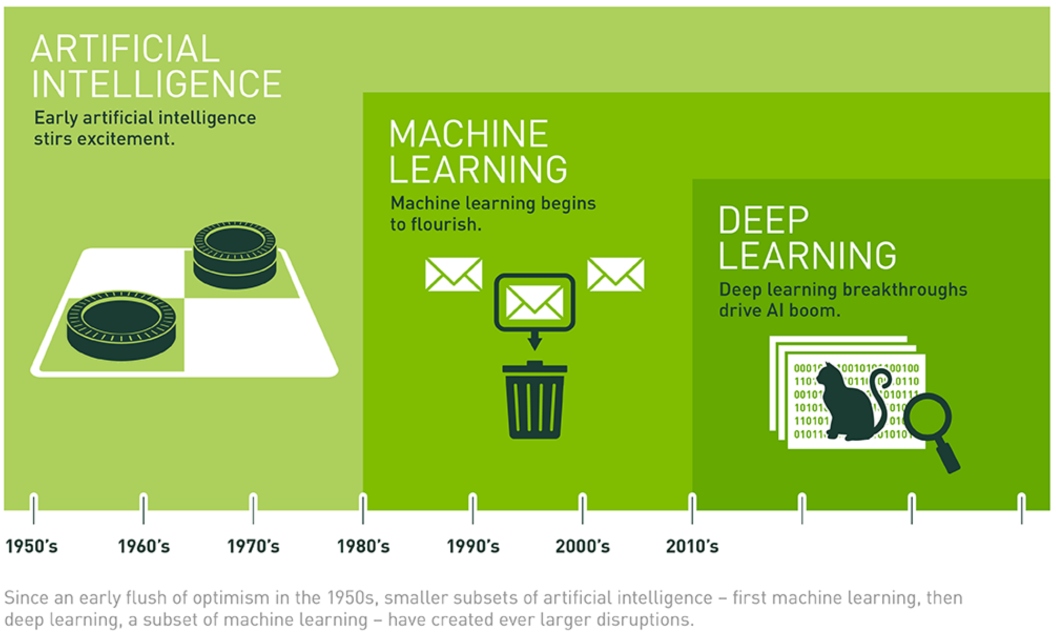

ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's
Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

Figure 1. Development of Artificial Intelligence [1]

2.1.2 Perceptron Model

In 1957, psychologist Frank Rosenblatt defined the concept of the Perceptron algorithm based on Warren and Walters' work. They classify biological neurons as simple logic gates with binary outputs. In a more intuitive way, a neuron can be understood as a child node of a neural network in a biological brain. Here, the variable signal, considered as the input signal, reaches the dendrites. When the intensity of the input signal exceeds a certain threshold, an output signal is generated and transmitted by the dendrites. [2]

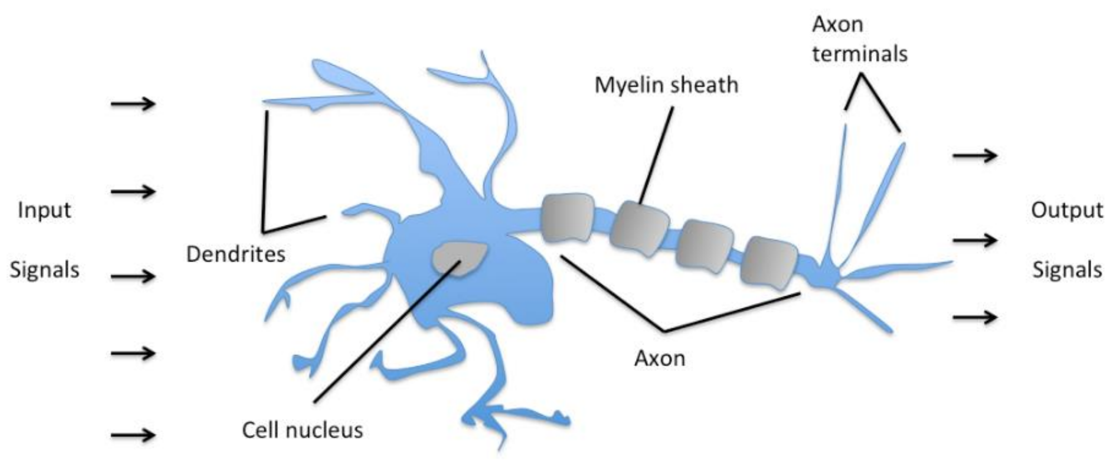


Figure 2. Schematic of a biological neuron [2]

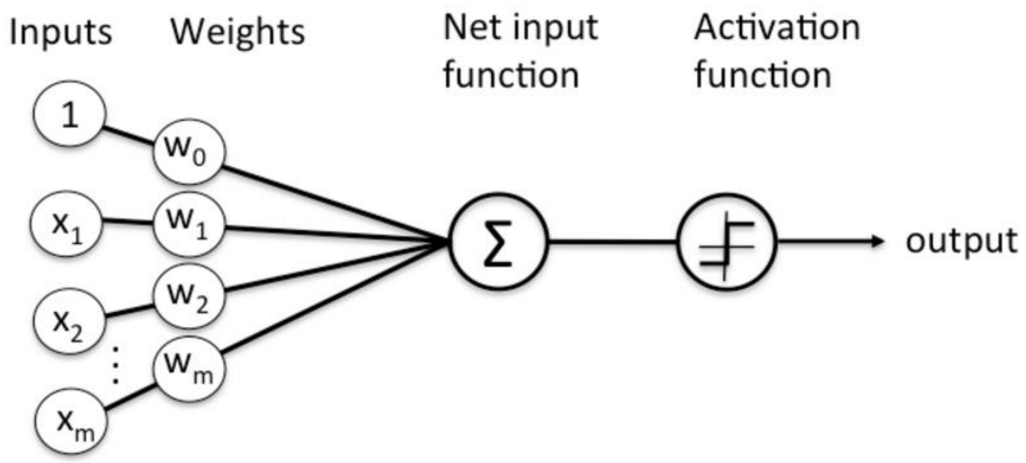


Figure 3. Perception Model [2]

The purpose of the perceptron algorithm is to learn a weight vector w for a sample set of multidimensional features, such that after multiplying w by the input feature vector x, based on the result, it can be determined whether a neuron is activated. Perceptron model is a model of two class classification.

2.1.3 Deep Learning

The concept of deep learning stems from the research of artificial neural networks. Artificial neural network is a computational model that inspires biological neural networks that process information from the human brain. The artificial neural network has made a series of breakthroughs in the fields of speech recognition, computer vision, and text processing. Multilayer Perceptron (MLP) is a specific artificial neural network, and it is also called Artificial Neural Network (ANN). In addition to the input and output layer, it can have multiple hidden layers in the middle, the simplest MLP contains only one hidden layer, that is, the structure of the three layers.

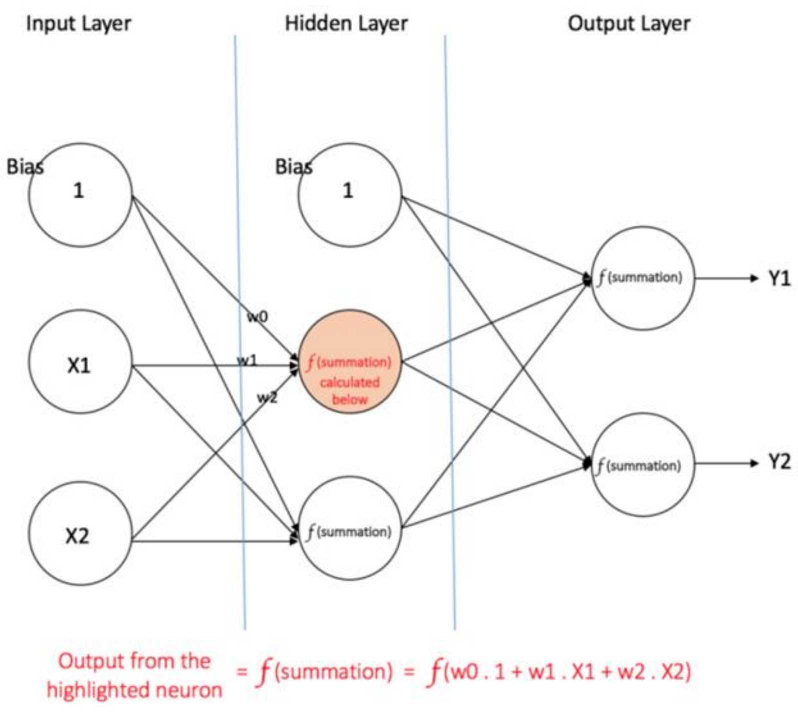

Input Layer
Hidden Layer
Output Layer
Bias
1
Bias
1
X1
X2
w0
w1
w2
f(summation) calculated below
f(summation)
f(summation)
f(summation)
Y1
Y2
Output from the highlighted neuron = f(summation) = f(w0 . 1 + w1 . X1 + w2 . X2)

Figure 4. MLP with some hidden layers [3]

2.1.4 Convolutional Neural Network

Convolutional Neural Network is an application of deep learning algorithms in the field of image processing, shown below. Convolution neural network is an efficient identification method which has been developed in recent years and has attracted wide attention. In the 1960s, Hubel and Wiesel in studying the neurons used for local sensitivity and directional selection in the cat cortex, they found that their unique network structure can effectively reduce the complexity of the feedback neural network, and then put forward the convolution neural network (Convolutional Neural Networks is referred to as CNN). The new identification machine proposed by K. Fukushima in 1980, which is the first implementation network of convolution neural network. Subsequently, more researchers made improvements to the network. Among them, the representative research results are the "improved cognitive machine" proposed by Alexander and Taylor, which synthesizes the advantages of various improvement methods and avoids the time-consuming error reverse propagation. [3]

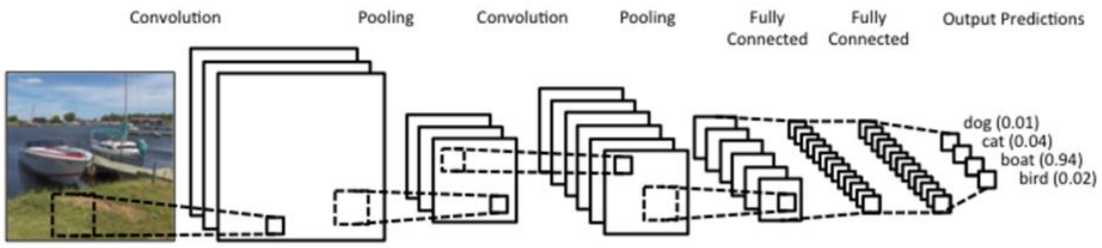


Figure 5.CNN Concept Diagram [4]

There are several layers that contain in the CNN:

1. Convolution Layer: The convolution layer can simulate the nature of the local sensory field, which is not fully connected to the previous layer, but is a small area connection. This small piece is the local receptive field. By constructing specific convolution neurons, the artificial neural network can simulate the properties of different neurons that stimulate different reactions to different shapes. As shown in Figure 6, a neuron forms a feature map by processing layer, and then multiple layers will superimpose, and the number of layers gradually will accumulate.

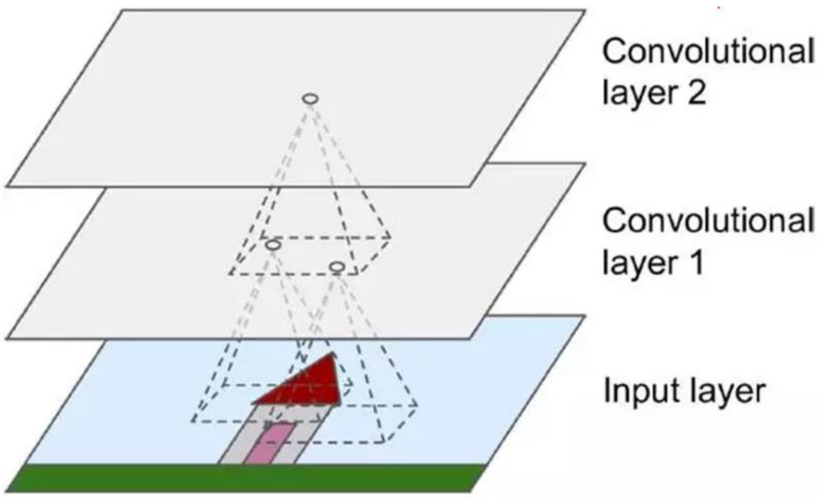


Figure 6. Convolution Layer [5]

2. Pooling layer: The memory consumption is huge when the picture size is large, and the role of pooling layer is to condense the effect and ease the memory pressure, which means selecting a certain size area and representing the area with a representative element. There are two specific Pooling, averaging and taking the maximum value, and the common type of pooling layer is Max-Pooling, shown in Figure 7. The main benefit of Max-Pooling is that if the picture is panned a few Pixels, the judgment of the result will not have an impact at all, and Max-Pooling has a good anti-noise function.

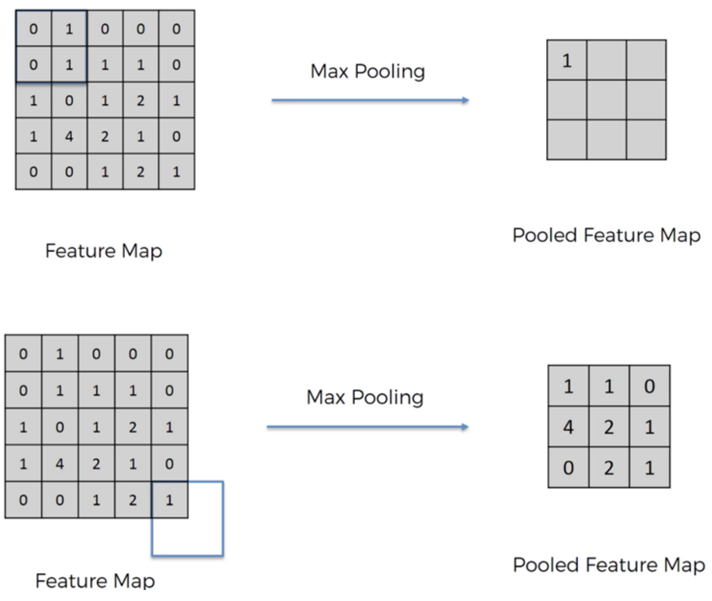


Figure 7. Max-Pooling [4]

3. Flatten: Convolution layer can't connect with Dense fully connected layer directly. The data of convolution layer needs to be flattened, and then can be directly added to the Dense layer, so the use of flatten is to compress the height, width and channel data from convolution layer in 2-dimension into a 1-dimensional array which the length is height x width x channel, in order to connect with fully connected layer successfully, shown in Figure 8.

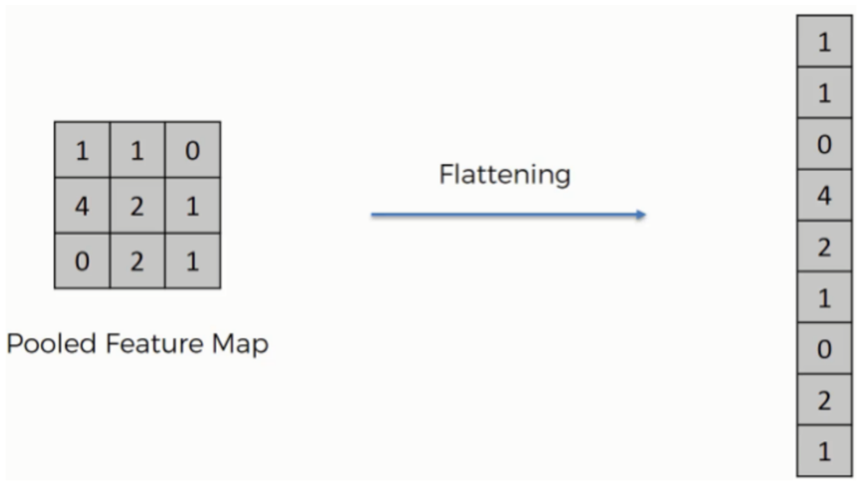


Figure 8. Flattening Process [6]

4. Fully connected layer: The fully connected layer (FC) acts as a "classifier" throughout the convolution neural network, shown in Figure 9. If the operation of convolution layer, pool layer, and activation function layer are to map the original

data to the hidden layer feature space, the fully connected layer plays the role of mapping the "distributed feature representation" to the sample mark-up space. In practical use, the fully connected layer can be realized by convolution operation.

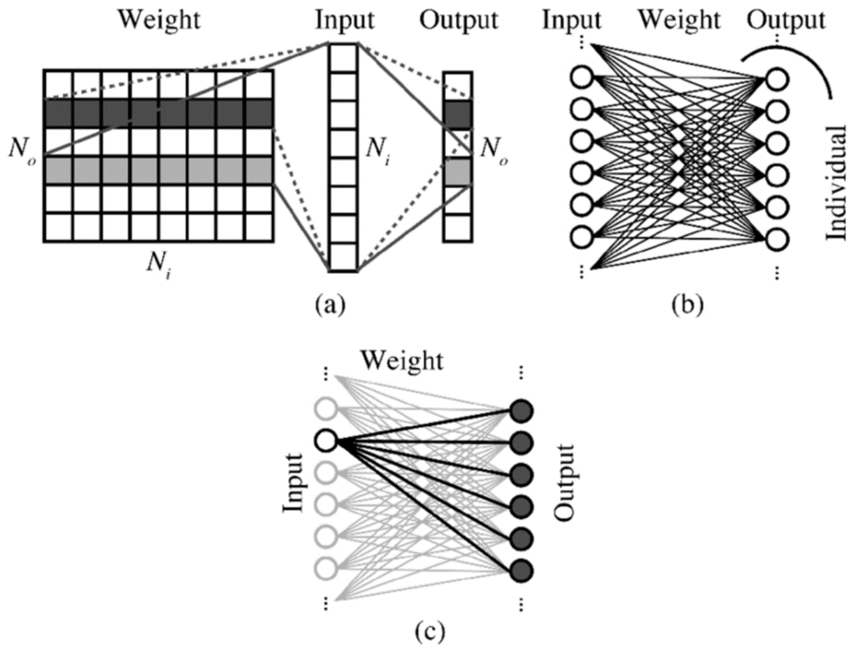


Figure 9. Full Connected Layer Structure [7]

2.2 Programming Tools

2.2.1 Python

Python is a simple, interpreted, interactive, high-level programming language. Clear and elegant syntax make it widely acclaimed. It has most of the features of an object-oriented language for full object-oriented programming. Python is portable and is cross-platform for a variety of operating systems includes Windows, MacOS, and Linux. These kinds of features above make it very popular internationally and are gaining more and more applications. Python version 2.7 is used in this project and the IDE is PyCharm. [3]

2.2.2 OpenCV

OpenCV (Open source Computer Vision) is an open source library that widely used in the field of computer vision. The library is written in C and C ++ and can be run on Windows, Linux, Mac OS systems. OpenCV supports interfaces in many programming languages, the interface of Python is used in this thesis and the version is 4.1.0. All the library's code is optimized and computationally efficient because it is more focused on design as an open source library for real-time systems. OpenCV uses C language to optimize, and, in the multi-core machine above, it will One of its goals is to provide a friendly machine vision interface function that enables complex machine vision products to accelerate. The library contains over 500 interface functions spanning areas such as industrial product testing, medical image processing, security, user interface, camera Calibration, 3D imaging, machine vision and more.

2.2.3 NumPy

NumPy is a powerful Python library for performing computations on multidimensional arrays. The word NumPy comes from two words - Numerical and Python. NumPy provides a large number of library functions and operations that help programmers easily perform numerical calculations. This type of numerical calculation is widely used in the machine learning model, Image processing and computer graphics and math tasks. [10]

2.2.4 Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits. [11]

2.2.5 Captcha

Captcha library is a python library that enables users to generate audio and image CAPTCHAs. Captcha library is easy to install and can be used in a simple way.

2.2.6 TensorFlow

TensorFlow is an open source software library for machine learning for a variety of perception and language understanding tasks. In 2012, Google released its first generation of large-scale distributed deep learning framework - Google Distbelief, which is widely used in various applications of Google, such as Google Translate and YouTube. In 2015, Google has open sourced the second-generation of large-scale distributed deep learning platform – TensorFlow. Nowadays many companies are using this technology, including Airbnb, Uber and Intel.

TensorFlow uses a dataflow graph to represent computation in terms of the dependencies between individual operations.

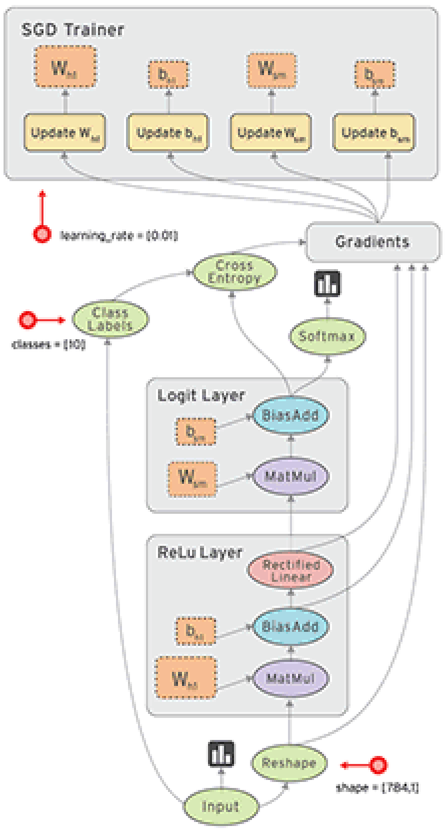


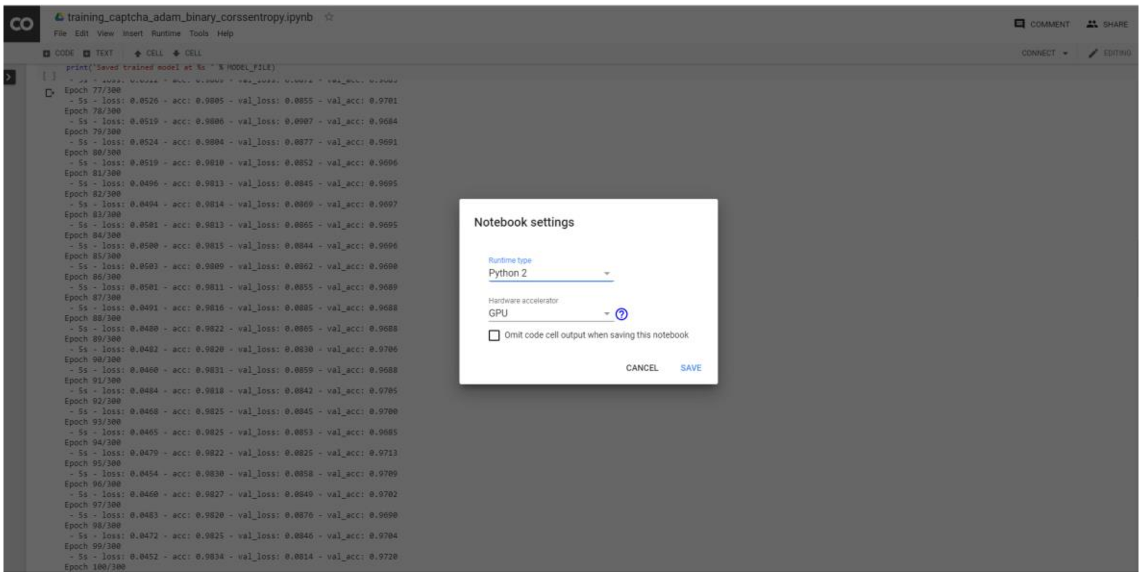
Figure 10. Dataflow graph in TensorFlow [4]

2.2.7 Keras

Keras is a high-level neural networks 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. [13]

2.2.8 Google Colaboratory

Google Colaboratory, also known as Colab, is a free Jupyter notebook environment (a web-based interactive computing environment, for creating Jupyter Notebook documents) that requires no setup and runs entirely in the cloud. With Colaboratory you can write and execute code, save and share your analyses, and access powerful computing resources, all for free from your browser. Colaboratory was originally part of the Jupyter project but was eventually taken over by Google. Colab supports free GPU training acceleration. [14]


training_captcha_adam_binary_corssentropy.ipynb
File Edit View Insert Runtime Tools Help
CODE TEXT CELL CELL
COMMENT SHARE
CONNECT EDITING
CANCEL SAVE

```
print('Saved trained model at %s ' % MODEL_FILE)
Epoch 77/300
 - 5s - loss: 0.0526 - acc: 0.9805 - val_loss: 0.0855 - val_acc: 0.9701
Epoch 78/300
 - 5s - loss: 0.0519 - acc: 0.9806 - val_loss: 0.0907 - val_acc: 0.9684
Epoch 79/300
 - 5s - loss: 0.0524 - acc: 0.9804 - val_loss: 0.0877 - val_acc: 0.9691
Epoch 80/300
 - 5s - loss: 0.0519 - acc: 0.9810 - val_loss: 0.0852 - val_acc: 0.9696
Epoch 81/300
 - 5s - loss: 0.0496 - acc: 0.9813 - val_loss: 0.0845 - val_acc: 0.9695
Epoch 82/300
 - 5s - loss: 0.0494 - acc: 0.9814 - val_loss: 0.0869 - val_acc: 0.9697
Epoch 83/300
 - 5s - loss: 0.0501 - acc: 0.9813 - val_loss: 0.0865 - val_acc: 0.9695
Epoch 84/300
 - 5s - loss: 0.0500 - acc: 0.9815 - val_loss: 0.0844 - val_acc: 0.9696
Epoch 85/300
 - 5s - loss: 0.0503 - acc: 0.9809 - val_loss: 0.0862 - val_acc: 0.9690
Epoch 86/300
 - 5s - loss: 0.0501 - acc: 0.9811 - val_loss: 0.0855 - val_acc: 0.9689
Epoch 87/300
 - 5s - loss: 0.0491 - acc: 0.9816 - val_loss: 0.0885 - val_acc: 0.9688
Epoch 88/300
 - 5s - loss: 0.0480 - acc: 0.9822 - val_loss: 0.0865 - val_acc: 0.9688
Epoch 89/300
 - 5s - loss: 0.0482 - acc: 0.9820 - val_loss: 0.0830 - val_acc: 0.9706
Epoch 90/300
 - 5s - loss: 0.0460 - acc: 0.9831 - val_loss: 0.0859 - val_acc: 0.9688
Epoch 91/300
 - 5s - loss: 0.0484 - acc: 0.9818 - val_loss: 0.0842 - val_acc: 0.9705
Epoch 92/300
 - 5s - loss: 0.0468 - acc: 0.9825 - val_loss: 0.0845 - val_acc: 0.9700
Epoch 93/300
 - 5s - loss: 0.0465 - acc: 0.9825 - val_loss: 0.0853 - val_acc: 0.9685
Epoch 94/300
 - 5s - loss: 0.0479 - acc: 0.9822 - val_loss: 0.0825 - val_acc: 0.9713
Epoch 95/300
 - 5s - loss: 0.0454 - acc: 0.9830 - val_loss: 0.0858 - val_acc: 0.9709
Epoch 96/300
 - 5s - loss: 0.0460 - acc: 0.9827 - val_loss: 0.0849 - val_acc: 0.9702
Epoch 97/300
 - 5s - loss: 0.0483 - acc: 0.9820 - val_loss: 0.0876 - val_acc: 0.9690
Epoch 98/300
 - 5s - loss: 0.0472 - acc: 0.9825 - val_loss: 0.0846 - val_acc: 0.9704
Epoch 99/300
 - 5s - loss: 0.0452 - acc: 0.9834 - val_loss: 0.0814 - val_acc: 0.9720
Epoch 100/300
```

Notebook settings

Runtime type: Python 2

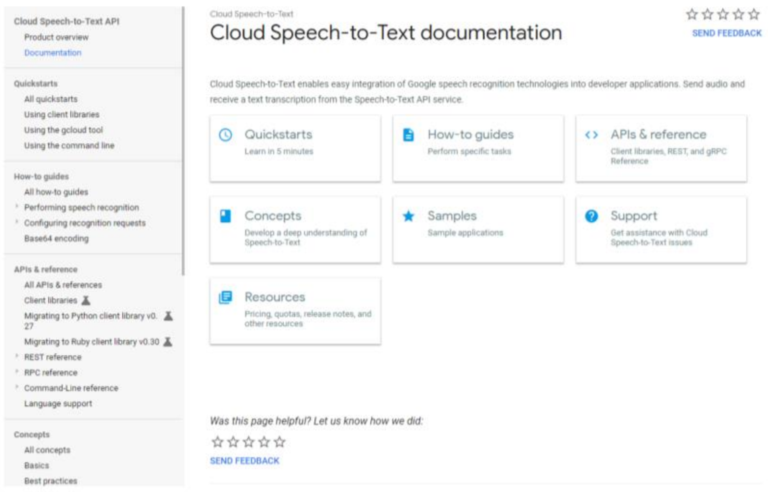
Hardware accelerator: GPU

[ ] Omit code cell output when saving this notebook

Figure 11. Training model on Google Colaboratory with GPU acceleration.

2.2.9 Google Cloud Speech-to-Text

Google Cloud Speech-to-Text enables developers to convert audio to text by applying powerful neural network models in an easy-to-use API. The API recognizes 120 languages and variants to support your global user base. You can enable voice command-and-control, transcribe audio from call centers, and more. It can process real-time streaming or prerecorded audio, using Google’s machine learning technology. [14]


Cloud Speech-to-Text API
Product overview
Documentation

Quickstarts
All quickstarts
Using client libraries
Using the gcloud tool
Using the command line

How-to guides
All how-to guides
Performing speech recognition
Configuring recognition requests
Base64 encoding

APIs & reference
All APIs & references
Client libraries
Migrating to Python client library v0.27
Migrating to Ruby client library v0.30
REST reference
RPC reference
Command-Line reference
Language support

Concepts
All concepts
Basics
Best practices

Cloud Speech-to-Text

Cloud Speech-to-Text documentation

SEND FEEDBACK

Cloud Speech-to-Text enables easy integration of Google speech recognition technologies into developer applications. Send audio and receive a text transcription from the Speech-to-Text API service.

Quickstarts
Learn in 5 minutes

How-to guides
Perform specific tasks

APIs & reference
Client libraries, REST, and gRPC Reference

Concepts
Develop a deep understanding of Speech-to-Text

Samples
Sample applications

Support
Get assistance with Cloud Speech-to-Text issues

Resources
Pricing, quotas, release notes, and other resources

Was this page helpful? Let us know how we did:

SEND FEEDBACK

Figure 12. Google Cloud Speech-to-Text

2.2.10 Flask

Flask is a lightweight web application framework written in Python, based on the Werkzeug WSGI toolkit and the Jinja2 template engine. Flask is compliant with the BSD licenses. It implements the core of the framework in a minimalist way while retaining scalability.

Flask has a default development server and is easy to debug. If the application mode is set to debug, whenever files change in this application, the server would detect them and reload automatically to get the application updated. At the same time, Flask provides us with a helpful debugger on the occurrence of exceptions. [5]

2.3 NAO Robot

NAO robot is a humanoid robot developed by the Aldebaran Robotics in France (acquired by SoftBank Group in 2015 and rebranded as SoftBank Robotics) and is widely used for research and education purpose in numerous academic institutions. NAO can be used as a research robot in schools, universities and universities. It is responsible for teaching programming and developing human-machine interaction. NAO robot has varieties of sensors which allow the robot can move like human beings. NAO robot has dual cameras, which can support it be sued to fully identify and position objects at 30 frames per second and the resolution is 1280 x 960 pixels. It contains 4 microphones which are located on the front, behind, left and right respectively, and all of them are 300 Hz to 8000 Hz. NAO robot also has a stereo system which consists of two speakers and an LED light. The robot has two CPUs, which are both Intel series. One is located on the head of the robot, the kernel is run in the language of Linux, and supports the Aldebaran company's own development of the NAOqi framework.

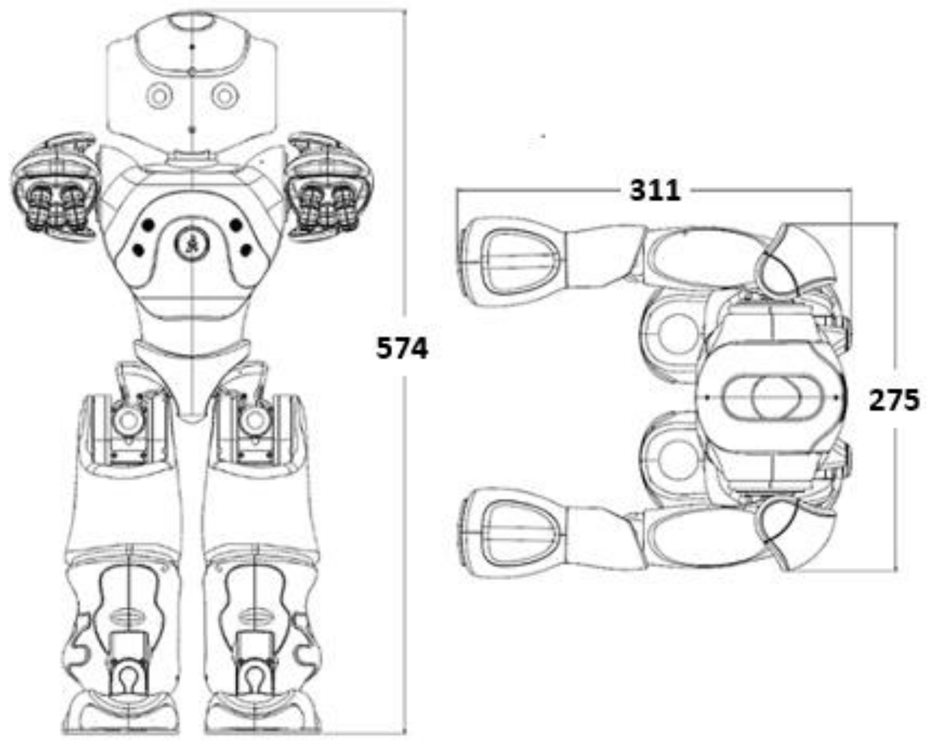


Figure 13. NAO robot structure (Unit: mm) [6]

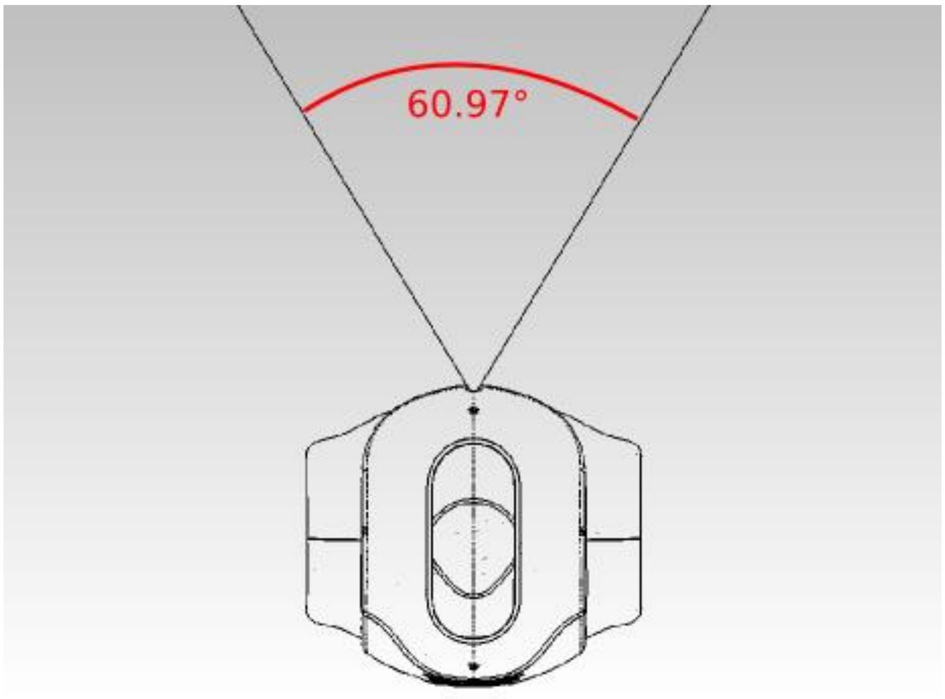


Figure 14. NAO Video camera [7]

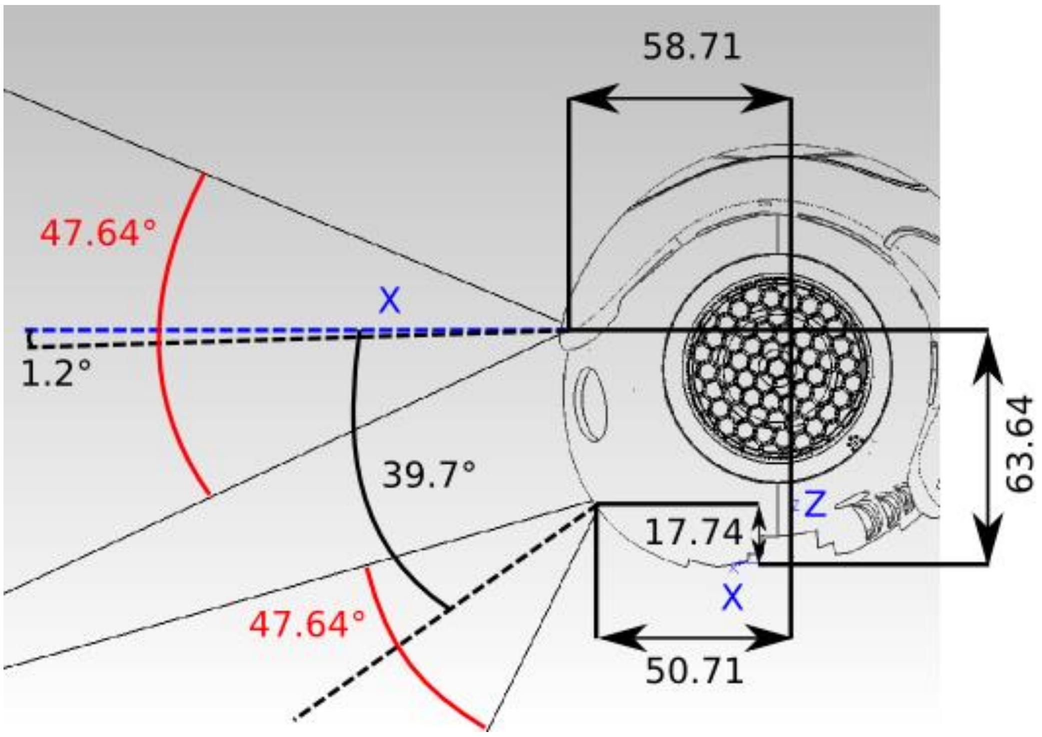


Figure 15. NAO Video camera [7]

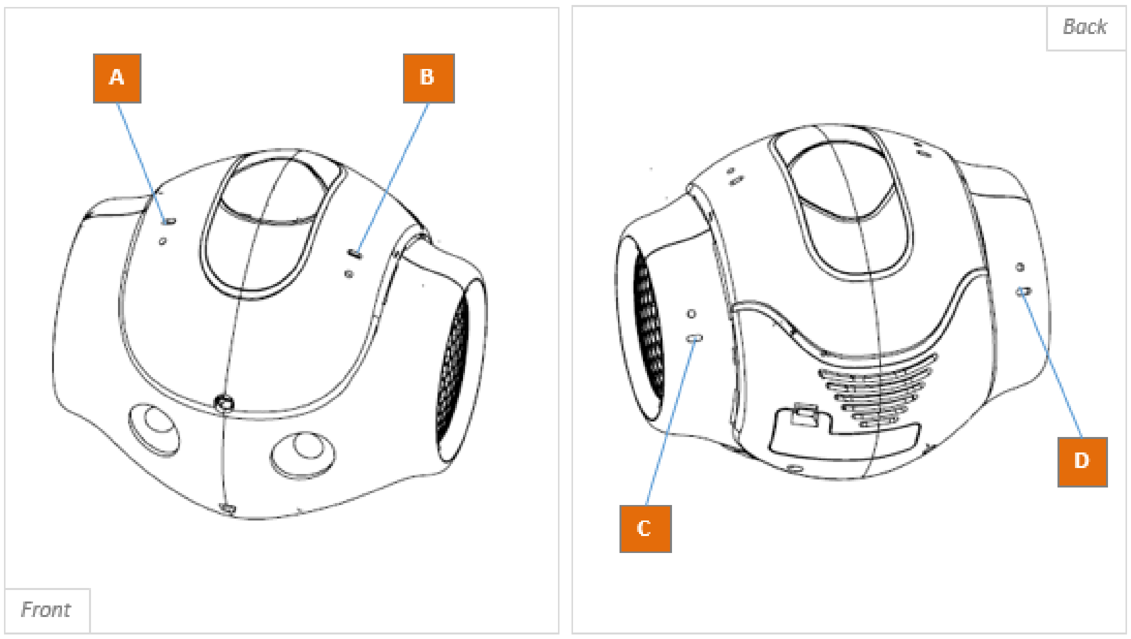


Figure 16. NAO robot Microphones [9]

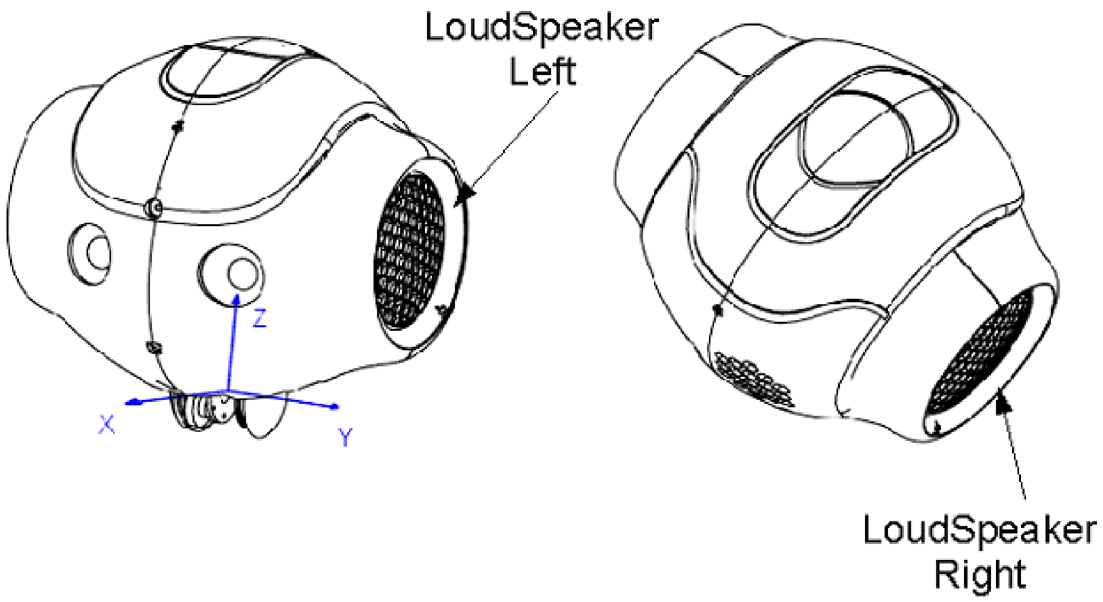


Figure 17. NAO robot Loudspeakers [10]

2.3.1 NAOqi

NAOqi framework is the programming framework developed by Aldebaran, used to program NAO.

NAOqi framework provides varieties of programming SDK including C++ and Python. In NAOqi framework, the robot communicates with the external world must through many proxies. In this thesis, ALAudioRecorder proxy and ALTextToSpeech proxy are used in order to make conversations between robots and users.

2.3.2 ALAudioRecorder

ALAudioRecorder provides recording services in “WAV” and “OGG” file format of the signals coming from the robot’s microphones.

ALAudioRecorder relies on the Linux library SNDFile to efficiently encode audio inputs in real time. ALAudioRecorder collects input signals through ALAudioDevice.

ALAudioDevice provides other NAOqi modules with access to NAO’s audio inputs (microphones) and outputs (loudspeakers). It is based on the standard Linux ALSA (Advanced Linux Sound Library) library to communicate with NAO’s sound driver and subsequently to microphones and speakers.

To process data coming from the microphones, the procedure is different. Indeed, a NAOqi module willing to process such data will first “subscribe” to ALAudioDevice and specify the format of the data that it requires (number of channels, sample rate, etc...). The data correctly formatted will then be automatically and regularly sent to the requesting module by using one of its methods as a callback.

The recording capabilities are for now limited to the following formats:

1. four channels 48000Hz in OGG.
2. four channels 48000Hz in WAV.
3. one channel (front, rear, left or right), 16000Hz, in OGG.
4. one channel (front, rear, left or right), 16000Hz, in WAV. [8]

2.3.3 ALTextToSpeech

The ALTextToSpeech module allows the robot to speak. It sends commands to a text-to-speech engine and authorizes also voice customization. The result of the synthesis is sent to the robot’s loudspeakers. [22]

3 OVERALL DESIGN

The whole structure of the system is shown below. First of all, by using Captcha library, a training dataset and a testing dataset are created and containing thousands of CAPTCHA images with certain labels as their file names. Secondly, those images data are loaded to the memory and are processed to fit the training format. After the image data and labels are prepared, a convolutional neural network is going to be established with dataset the input. Then we train our model in several rounds to enhance the accuracy and also select the best optimizer and loss functions in this model. When the model file is trained with a great performance in the test dataset, we then deploy it to NAO robot and a web service to test it recognizing capability in the real situation.

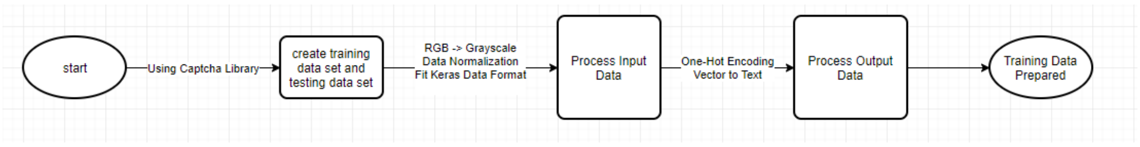


Figure 18. Data Preparation flowchart

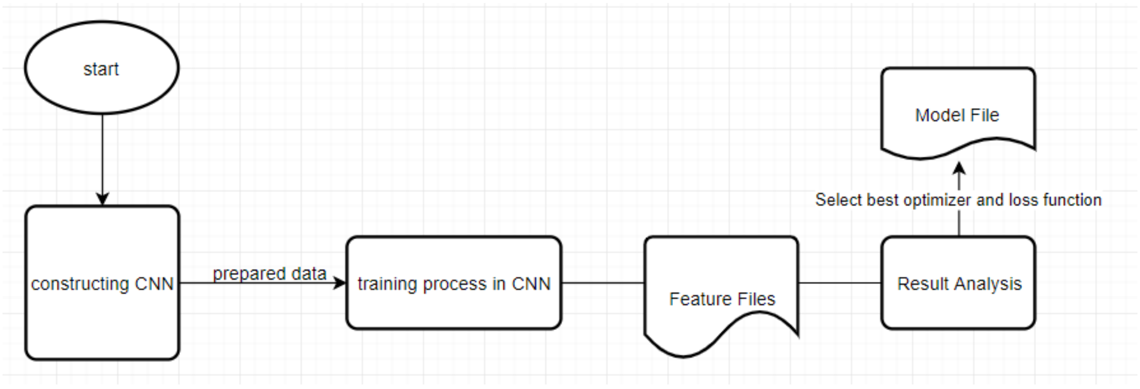


Figure 19. Training Process Flowchart

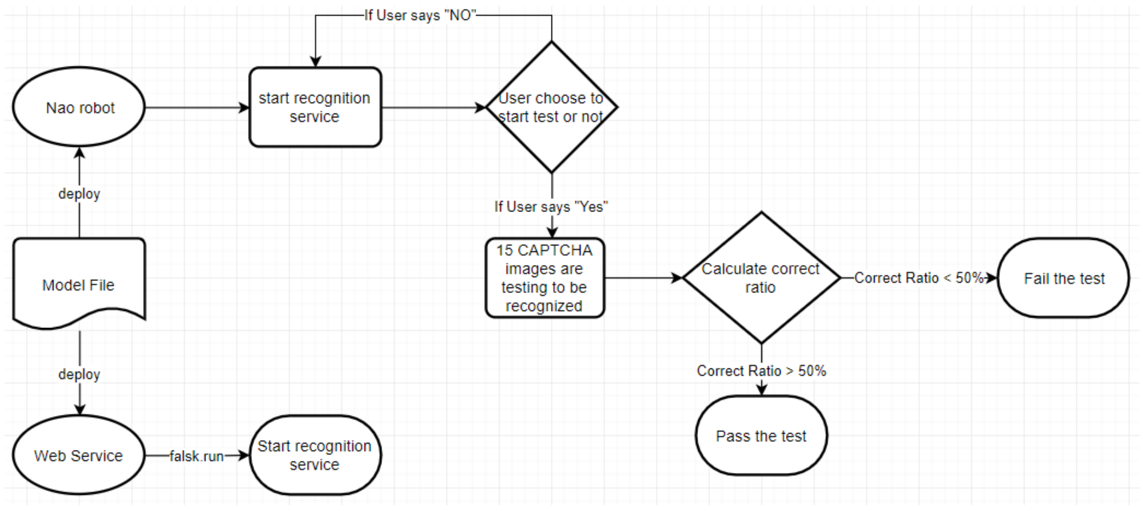


Figure 20. Deployment Flowchart

4 DATA PREPROCESSING

4.1 Create Dataset

CAPTCHA creation is very related to the generators. Different generators can create very different CAPTCHA images. Captcha library is used to create CAPTCHA images in this thesis.

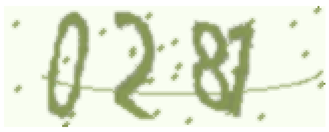
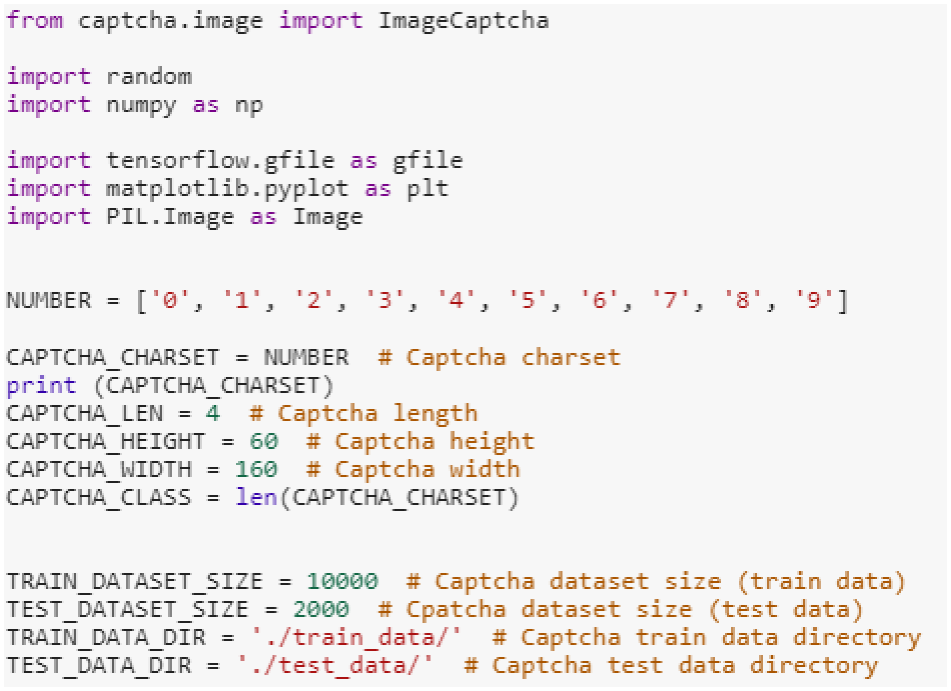


Figure 21. CAPTCHA example generated by Captcha library

We first create a charset which contains the characters we would like to insert in the CAPTCHA image. In this case, the number [0, 1, 2, …, 9] are used as the charset.

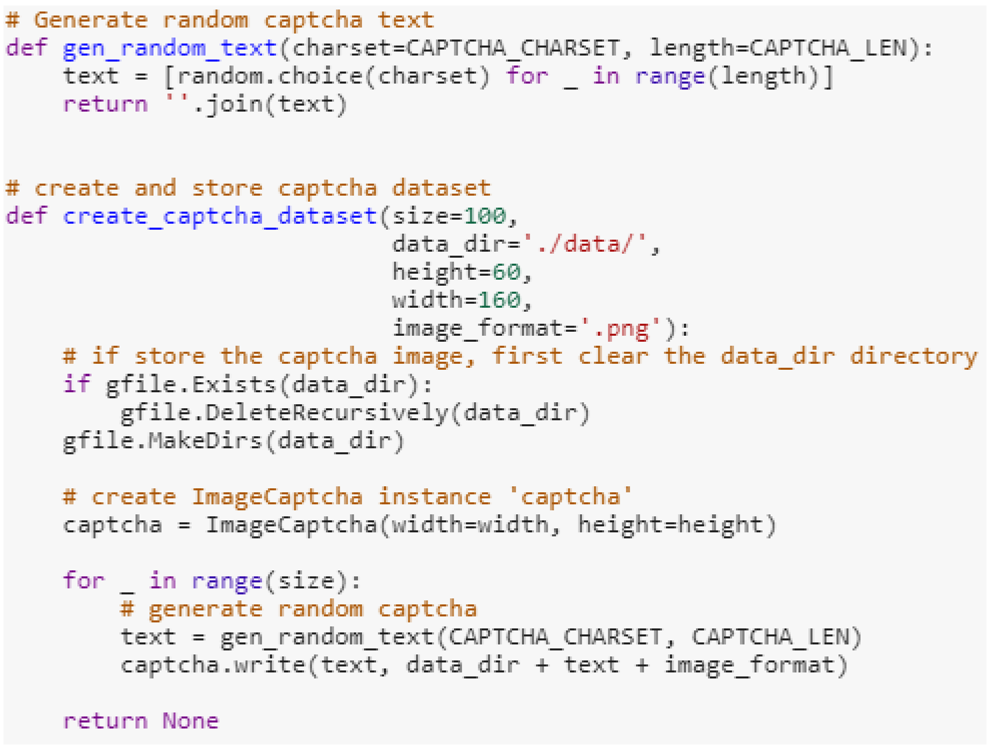


```
from captcha.image import ImageCaptcha

import random
import numpy as np

import tensorflow.gfile as gfile
import matplotlib.pyplot as plt
import PIL.Image as Image


NUMBER = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

CAPTCHA_CHARSET = NUMBER  # Captcha charset
print (CAPTCHA_CHARSET)
CAPTCHA_LEN = 4  # Captcha length
CAPTCHA_HEIGHT = 60  # Captcha height
CAPTCHA_WIDTH = 160  # Captcha width
CAPTCHA_CLASS = len(CAPTCHA_CHARSET)


TRAIN_DATASET_SIZE = 10000  # Captcha dataset size (train data)
TEST_DATASET_SIZE = 2000  # Cpatcha dataset size (test data)
TRAIN_DATA_DIR = './train_data/'  # Captcha train data directory
TEST_DATA_DIR = './test_data/'  # Captcha test data directory
```

Figure 22. CAPTCHA charset, dataset size, and data directory

The CAPTCHA image is set to 160 pixels width and 60 pixels height. Each image contains 4 characters. As training dataset needs a large amount of data to construct the recognition model, 10000 is set to the size of the training dataset and the size of the test dataset is set to 2000. However, as 0 to 9 has 10000 possible combinations within 4 characters, which is the same size of our training dataset, the final dataset must be less than 10000

CAPTCHA images because CAPTCHA images containing the same characters are not treated as different CAPTCHAs.



```python
# Generate random captcha text
def gen_random_text(charset=CAPTCHA_CHARSET, length=CAPTCHA_LEN):
    text = [random.choice(charset) for _ in range(length)]
    return ''.join(text)


# create and store captcha dataset
def create_captcha_dataset(size=100,
                           data_dir='./data/',
                           height=60,
                           width=160,
                           image_format='.png'):
    # if store the captcha image, first clear the data_dir directory
    if gfile.Exists(data_dir):
        gfile.DeleteRecursively(data_dir)
    gfile.MakeDirs(data_dir)

    # create ImageCaptcha instance 'captcha'
    captcha = ImageCaptcha(width=width, height=height)

    for _ in range(size):
        # generate random captcha
        text = gen_random_text(CAPTCHA_CHARSET, CAPTCHA_LEN)
        captcha.write(text, data_dir + text + image_format)

    return None
```

Figure 23. CAPTCHA generating methods

The figure above are the methods that generate CAPTCHA dataset. The first method aims to create a random combination of four-character string, which needs charset and length of the text as the arguments. The second methods need five arguments: the size of the dataset, the directory of the dataset, the height, width and the image format of the CAPTCHA image. We use the provided height, width and image format arguments to define our CAPTCHA image format, then create a four-character string in random combination. The string is used as the file name for each CAPTCHA containing that certain text. The label for each CAPTCHA is the file name without suffix (.png). Figure below shows how dataset looks like.

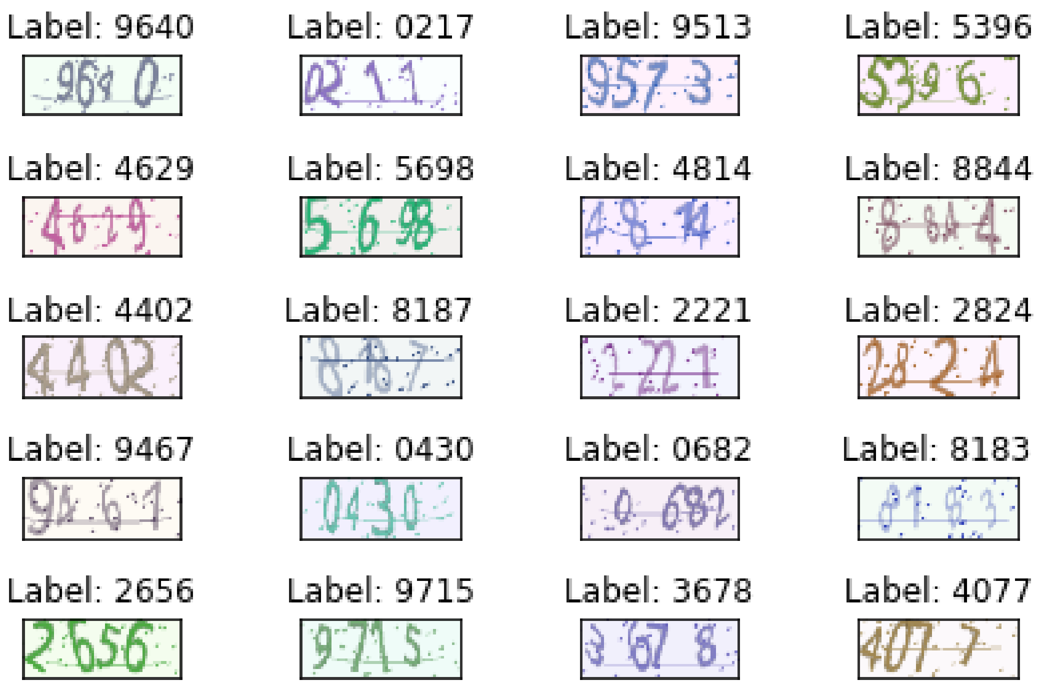


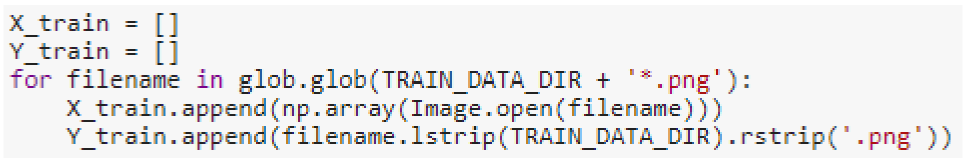
Figure 24. CAPTCHA raw images with labels

4.2 Process Input Data

4.2.1 RGB Format to Grayscale

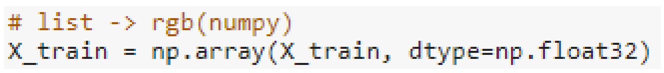
As our CAPTCHA image is RGB (Red Green Blue) format, each image contains many color information. In order to accelerate the training process, we need to reduce the amount of data contained in each CAPTCHA image. The color information is not so necessary, so we convert our RGB image to grayscale, which will reduce the number of color channels from three to one.

Before converting color space, we need first load our dataset and convert image to numpy array which Keras only accepts input data in this format.



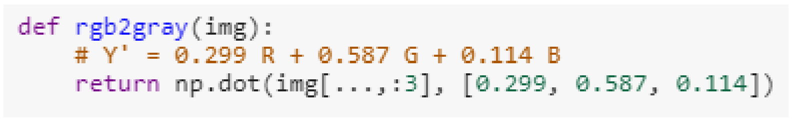
```
X_train = []
Y_train = []
for filename in glob.glob(TRAIN_DATA_DIR + '*.png'):
    X_train.append(np.array(Image.open(filename)))
    Y_train.append(filename.lstrip(TRAIN_DATA_DIR).rstrip('.png'))
```

Figure 25. load training and testing dataset



```
# list -> rgb(numpy)
X_train = np.array(X_train, dtype=np.float32)
```

Figure 26. Convert image to numpy array



```
def rgb2gray(img):
    # Y' = 0.299 R + 0.587 G + 0.114 B
    return np.dot(img[...,:3], [0.299, 0.587, 0.114])
```

Figure 27. RGB to grayscale method

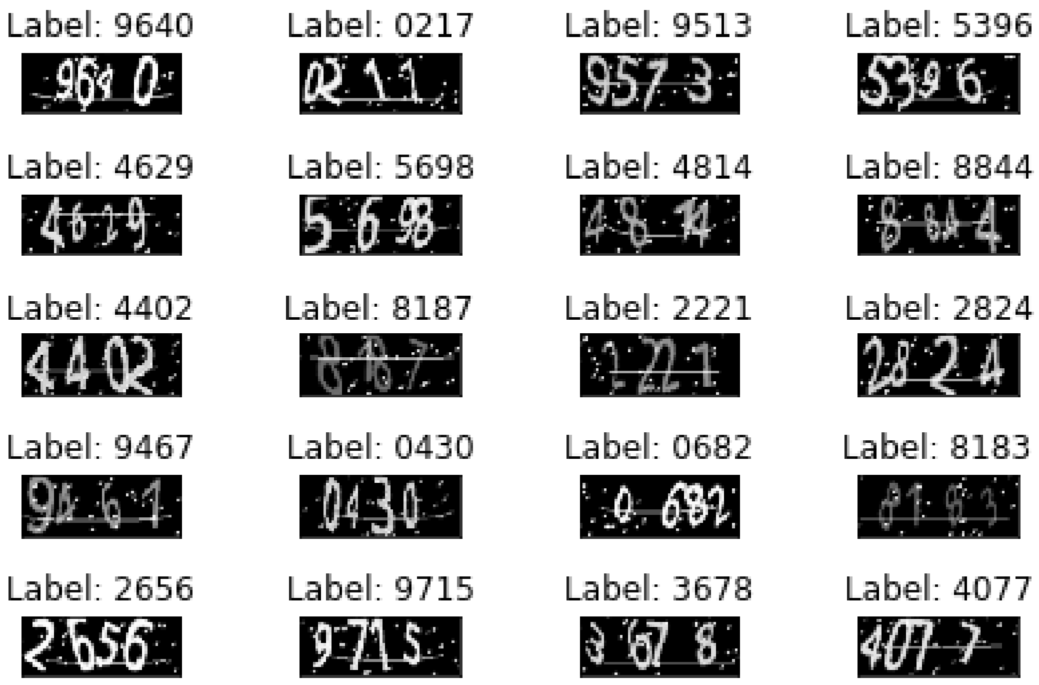
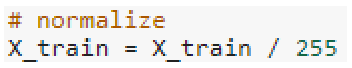


Figure 28. Grayscale CAPTCHA image

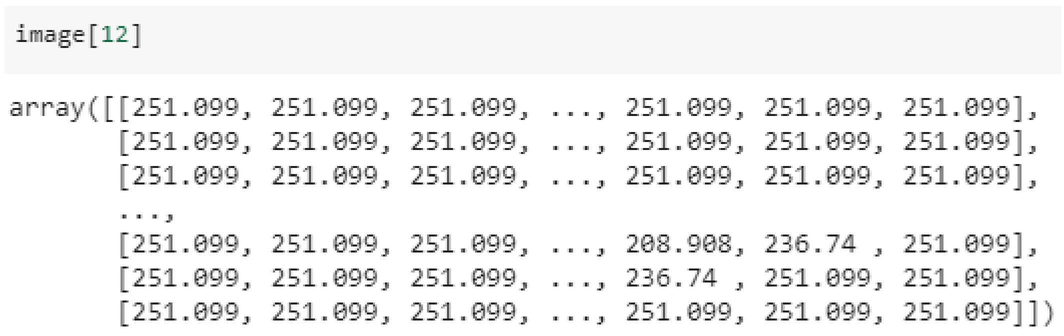
4.2.2 Data Normalization

Data Normalization is to scale the data so that it falls into a small specific interval. Normalizing the data for each batch can improve the generalization of the model and prevent overfitting. It can also improve the convergence speed of iterative solutions.



```
# normalize
X_train = X_train / 255
```

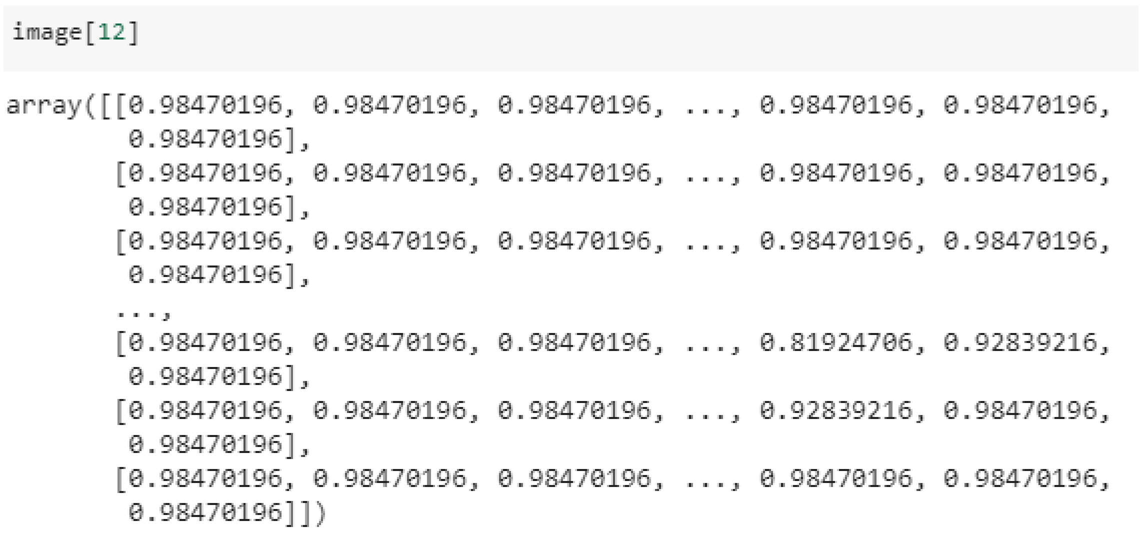
Figure 29. Data Normalization process



```
image[12]
```

```
array([[251.099, 251.099, 251.099, ..., 251.099, 251.099, 251.099],
       [251.099, 251.099, 251.099, ..., 251.099, 251.099, 251.099],
       [251.099, 251.099, 251.099, ..., 251.099, 251.099, 251.099],
       ...,
       [251.099, 251.099, 251.099, ..., 208.908, 236.74 , 251.099],
       [251.099, 251.099, 251.099, ..., 236.74 , 251.099, 251.099],
       [251.099, 251.099, 251.099, ..., 251.099, 251.099, 251.099]])
```

Figure 30. Before normalization



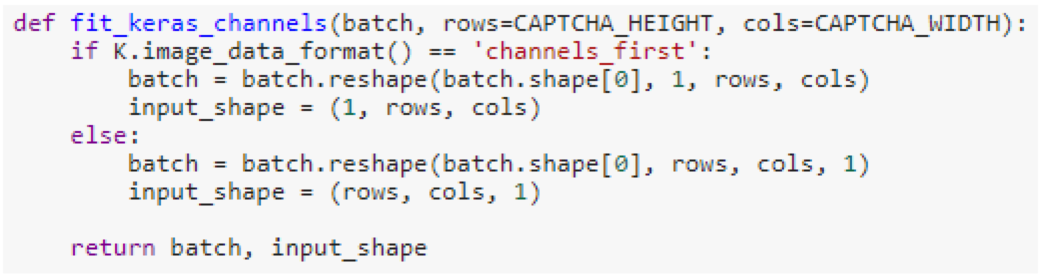
```
image[12]
```

```
array([[0.98470196, 0.98470196, 0.98470196, ..., 0.98470196, 0.98470196,
        0.98470196],
       [0.98470196, 0.98470196, 0.98470196, ..., 0.98470196, 0.98470196,
        0.98470196],
       [0.98470196, 0.98470196, 0.98470196, ..., 0.98470196, 0.98470196,
        0.98470196],
       ...,
       [0.98470196, 0.98470196, 0.98470196, ..., 0.81924706, 0.92839216,
        0.98470196],
       [0.98470196, 0.98470196, 0.98470196, ..., 0.92839216, 0.98470196,
        0.98470196],
       [0.98470196, 0.98470196, 0.98470196, ..., 0.98470196, 0.98470196,
        0.98470196]])
```

Figure 31. After normalization

4.2.3 Fit Keras Channels

Keras provides two types of saving data format – “channels-first” and “channels-last”. The default way is “channels first”, it puts the position of channels value before the image height and width in the vector. On the other hand, “channels-last” put the value of the channel at the end position instead. Here we define a method to make our values fit Keras channels format.



```python
def fit_keras_channels(batch, rows=CAPTCHA_HEIGHT, cols=CAPTCHA_WIDTH):
    if K.image_data_format() == 'channels_first':
        batch = batch.reshape(batch.shape[0], 1, rows, cols)
        input_shape = (1, rows, cols)
    else:
        batch = batch.reshape(batch.shape[0], rows, cols, 1)
        input_shape = (rows, cols, 1)

    return batch, input_shape
```

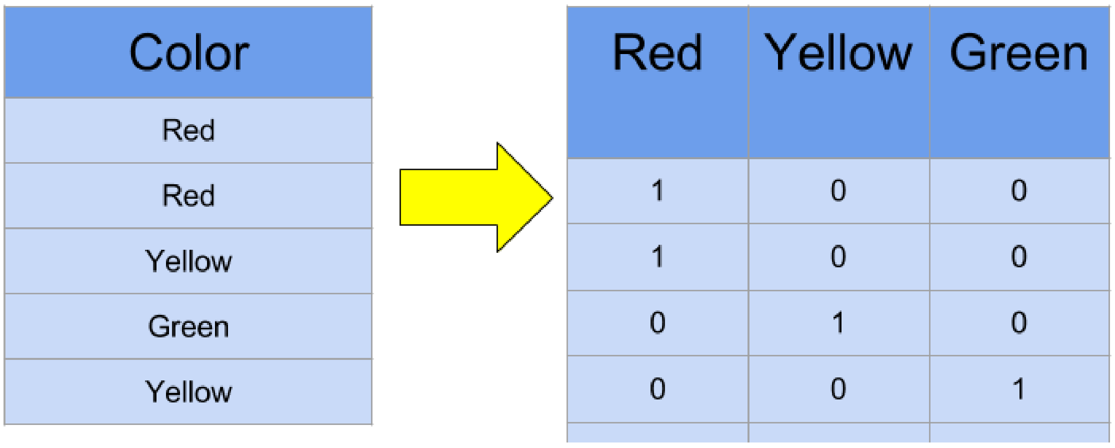
Figure 32. Method of fit Keras channels

Batch number is the number of images that be processed together when training the model.

4.3 Process Output Data

4.3.1 One-hot Encoding

One-hot encoding, also known as one-bit efficient encoding, uses an N-bit status register to encode N states, each state is independent of its register bits, and at any time, only One is valid. It can be understood that for each feature if it has m possible values, it is mono-thermally and becomes m binary features. Also, these features are mutually exclusive, with only one activation at a time. Therefore, the data becomes sparse.

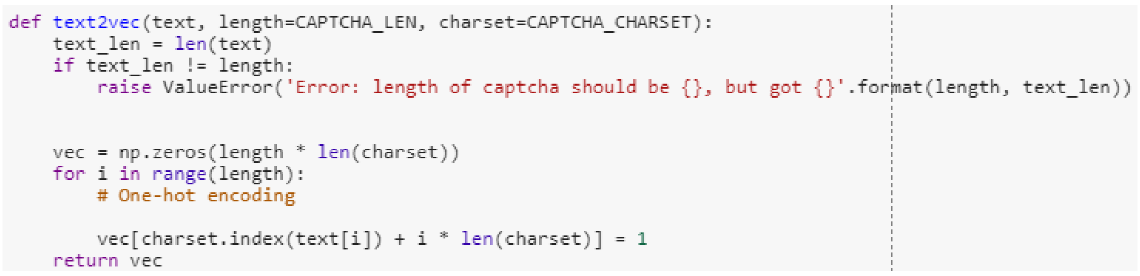


Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

Figure 33. example of one-hot encoding [11]

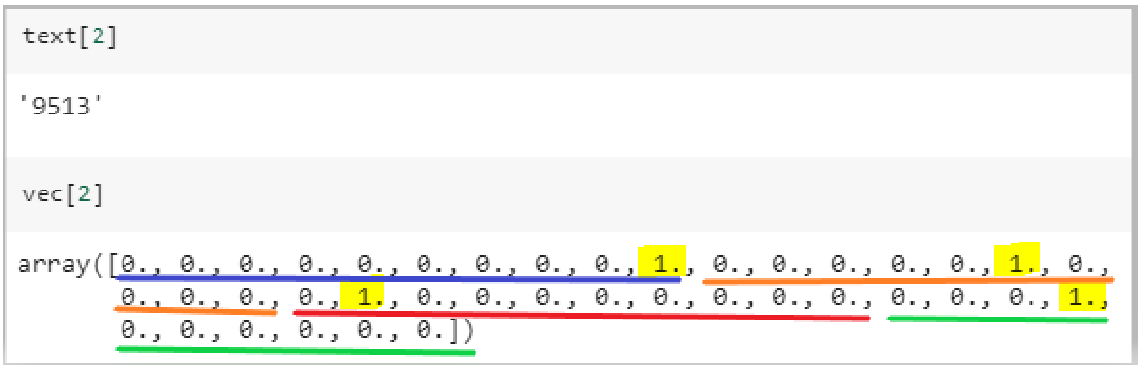
We are encoding our labels with this encoding system. After one-hot encoding, the text value will transmit to vector, then use the method from NumPy library to convert this vector to NumPy array fit the Keras format. Besides, transmitting our label text to vector can also accelerate our training process.



```python
def text2vec(text, length=CAPTCHA_LEN, charset=CAPTCHA_CHARSET):
    text_len = len(text)
    if text_len != length:
        raise ValueError('Error: length of captcha should be {}, but got {}'.format(length, text_len))


    vec = np.zeros(length * len(charset))
    for i in range(length):
        # One-hot encoding

        vec[charset.index(text[i]) + i * len(charset)] = 1
    return vec
```

Figure 34. Method of transmitting label text to one-hot vector

we create an array of length 10 four times during the loop, the number [0 - 9] occupy one position of the array based on one-hot encoding. After the looping, we combine these 4 arrays into one 40 length array. The figure below is an example. The label “9513” transmitted to an array with 40 lengths after one-hot encoding.

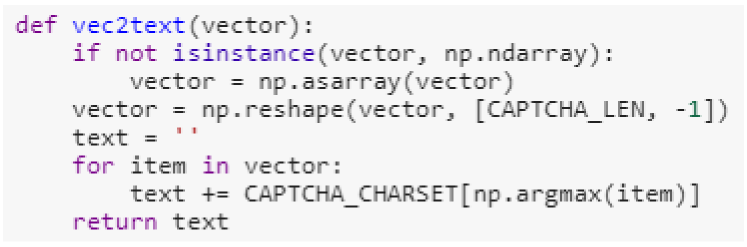


```
text[2]

'9513'

vec[2]

array([0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0.,
       0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
       0., 0., 0., 0., 0., 0.])
```

Figure 35. "9513" encoded by one-hot encoding

4.3.2 Decoding Output Vector to Text

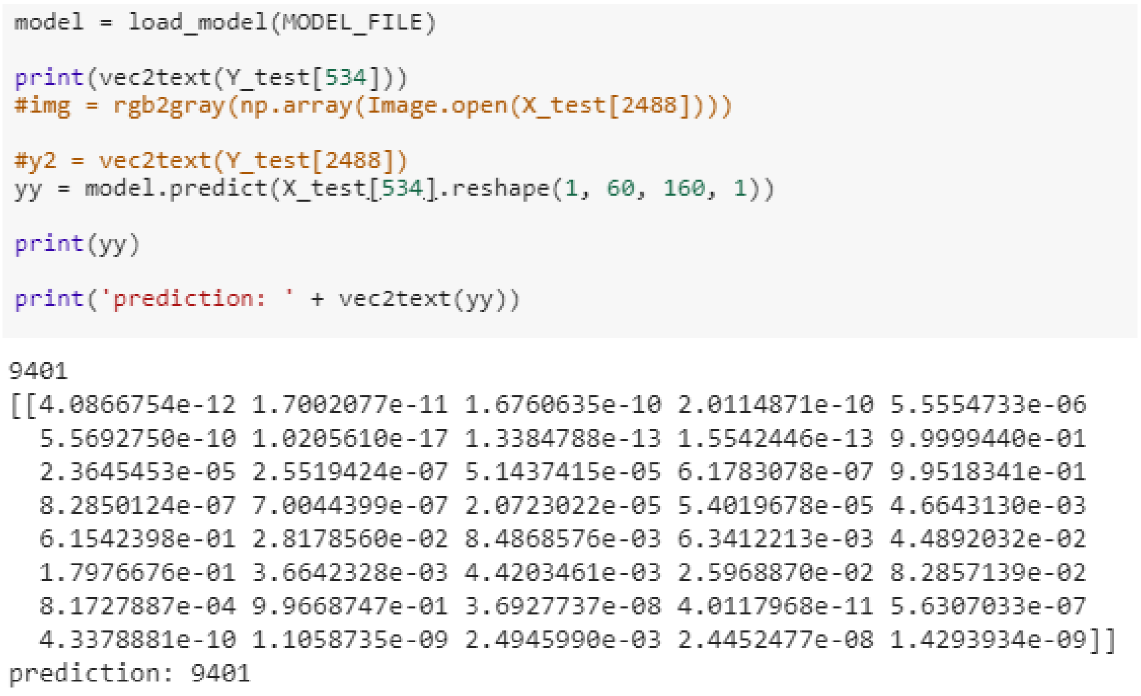
The predicted value our model outputs is an array containing 40 probability values. This cannot be recognized so we should translate them to a four-character text. Ten values make a group, and the highest probability value will be selected as the representative value of these ten. 4 groups have 4 output values, combining these 4 values and converting them to text would consist our prediction.



```python
def vec2text(vector):
    if not isinstance(vector, np.ndarray):
        vector = np.asarray(vector)
    vector = np.reshape(vector, [CAPTCHA_LEN, -1])
    text = ''
    for item in vector:
        text += CAPTCHA_CHARSET[np.argmax(item)]
    return text
```

Figure 36. Method of transmitting vector to text

We first convert this vector to a NumPy array, then use argmax() method to translate vector value to text.



```python
model = load_model(MODEL_FILE)

print(vec2text(Y_test[534]))
#img = rgb2gray(np.array(Image.open(X_test[2488])))

#y2 = vec2text(Y_test[2488])
yy = model.predict(X_test[534].reshape(1, 60, 160, 1))

print(yy)

print('prediction: ' + vec2text(yy))
```

```
9401
[[4.0866754e-12 1.7002077e-11 1.6760635e-10 2.0114871e-10 5.5554733e-06
  5.5692750e-10 1.0205610e-17 1.3384788e-13 1.5542446e-13 9.9999440e-01
  2.3645453e-05 2.5519424e-07 5.1437415e-05 6.1783078e-07 9.9518341e-01
  8.2850124e-07 7.0044399e-07 2.0723022e-05 5.4019678e-05 4.6643130e-03
  6.1542398e-01 2.8178560e-02 8.4868576e-03 6.3412213e-03 4.4892032e-02
  1.7976676e-01 3.6642328e-03 4.4203461e-03 2.5968870e-02 8.2857139e-02
  8.1727887e-04 9.9668747e-01 3.6927737e-08 4.0117968e-11 5.6307033e-07
  4.3378881e-10 1.1058735e-09 2.4945990e-03 2.4452477e-08 1.4293934e-09]]
prediction: 9401
```

Figure 37. Example of vector to text

Here is an example of the translating process. The predicted value of the model is an array with 40 probability values, we choose the highest value in every ten items, translate them to text and combine them to a four-character prediction value.

5 MODEL STRUCTURE DESIGN

5.1 AlexNet and VGG-16 Model

In 2012, Alex Krizhevsky and Ilya Sutskever designed a deep convolutional neural network AlexNet at the University of Toronto's Geoff Hinton lab, winning the 2012 ImageNet LSVRC championship with an accuracy rate far exceeding the second place, which caused a lot of sensation. AlexNet can be said to be a historical network structure. Before that, deep learning has been silent for a long time. Since the birth of AlexNet in 2012, the latter ImageNet champions have been done with the convolutional neural network (CNN), which makes CNN become the core algorithm model in image recognition classification, which brings about the explosion of deep learning.

The success of AlexNet is related to the characteristics of this model design. AlexNet has 3 main features:

1. A nonlinear activation function is used: ReLU
2. Methods to prevent overfitting: Dropout, Data augmentation
3. Other: Multi-GPU implementation, use of LRN normalization layer

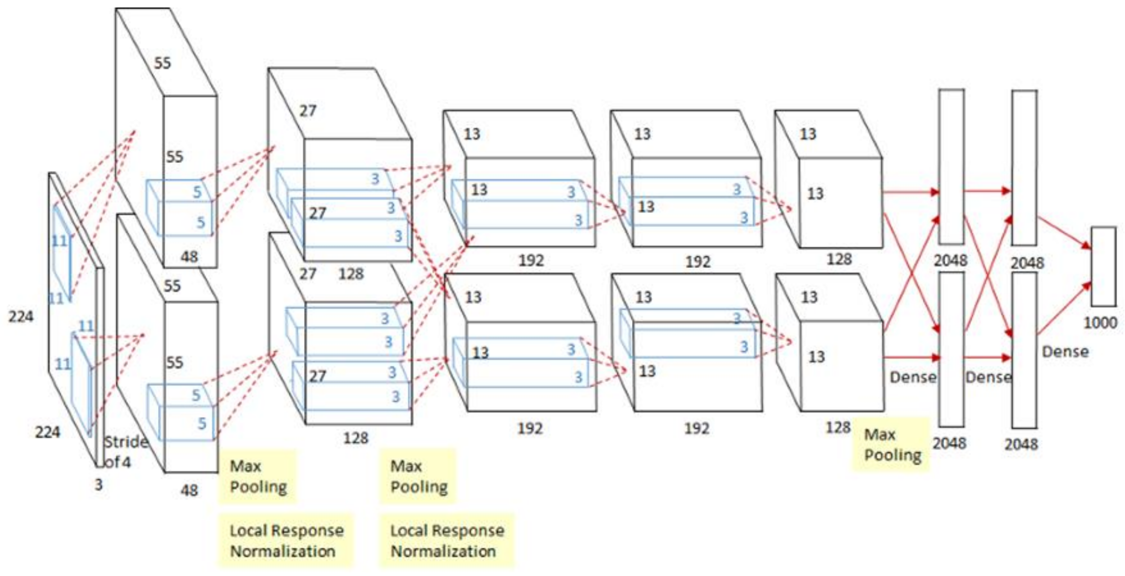


Figure 38. AlexNet Structure [12]

In 2014, the University of Oxford's Visual Geometry Group and Google DeepMind researchers developed a new deep convolutional neural network: VGGNet and won the second price in the ILSVRC2014 competition classification project. VGGNet explores the relationship between the depth of convolutional neural networks and its performance and successfully constructs a 16-to-19-layer deep convolutional neural network, which proves that increasing the depth of the network can affect the final performance of the network to a certain extent, and drastically reduce the error rate. Besides, the scalability is very strong, and the generalization of migrating to other image data is also very good. So far, VGG is still used to extract image features.

VGGNet can be seen as a deeper version of AlexNet, which also consists of two parts: the convolutional layer and the fully connected layer.

Small convolution kernel is an important feature of VGG model. VGG uses a convolutional layer of multiple smaller convolution kernels (3x3) instead of a convolutional layer with a larger convolution kernel. On the one hand, it can reduce parameters, on the other hand, it is equivalent to more nonlinear mapping, which can increase the network fitting/expression capabilities. [13]

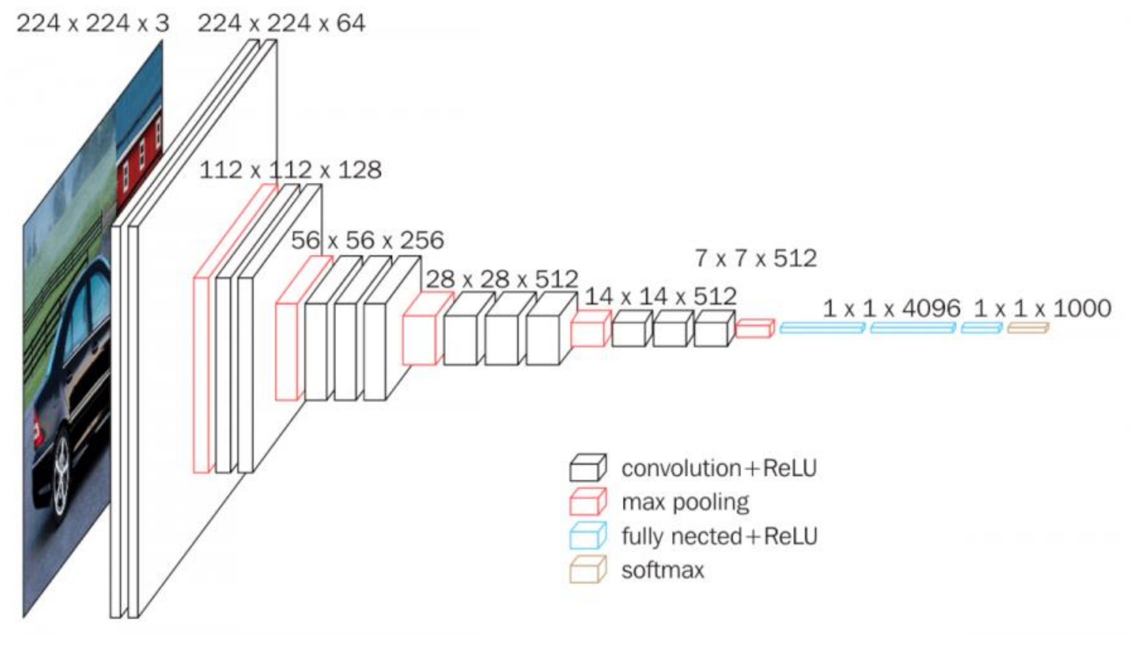


Figure 39. VGG-16 model [13]

5.2 Construct Model

In this thesis, VGG-16 model is used to create our convolutional neural network. The network consists of 3 convolutional layers, 3 pooling layers,1 dropout layer, and 2 full connected layers. The last full connected layer classifies the output into 10 classes for 4 times, and each time the maximum probability is the output value. Then we splice 4 classification results together to get the final result.

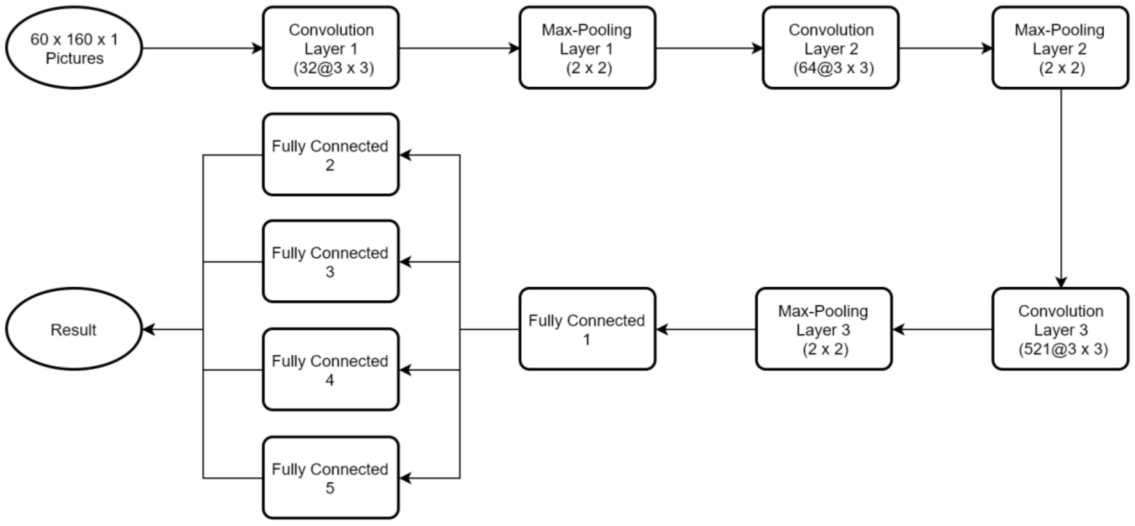

60 x 160 x 1 Pictures
Convolution Layer 1 (32@3 x 3)
Max-Pooling Layer 1 (2 x 2)
Convolution Layer 2 (64@3 x 3)
Max-Pooling Layer 2 (2 x 2)
Convolution Layer 3 (521@3 x 3)
Max-Pooling Layer 3 (2 x 2)
Fully Connected 1
Fully Connected 2
Fully Connected 3
Fully Connected 4
Fully Connected 5
Result

Figure 40. Flow chart of the CNN

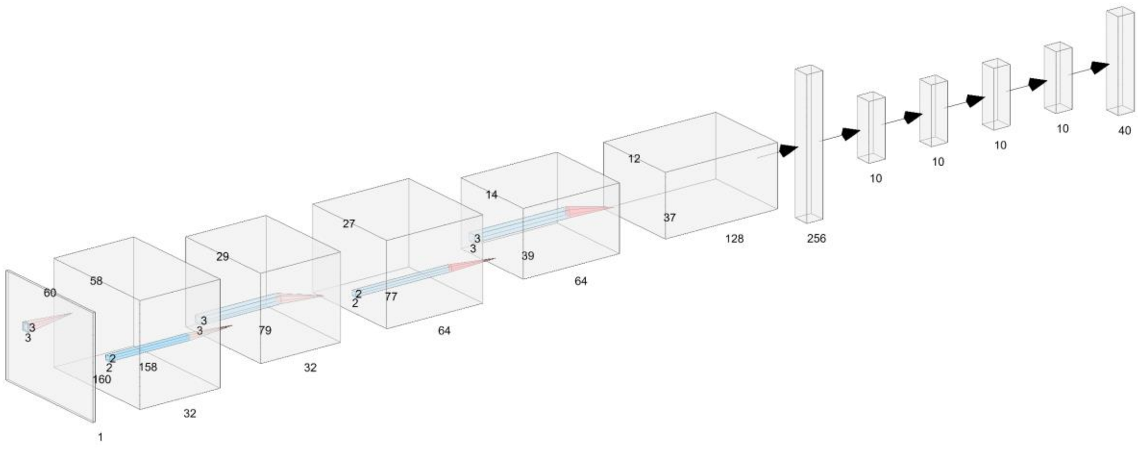
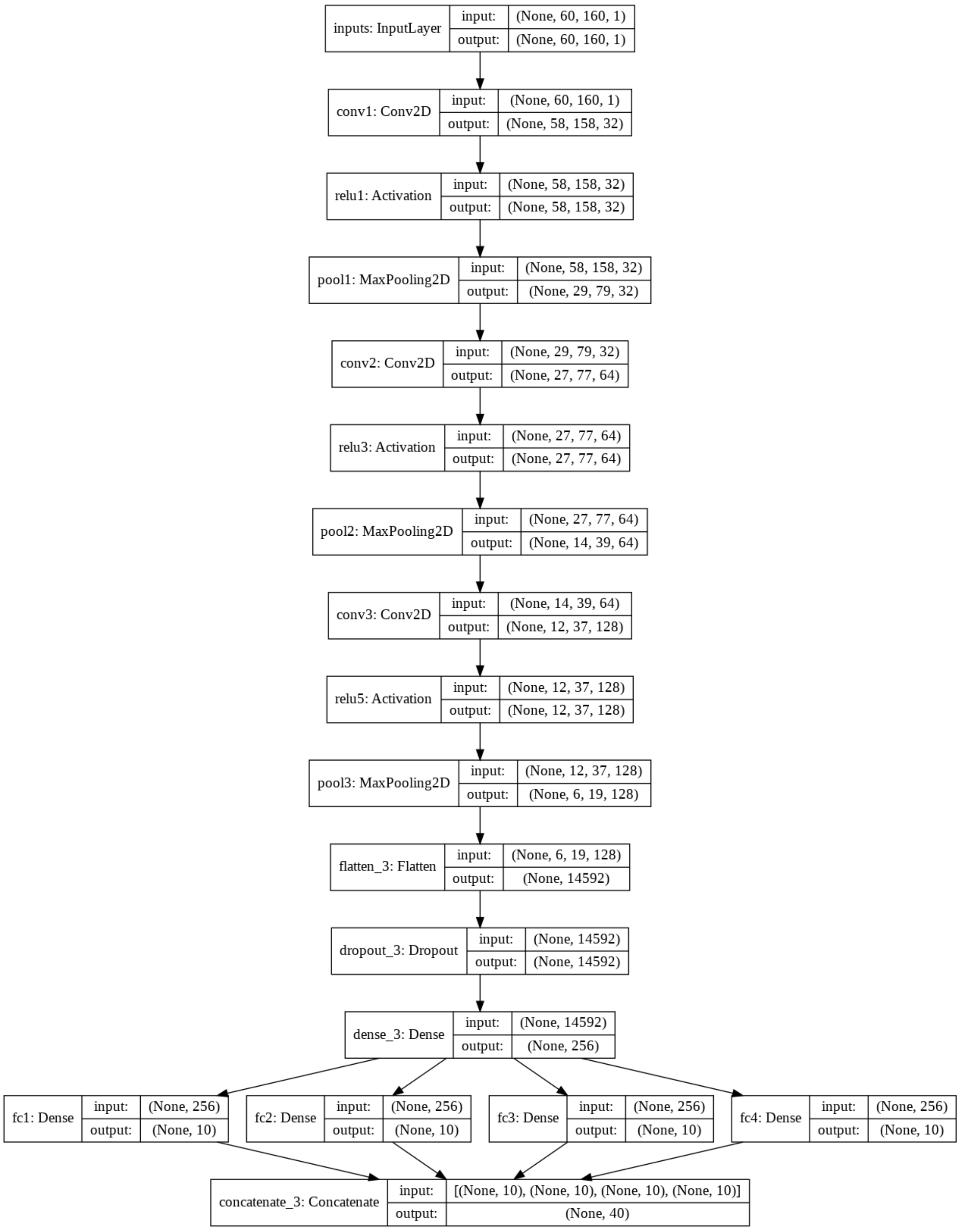


Figure 41. CNN model in this project

The output is an array containing each value’s probability, we then use vector to text method which introduced above to decoding the value into predicted text.



inputs: InputLayer | input: (None, 60, 160, 1) | output: (None, 60, 160, 1)
conv1: Conv2D | input: (None, 60, 160, 1) | output: (None, 58, 158, 32)
relu1: Activation | input: (None, 58, 158, 32) | output: (None, 58, 158, 32)
pool1: MaxPooling2D | input: (None, 58, 158, 32) | output: (None, 29, 79, 32)
conv2: Conv2D | input: (None, 29, 79, 32) | output: (None, 27, 77, 64)
relu3: Activation | input: (None, 27, 77, 64) | output: (None, 27, 77, 64)
pool2: MaxPooling2D | input: (None, 27, 77, 64) | output: (None, 14, 39, 64)
conv3: Conv2D | input: (None, 14, 39, 64) | output: (None, 12, 37, 128)
relu5: Activation | input: (None, 12, 37, 128) | output: (None, 12, 37, 128)
pool3: MaxPooling2D | input: (None, 12, 37, 128) | output: (None, 6, 19, 128)
flatten_3: Flatten | input: (None, 6, 19, 128) | output: (None, 14592)
dropout_3: Dropout | input: (None, 14592) | output: (None, 14592)
dense_3: Dense | input: (None, 14592) | output: (None, 256)
fc1: Dense | input: (None, 256) | output: (None, 10)
fc2: Dense | input: (None, 256) | output: (None, 10)
fc3: Dense | input: (None, 256) | output: (None, 10)
fc4: Dense | input: (None, 256) | output: (None, 10)
concatenate_3: Concatenate | input: [(None, 10), (None, 10), (None, 10), (None, 10)] | output: (None, 40)

Figure 42. The whole structure of the CNN

6 MODEL TRAINING PROCESS

After we establish our CNN model, it's time to train our model with pre-defined batches and epochs.

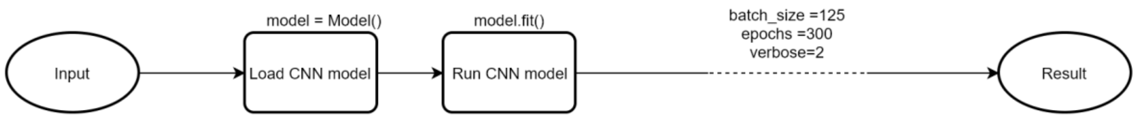
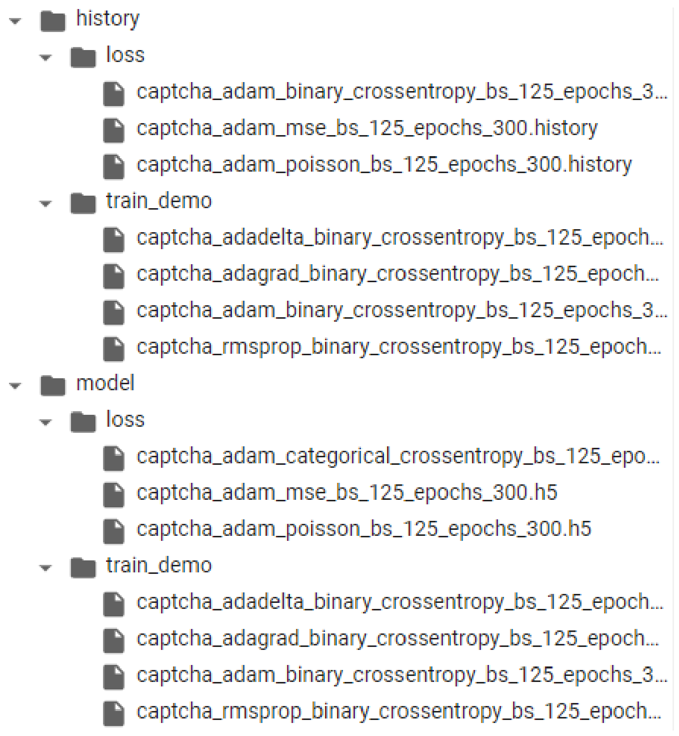


Figure 43. Flow chart of the training process

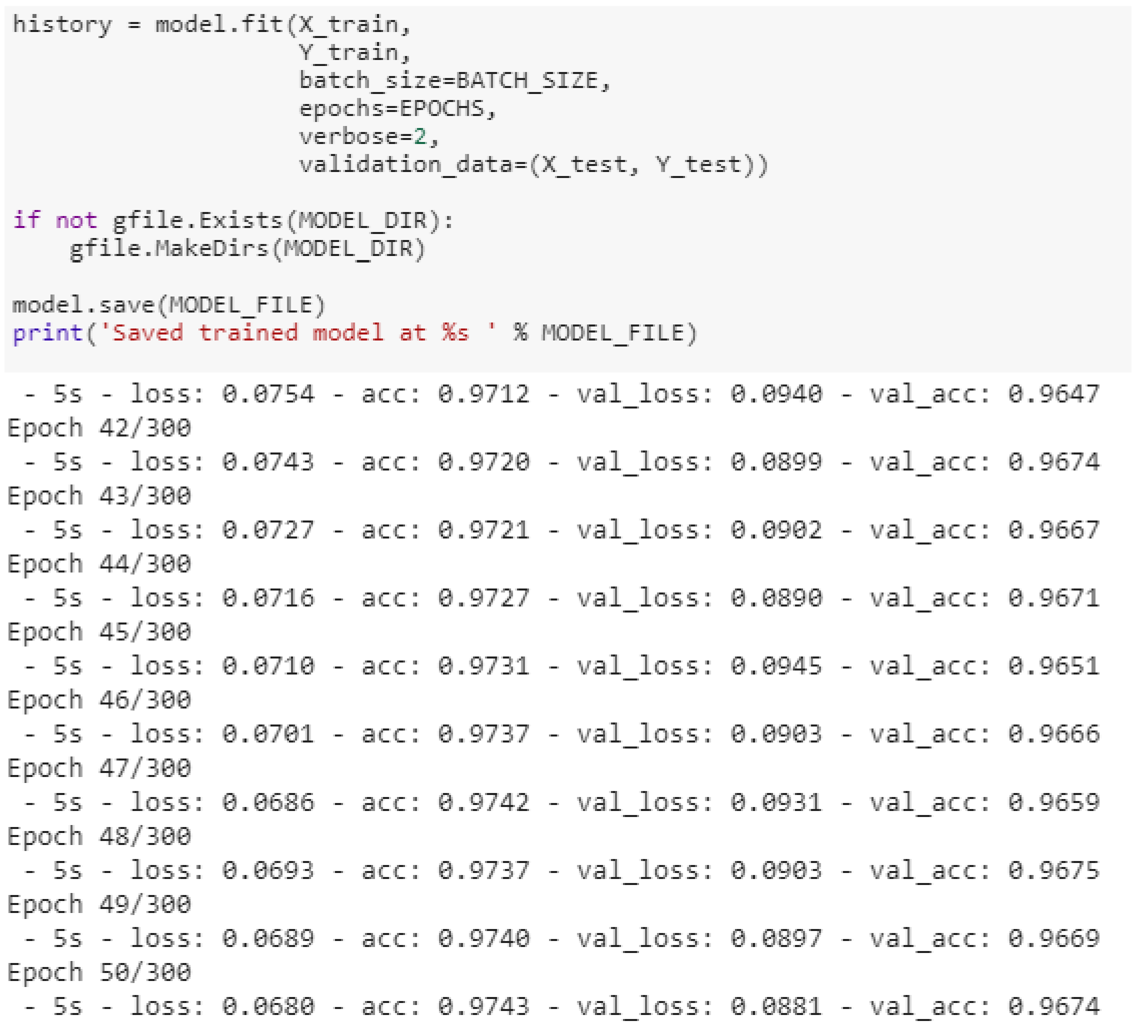
In order to select the best optimizer and loss function in this model, 7 combinations are choosing to be trained. Each model saved as a .h5 file, and the corresponding log is also created along with the model file and saved in history folder with the .history suffix.



```
history
    loss
        captcha_adam_binary_crossentropy_bs_125_epochs_3...
        captcha_adam_mse_bs_125_epochs_300.history
        captcha_adam_poisson_bs_125_epochs_300.history
    train_demo
        captcha_adadelta_binary_crossentropy_bs_125_epoch...
        captcha_adagrad_binary_crossentropy_bs_125_epoch...
        captcha_adam_binary_crossentropy_bs_125_epochs_3...
        captcha_rmsprop_binary_crossentropy_bs_125_epoch...
model
    loss
        captcha_adam_categorical_crossentropy_bs_125_epo...
        captcha_adam_mse_bs_125_epochs_300.h5
        captcha_adam_poisson_bs_125_epochs_300.h5
    train_demo
        captcha_adadelta_binary_crossentropy_bs_125_epoch...
        captcha_adagrad_binary_crossentropy_bs_125_epoch...
        captcha_adam_binary_crossentropy_bs_125_epochs_3...
        captcha_rmsprop_binary_crossentropy_bs_125_epoch...
```

Figure 44. Model and history files

We use model.fit() method (provided by Keras library) to start training our model.



```
history = model.fit(X_train,
                    Y_train,
                    batch_size=BATCH_SIZE,
                    epochs=EPOCHS,
                    verbose=2,
                    validation_data=(X_test, Y_test))

if not gfile.Exists(MODEL_DIR):
    gfile.MakeDirs(MODEL_DIR)

model.save(MODEL_FILE)
print('Saved trained model at %s ' % MODEL_FILE)
```

```
 - 5s - loss: 0.0754 - acc: 0.9712 - val_loss: 0.0940 - val_acc: 0.9647
Epoch 42/300
 - 5s - loss: 0.0743 - acc: 0.9720 - val_loss: 0.0899 - val_acc: 0.9674
Epoch 43/300
 - 5s - loss: 0.0727 - acc: 0.9721 - val_loss: 0.0902 - val_acc: 0.9667
Epoch 44/300
 - 5s - loss: 0.0716 - acc: 0.9727 - val_loss: 0.0890 - val_acc: 0.9671
Epoch 45/300
 - 5s - loss: 0.0710 - acc: 0.9731 - val_loss: 0.0945 - val_acc: 0.9651
Epoch 46/300
 - 5s - loss: 0.0701 - acc: 0.9737 - val_loss: 0.0903 - val_acc: 0.9666
Epoch 47/300
 - 5s - loss: 0.0686 - acc: 0.9742 - val_loss: 0.0931 - val_acc: 0.9659
Epoch 48/300
 - 5s - loss: 0.0693 - acc: 0.9737 - val_loss: 0.0903 - val_acc: 0.9675
Epoch 49/300
 - 5s - loss: 0.0689 - acc: 0.9740 - val_loss: 0.0897 - val_acc: 0.9669
Epoch 50/300
 - 5s - loss: 0.0680 - acc: 0.9743 - val_loss: 0.0881 - val_acc: 0.9674
```

Figure 45. Training process

X_train, Y_train are the images and labels of the training dataset. batch_size, which has been explained above, is the number of images that be processed together. Epochs value is the number of training rounds. After trying several times, 300 rounds are great enough that our model can reach a high accuracy in test dataset within these rounds of processes. Verbose control the log information. 0 means that the log information is not output via the standard output stream, 1 is represented as the output progress bar record, and 2 is represented as the output row record for each epoch result analysis. Validation_data makes model calculate the accuracy and the loss in test dataset in order to overcome over-fitting problem that our model is suitable for more general application scenarios than just performing well on training sets.

7 RESULT ANALYSIS

7.1 Loss Function

The loss function is used to estimate the degree of inconsistency between the predicted value f(x) of your model and the true value Y. It is a non-negative real-valued function, usually expressed by $L(Y, f(x))$. The smaller the loss function, the better the robustness of the model. The loss function is the core part of the empirical risk function and an important part of the structural risk function. The structural risk function of the model includes empirical risk terms and regular terms, which can usually be expressed as follows:

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L\big(y_i, f(x_i; \theta)\big) + \lambda\, \Phi(\theta)$$

Among them, the previous mean function represents the empirical risk function, L represents the loss function, and the latter Φ is a regularizer or a penalty term, which can be L1 or L2, or Other regular functions. The whole expression means to find the value of θ when the objective function is minimized.

7.1.1 Mean Squared Error

Mean Square Error (MSE) is the most commonly used regression loss function. The calculation method is to find the sum of the squares of the distance between the predicted value and the true value. The formula is shown below:

$$\text{MSE} = \sum_{i=1}^{n} (y_i - y_i^p)^2$$

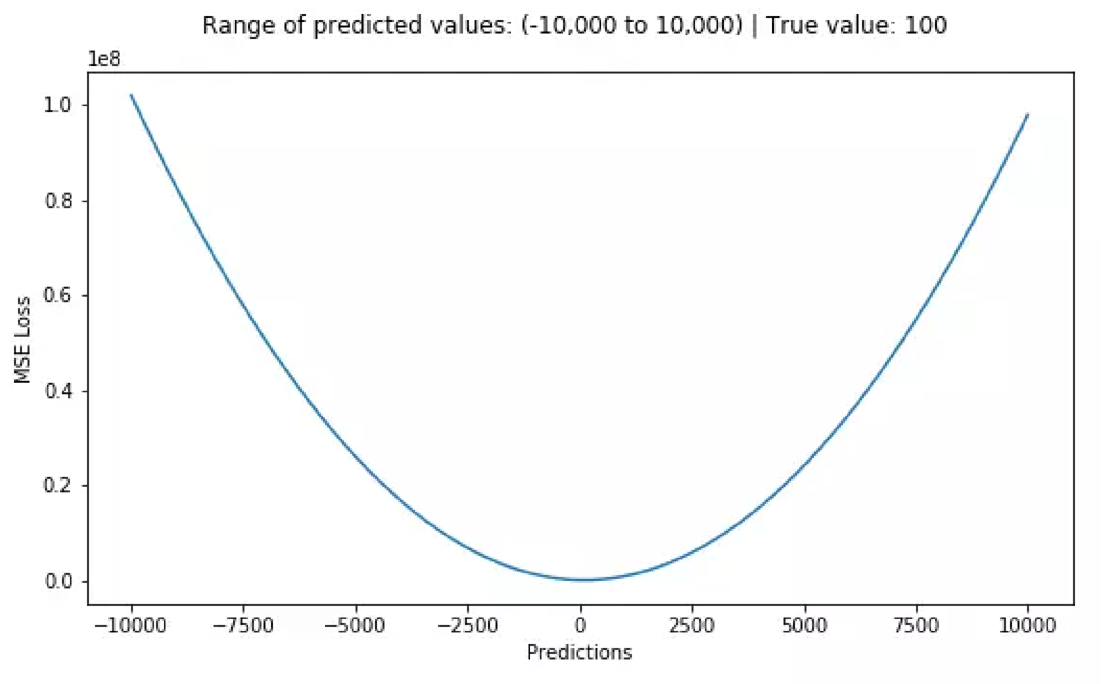


Figure 46. MSE graph [15]

7.1.2 Poisson Regression

The Poisson loss function is a measure of how the predicted distribution diverges from the expected distribution, the Poisson as loss function is a variant from Poisson Distribution, where the Poisson distribution is widely used for modeling count data. It can be shown to be the limiting distribution for a normal approximation to a binomial where the number of trials goes to infinity and the probability goes to zero and both happen at such a rate that np is equal to some mean frequency for the process. [16]

$$\mathrm{L} \;=\; \frac{1}{n}\sum_{i-1}^{n}(y_{pred}^{(i)} \;-\; y_{true}^{i} \;\cdot\; \log(y_{pred}^{(i)}))$$

7.1.3 Cross Entropy

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

$$\mathrm{L} = -\frac{1}{\mathrm{n}}\sum_{\mathrm{i}=1}^{\mathrm{n}}\left[\mathrm{y}^{(\mathrm{i})\log}\left(\widehat{\mathrm{y}^{(\mathrm{i})}}\right) + \left(1 - \mathrm{y}^{(\mathrm{i})}\right)\log\left(1 - \widehat{\mathrm{y}^{(\mathrm{i})}}\right)\right]$$

Cross-entropy measures the divergence between two probability distribution, if the cross-entropy is large, which means that the difference between two distribution is large, while if the cross-entropy is small, which means that the two distribution is similar to each other. [16]

7.2 Optimizer

The function of the optimizer is to minimize (or maximize) the loss function E(x) by improving the training method.

Optimization algorithms fall into two broad categories:

1. The first order optimization algorithm

This algorithm uses the gradient values of the various parameters to minimize or maximize the loss function E(x). The most commonly used first-order optimization algorithm is gradient descent. Function gradient is a multivariate expression of the derivative $\frac{dy}{dx}$ used to represent the instantaneous rate of change of y with respect to x. Often in order to calculate the derivative of a multivariate function, the gradient is substituted for the derivative and the partial derivative is used to calculate the gradient. One major difference between the gradient and the derivative is that the gradient of the function forms a vector field.

2. Second-order optimization algorithm

The second-order optimization algorithm uses a second derivative (also called the Hessian method) to minimize or maximize the loss function. This method is not widely used because of the high computational cost of the second derivative.

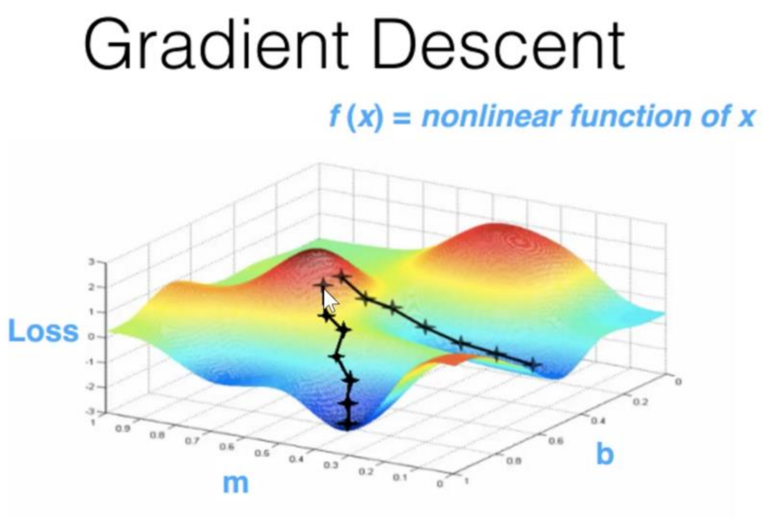


Figure 47. Gradient Descent [17]

In each iteration, the gradient descent updates the independent variable along the gradient of the current position based on the current position of the independent variable. However, if the iterative direction of the independent variable depends only on the current position of the independent variable, this may cause some problems. Some researchers have proposed a technique called Momentum, which accelerates non-momentum training by optimizing the training of related directions and weakening the oscillations of irrelevant directions. In the parameter update process, the principle is similar: 1. Make the network more optimal and more stable convergence; 2. Reduce the oscillation process.

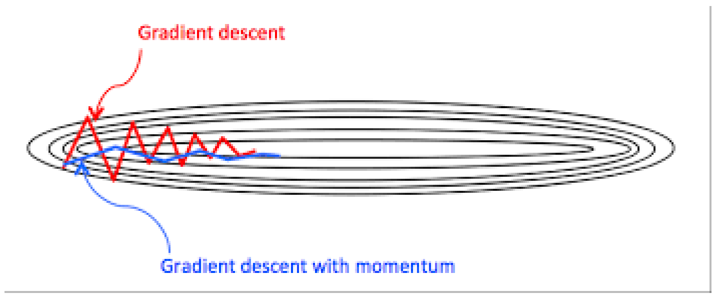


Figure 48. With momentum & without momentum [30]

There are four optimizers that used in the training process: Adagrad, Adadelta, Rmsprop, and Adam.

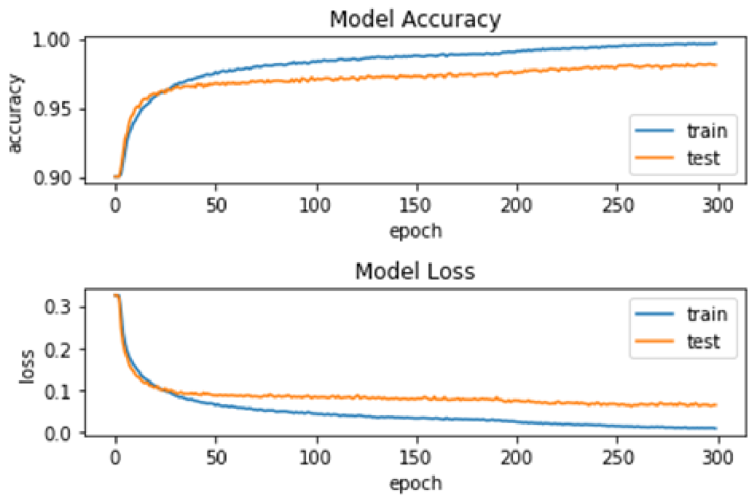


Figure 49. adam & binary cross entropy

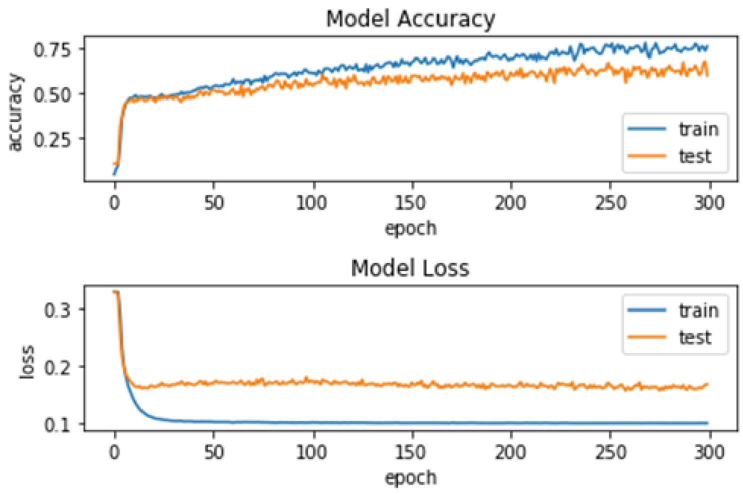


Figure 50. adam & Poisson

7.3 Loss Function Analysis

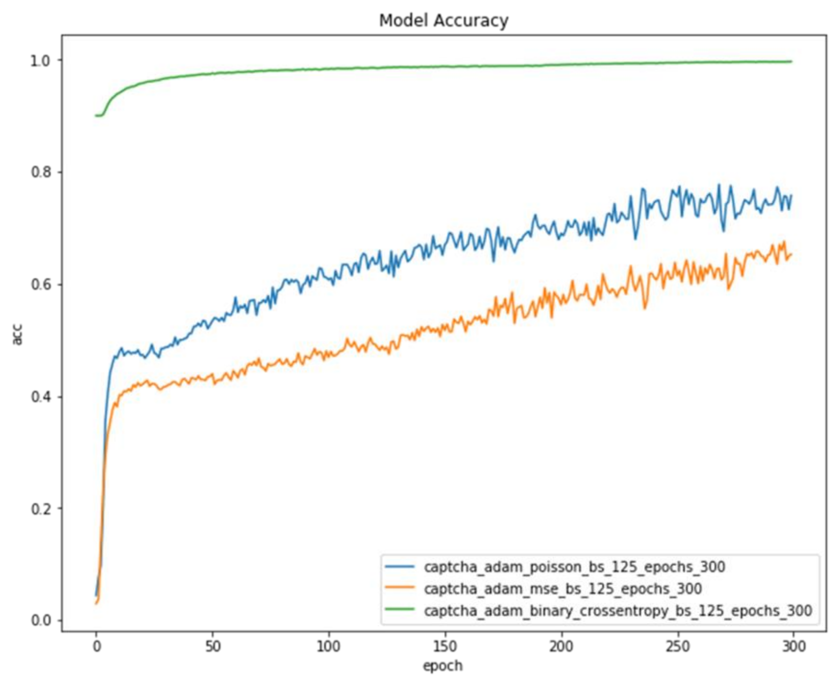


Figure 51. Model Accuracy comparison by different loss functions in train dataset

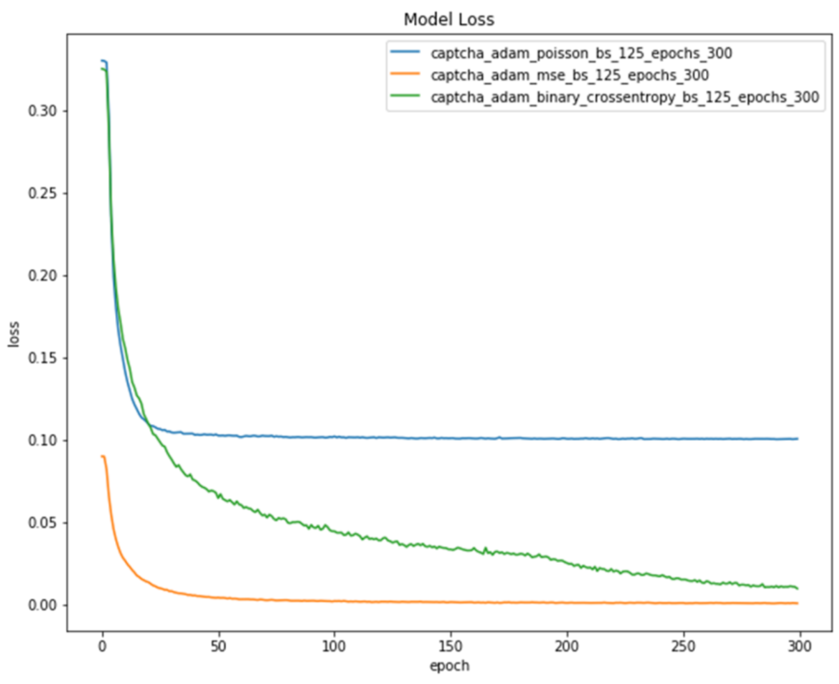


Figure 52. Model Loss comparison by different loss functions in train dataset

As we can see from the diagrams above, binary cross entropy performed better in train dataset as it has the highest accuracy and the loss rate also performed in a good way.

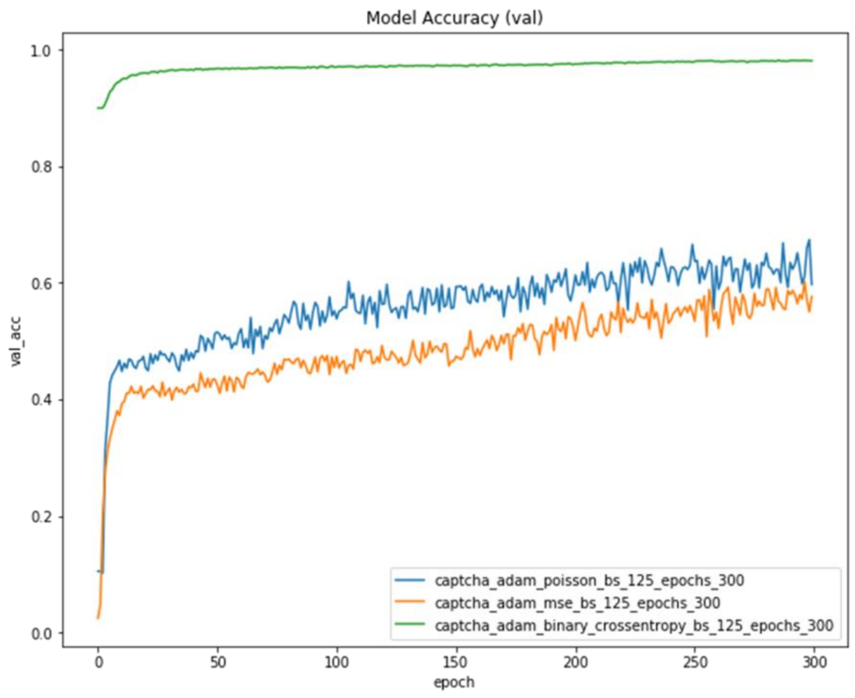


Figure 53. Model Accuracy comparison by different loss functions in the test dataset

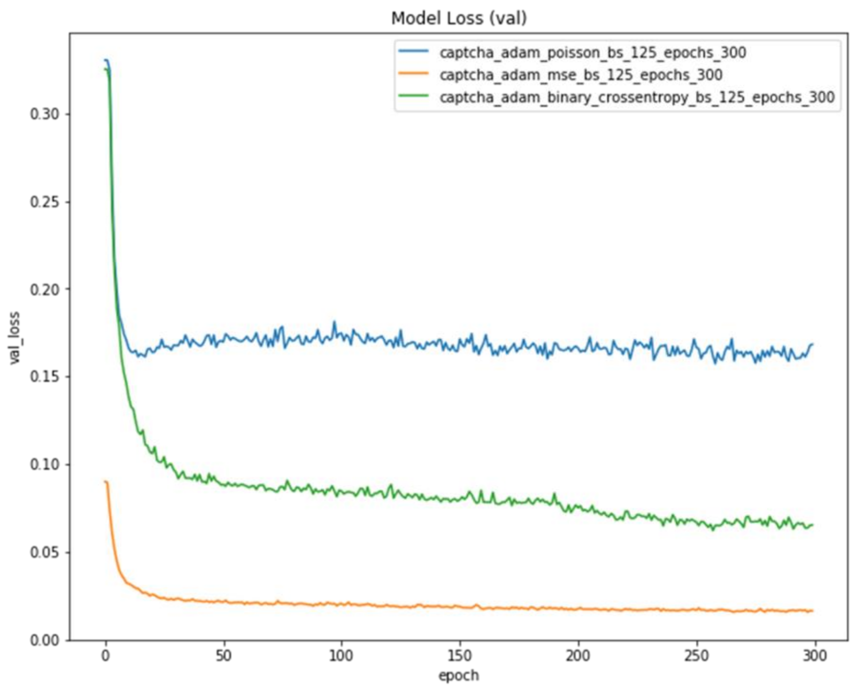


Figure 54. Model Loss comparison by different loss functions in the test dataset

In test dataset, MSE has a better loss rate compared other 2 methods, however, binary cross entropy is far more accurate than the other two.

7.4 Optimizer Analysis

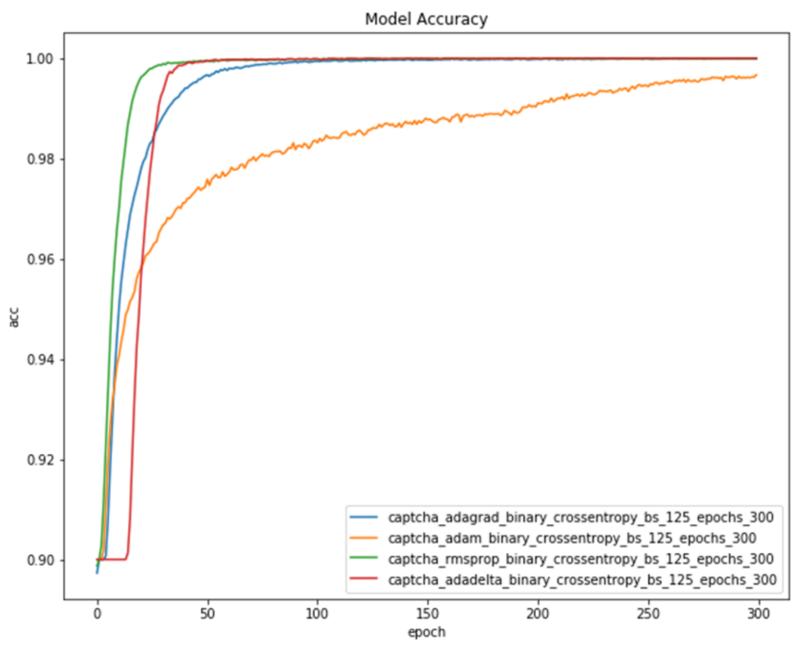


Figure 55. Model Accuracy comparison by different optimizers in train dataset

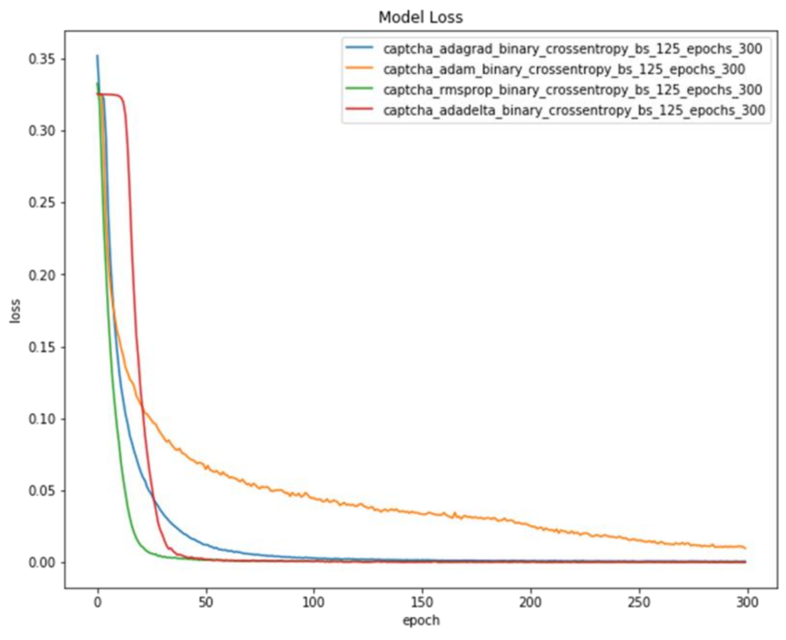


Figure 56. Model Loss comparison by different optimizers in train dataset

Except for Adam methods, those other three reached the highest accuracy and lowest loss rate in training set are quite similar, but Adam has better continuous learning ability than the other three methods.

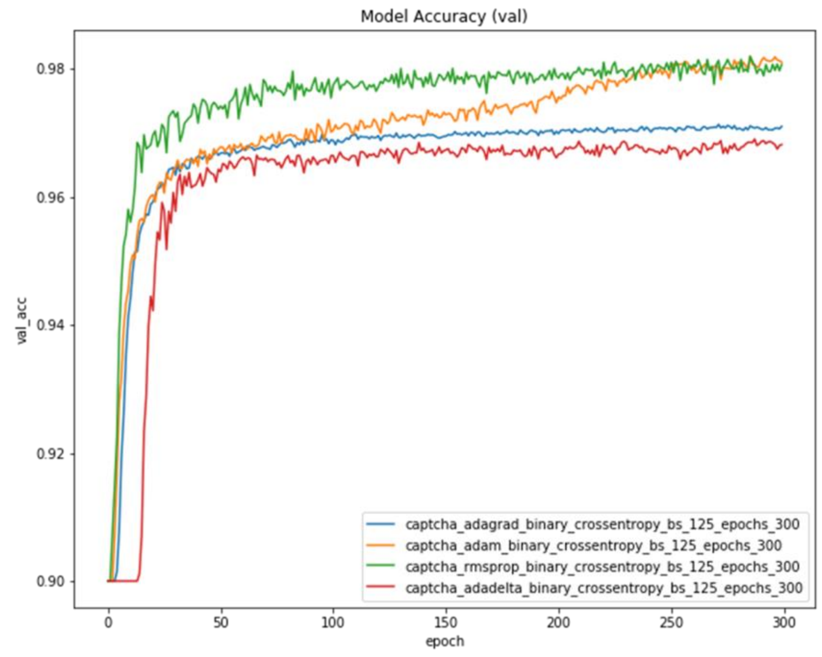


Figure 57. Model Accuracy comparison by different optimizers in the test dataset

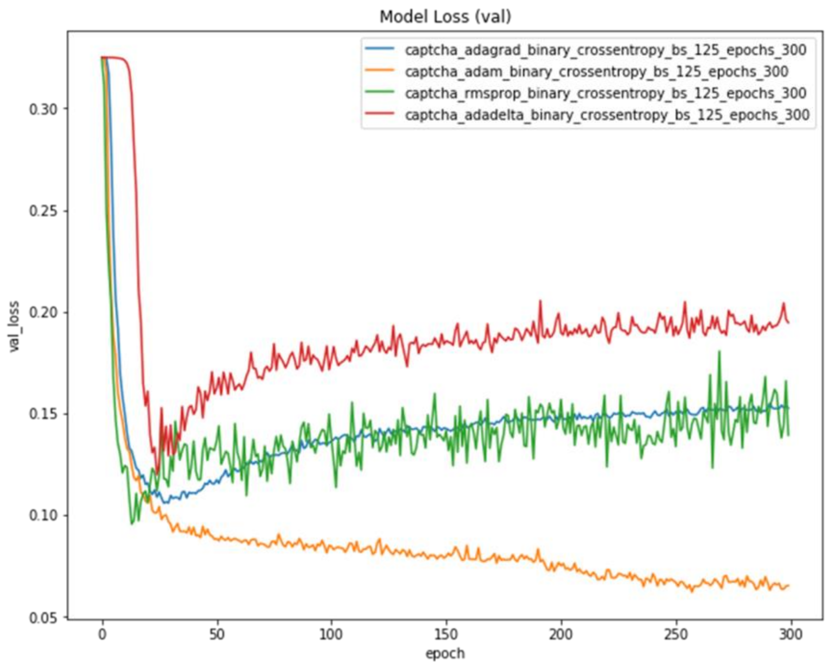


Figure 58. Model Loss comparison by different optimizers in the test dataset

However, after 300 epochs training, in the test dataset, Adam performs far better than the other three with the lowest loss rate. The other three reached a lower loss rate but re-bounded to a higher value after about 50 epochs, which means that with Adagrad, Adadelta, and Rmsprop, the model is overfitting with the training set that cannot continue to perform well on the test set or some general situations.

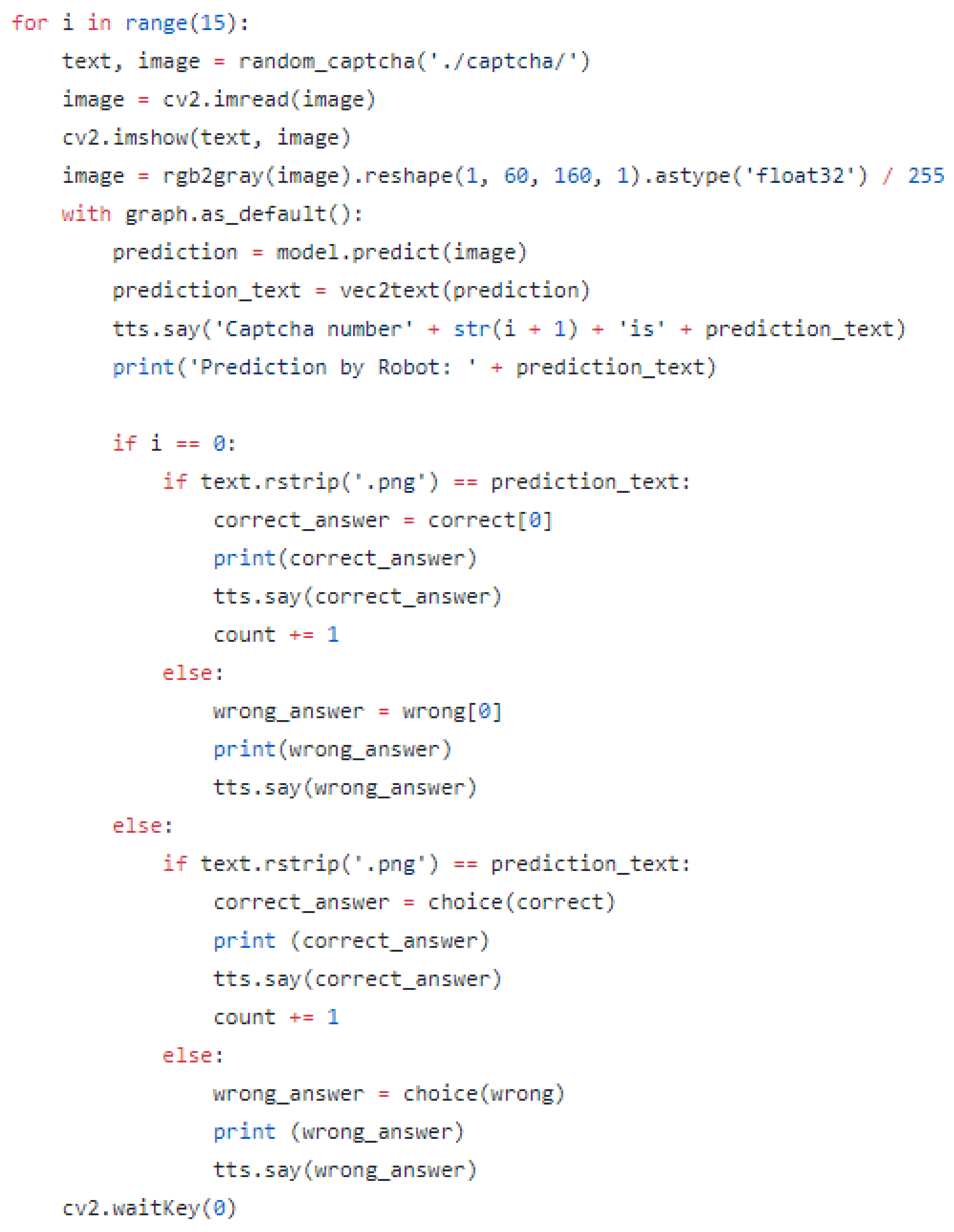
By comparing the performance within three loss functions and 4 optimizers, Adam with Binary Cross Entropy is more comprehensive on the test set, which will be used when deploying to real applications.

8 DEPLOYMENT

After we got our training model, it's time to deploy our model in real applications to test its generalization capability.

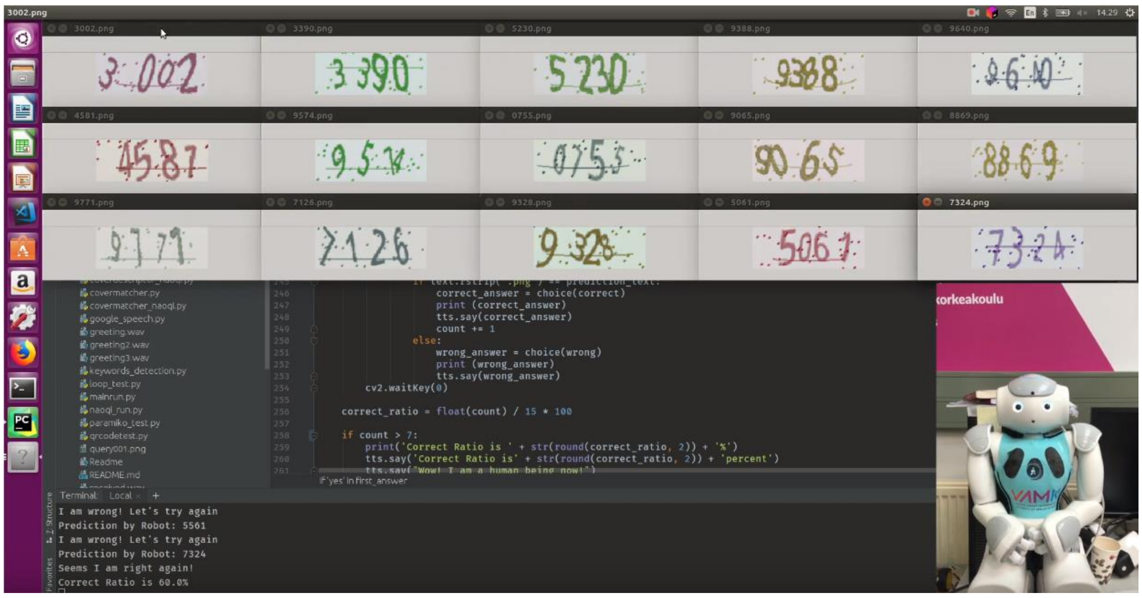
8.1 Deploy to NAO Robot

Deploy our recognition system on NAO robot is quite simple. First, we establish a framework that robot starts to begin the recognition test, then load our model to the robot and make it recognize 15 random generated CAPTCHA images, if the robot recognizes more than half of them, our model is considered performing well. Google Cloud Speech-to-Text is used for robot to detect starting command and ALTextToSpeech protocol is used to make responses. OpenCV library is used to display CAPTCHA images. OpenCV provides cv2.imread() method to read images, one advantage of using OpenCV is that after load image to memory through cv2.imread() method, the images are automatically converted to NumPy array format, which can make it easier to deal with the predicting output.



```
for i in range(15):
    text, image = random_captcha('./captcha/')
    image = cv2.imread(image)
    cv2.imshow(text, image)
    image = rgb2gray(image).reshape(1, 60, 160, 1).astype('float32') / 255
    with graph.as_default():
        prediction = model.predict(image)
        prediction_text = vec2text(prediction)
        tts.say('Captcha number' + str(i + 1) + 'is' + prediction_text)
        print('Prediction by Robot: ' + prediction_text)

        if i == 0:
            if text.rstrip('.png') == prediction_text:
                correct_answer = correct[0]
                print(correct_answer)
                tts.say(correct_answer)
                count += 1
            else:
                wrong_answer = wrong[0]
                print(wrong_answer)
                tts.say(wrong_answer)
        else:
            if text.rstrip('.png') == prediction_text:
                correct_answer = choice(correct)
                print (correct_answer)
                tts.say(correct_answer)
                count += 1
            else:
                wrong_answer = choice(wrong)
                print (wrong_answer)
                tts.say(wrong_answer)
    cv2.waitKey(0)
```

Figure 59. recognize CAPTCHA image



```
            if text.rstrip('.png') == prediction_text:
                correct_answer = choice(correct)
                print (correct_answer)
                tts.say(correct_answer)
                count += 1
            else:
                wrong_answer = choice(wrong)
                print (wrong_answer)
                tts.say(wrong_answer)
        cv2.waitKey(0)

    correct_ratio = float(count) / 15 * 100

    if count > 7:
        print('Correct Ratio is ' + str(round(correct_ratio, 2)) + '%')
        tts.say('Correct Ratio is' + str(round(correct_ratio, 2)) + 'percent')
        tts.say("Wow! I am a human being now!")
```

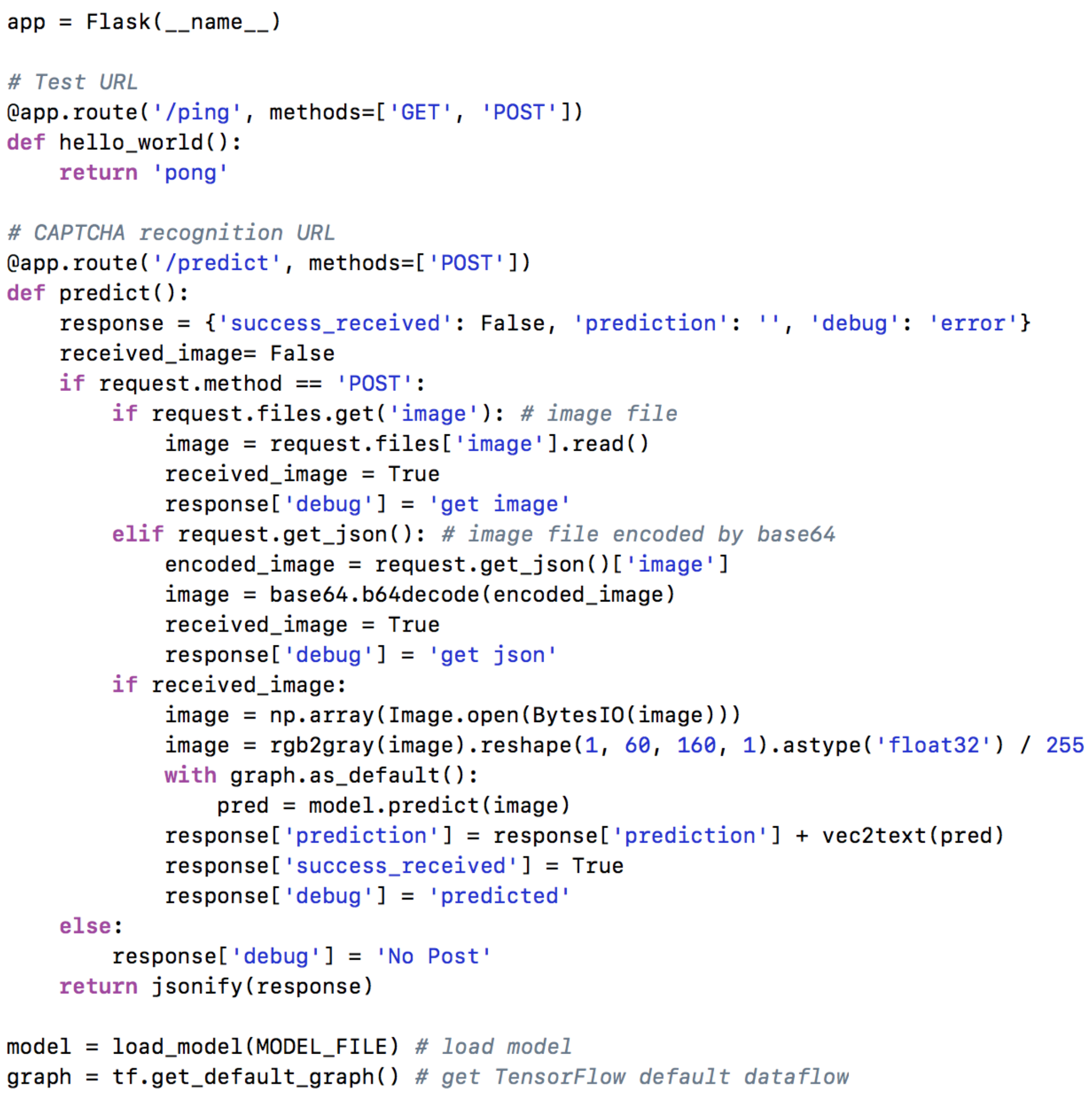
```
I am wrong! Let's try again
Prediction by Robot: 5561
I am wrong! Let's try again
Prediction by Robot: 7324
Seems I am right again!
Correct Ratio is 60.0%
```

Figure 60. Robot recognizing CAPTCHA images

A complete test video can be watched at https://youtu.be/4pxHi9h1caU

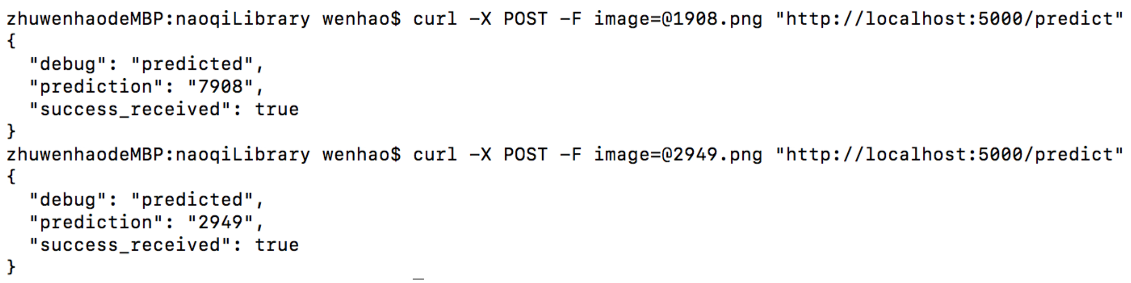
8.2 Deploy to A Web Service

We also provide an interface of our recognition service on web applications. Flask web framework is used to create our testing application, and CURL command is used to call the service.



```python
app = Flask(__name__)

# Test URL
@app.route('/ping', methods=['GET', 'POST'])
def hello_world():
    return 'pong'

# CAPTCHA recognition URL
@app.route('/predict', methods=['POST'])
def predict():
    response = {'success_received': False, 'prediction': '', 'debug': 'error'}
    received_image= False
    if request.method == 'POST':
        if request.files.get('image'): # image file
            image = request.files['image'].read()
            received_image = True
            response['debug'] = 'get image'
        elif request.get_json(): # image file encoded by base64
            encoded_image = request.get_json()['image']
            image = base64.b64decode(encoded_image)
            received_image = True
            response['debug'] = 'get json'
        if received_image:
            image = np.array(Image.open(BytesIO(image)))
            image = rgb2gray(image).reshape(1, 60, 160, 1).astype('float32') / 255
            with graph.as_default():
                pred = model.predict(image)
            response['prediction'] = response['prediction'] + vec2text(pred)
            response['success_received'] = True
            response['debug'] = 'predicted'
    else:
        response['debug'] = 'No Post'
    return jsonify(response)

model = load_model(MODEL_FILE) # load model
graph = tf.get_default_graph() # get TensorFlow default dataflow
```

Figure 61. Deploy CAPTCHA recognition service to a web server



```
zhuwenhaodeMBP:naoqiLibrary wenhao$ curl -X POST -F image=@1908.png "http://localhost:5000/predict"
{
  "debug": "predicted",
  "prediction": "7908",
  "success_received": true
}
zhuwenhaodeMBP:naoqiLibrary wenhao$ curl -X POST -F image=@2949.png "http://localhost:5000/predict"
{
  "debug": "predicted",
  "prediction": "2949",
  "success_received": true
}
```

Figure 62. Call a CAPTCHA recognition service through CURL command

9 CONCLUSION

In this thesis, a CAPTCHA recognition system is developed, by using convolutional neural network, which is a popular technique in deep learning field and can be deployed to a humanoid NAO robot and web service to execute recognition service. During the establishment and the training process, there are some issues found that can be improved in future development.

As the label text is translated to vector by one-hot encoding system, there are 36 of zeros and 4 of ones containing in the NumPy array. Because of this feature, if our model predicts the correct position of each zero, the accuracy would rise to a high level. When I was training the model, accuracy started at 90%, which is not an accurate number for the initial training moment. In the future, maybe another encoding system could be used to solve this problem.

Another thing that can be improved is that the CAPTCHA image only contains numbers as the text because of the huge amount of data if we introduced letters and other characters in our dataset (62^4 combinations if we introduce uppercase and lowercase letters). It can be improved if computing capability is increased in the future.

In this project, establishing the convolutional neural network and designing the training model are the core parts. CAPTCHAs are becoming more and more difficult to crack nowadays, but with the deep learning algorithm developing, more advanced neural network will be developed to cope with increasingly complex CAPTCHA algorithms.

10 REFERENCES

[1] M. COPELAND, 29 07 2016. [Online]. Available: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/.

[2] duanxxnj@163.com, 05 11 2016. [Online]. Available: https://www.zybuluo.com/Duanxx/note/425280.

[3] U. Karn, "Neural network Quick Start: What are multilayer sensors and reverse propagation?," [Online]. Available: https://zhuanlan.zhihu.com/p/23937778.

[4] yeler082, "Common convolution neural networks from basic principles to structural aggregation," 16 07 2018. [Online]. Available: https://blog.csdn.net/yeler082/article/details/81060020.

[5] Y. James, "[Material analysis & Machine Learning] Speech 5.1: A Convolutional network introduction (Neural Network).," 24 12 2017. [Online]. Available: https://medium.com/jameslearningnote/%E8%B3%87%E6%96%99%E5%88%86%E6%9E%90-%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E7%AC%AC5-1%E8%AC%9B-%E5%8D%B7%E7%A9%8D%E7%A5%9E%E7%B6%93%E7%B6%B2%E7%B5%A1%E4%BB%8B%E7%B4%B9-convolutional-neural-network-4f8249d65d4f.

[6] ""Machine learning" thoroughly understands CNN," 14 12 2017. [Online]. Available: https://mp.weixin.qq.com/s/EZGqfM61POfOZBA-QklNJA.

[7] S. Team, "Convolutional Neural Networks (CNN): Step 3 - Flattening," SuperDataScience, 18 08 2018. [Online]. Available: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening.

[8] S. T.-Y. M. I. T. A. M. M. Kota Ando, "A Multithreaded CGRA for Convolutional Neural Network Processing," 29 6 2017. [Online]. Available: https://file.scirp.org/pdf/CS_2017062915570340.pdf.

[9] M. Lutz, Learning Python, Fourth Edition, China Machine Press, 2011.

[10] "NumPy," [Online]. Available: https://www.numpy.org/.

[11] "Matplotlib," [Online]. Available: https://matplotlib.org/.

[12] T. Guide. [Online]. Available: https://www.tensorflow.org/guide/graphs.

[13] "Keras: The Python Deep Learning library," [Online]. Available: https://keras.io/.

[14] "Project Jupyter," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Project_Jupyter.

[15] "Google Speech-to-Text," [Online]. Available: https://cloud.google.com/speech-to-text/.

[16] "Flask (web framework)," Wikipedia, 27 04 2019. [Online]. Available: https://www.wikiwand.com/en/Flask_(web_framework).

[17] "NAO - Construction," Aldebaran Group, [Online]. Available: http://doc.aldebaran.com/2-1/family/robots/dimensions_robot.html.

[18] "NAO - Video camera," Aldebaran Group, [Online]. Available: http://doc.aldebaran.com/2-1/family/robots/video_robot.html.

[19] "ALAudioRecorder API," Aldebaran Group, [Online]. Available: http://doc.aldebaran.com/2-1/naoqi/audio/alaudiorecorder-api.html.

[20] "NAO - Microphones," Aldebaran Group, [Online]. Available: http://doc.aldebaran.com/2-1/family/robots/microphone_robot.html.

[21] "NAO - Loudspearkers," Aldebaran Group, [Online]. Available: http://doc.aldebaran.com/1-14/family/robots/loudspeaker_robot.html.

[22] "ALTextToSpeech Tutorial," Aldebaran Group, [Online]. Available: http://doc.aldebaran.com/2-4/naoqi/audio/altexttospeech-tuto.html.

[23] pingzishinee, "One-Hot Encoding," 13 12 2018. [Online]. Available: https://blog.csdn.net/u013317445/article/details/84983569.

[24] S.-H. Tsang, "Review: AlexNet, CaffeNet—Winner of ILSVRC 2012 (Image Classification)," 09 08 2018. [Online]. Available: https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in-ilsvrc-2012-image-classification-b93598314160.

[25] K. S. &. A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations*, 2014.

[26] M. u. Hassan, "VGG16 – Convolutional Network for Classification and Detection," 20 11 2018. [Online]. Available: https://neurohive.io/en/popular-networks/vgg16/.

[27] B. D. Digest, "5 Common Loss Functions," 21 06 2018. [Online]. Available: https://www.jiqizhixin.com/articles/2018-06-21-3.

[28] Z. Hao, "Loss Functions in Neural Networks," 07 06 2017. [Online]. Available: https://isaacchanghau.github.io/post/loss_functions/.

[29] J. Harris, "Stochastic gradient descent in plain English," 14 05 2018. [Online]. Available: https://medium.com/@julian.harris/stochastic-gradient-descent-in-plain-english-9e6c10cdba97.