The University of Hong Kong

ELEC6604 Neural Networks,
Fuzzy Systems and Genetic Algorithms

Routine Report for Google Vision Kit

Lecturer: Dr. G. Pang

Students: *ZHAO Yunqing* *CHENG Ming*

02/03/2019 Hong Kong

Contents

Part 1. Communication Mechanism

1.1 Connection method of Kit

Graph 1 connection procedure of devices

The steps of the communication are as follows:

1. Firstly, the Google Vision Kit must be connected to a Wi-Fi network, so we can use SSH proxy to login it remotely using our PC.
2. Secondly, because the Vision Kit has no screen or other peripheral devices, it is difficult to see the calculating/prediction result of ANN model. To address this problem, we developed a platform that the Vision Kit could communicate with mobile phone by WeChat.

1.2 WeChat Platform Construction

The details of step **2**: WeChat Platform Construction:

(1) In the first step, once the Vision Kit starts working, we changed the **source document** inside the system, making it run the python script automatically, which can let us login in a WeChat account of a Web version. In this procedure, we transform the above-mentioned python script as a specified service when plug in the power of Google Vision Kit. Hence, the goal can be achieved.

(2) After login in the Web WeChat by using a famous python library called "itchat" (**https://pypi.org/project/itchat/**), we used and revised some related functions of itchat to let the WeChat Account (dedicatedly used for Vision Kit) continuously listen to our group Chatting-Room, then analyze and return the feedback of the received messages.

We set a series of standards for parsing the received messages. For example, once the message with a prefix of "**cmd**", then the other part of the message will be

regarded as a Linux command by Kit, and our program could call OS module to execute this command.

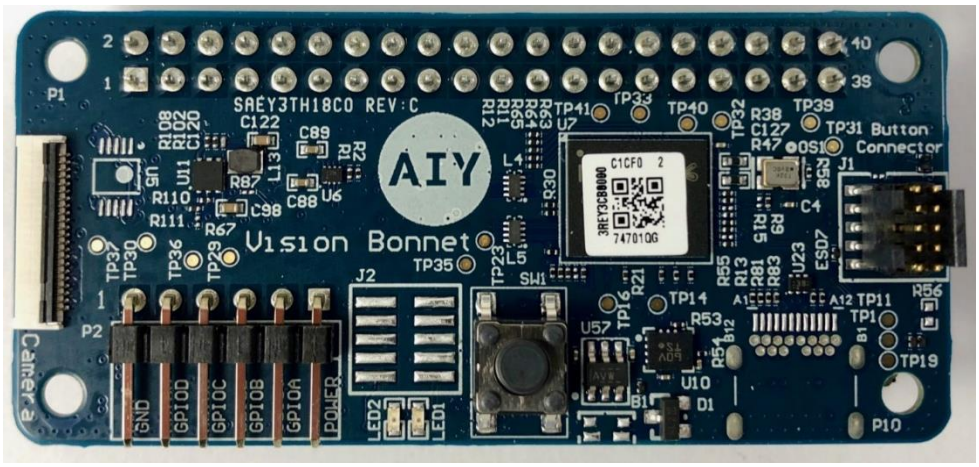
(3) On the other hand, the self-started program has multiple threads and it can execute several parallel tasks. One of the threads is to shoot a photo if the button is pressed, then save the picture in a pointed file path. Another thread is to scan this file path continuously, and send the new picture in it to WeChat, to tell us what it has shoot and saved.

By this communication mechanism and the construction of the WeChat platform, we can easily control the Vision Kit and get different types of responses from it. It is helpful to our next tasks.

Part 2. Recognition Platform

2.1 Official Guide from Google (Unsuccessful)

Since we want to build our own customized machine learning model, firstly we refer to the official guideline from the google, because we want to use the Bonnet (a part of hardware of Vision Kit) to accelerate the Machine Learning inference process by its TPU module.



Graph 2 Vision Bonnet

As the reference from official website of Google Vision Kit, (https://aiyprojects.withgoogle.com/vision/#makers-guide), we found that there are too many constraints, due to the **limited computational capability** of the Google Vision Kit, which means we can only use a very small number of open APIs released by google, on some specified network structures, e.g. MobileNet V1 and SqueezeNet, customized and provided by google. Meanwhile, we can only retrain the last fully connected network layer to make it recognize the things.

This method is of narrow limitation, in which we cannot achieve the goal by setting our

own TensorFlow Model and change the structure to what we want. Also, there are so many syntax error and uncertainty of the command provided by google. For a conclusion, we **cannot** use the official module to achieve the digit recognition goal.

2.2 TensorFlow is not Available on Kit (Unsuccessful)

From the mentioned situation in 2.1, we cannot successfully use the official compiler provided by google, which should have transformed the TensorFlow model to binary file, originally. Thus, we try to perform our own model by TensorFlow or Keras on Raspberry Pi, which we already have one trained model.

However, after having a detailed search on the website, we found that, the Raspberry Pi equipped on the Google Vision Kit is Raspberry Pi zero, based on the ARM 6 architecture, cannot support TensorFlow. (Generally, we can run TensorFlow directly on Raspberry Pi 3B or above version.)That means, we **cannot successfully** install the TensorFlow module at all, the provided official demo like face detection of your smile/angry from Google is just a binary file compiled by google.

However, the compiler doesn't support to transform your own TensorFlow model or it is extremely hard(which make us waste several days to try to make it works). Thus, this method is aborted.

2.3 Forward Inference using NumPy (Successful !)

Since 2.1 and 2.2 demonstrated that the unsuccessful working with official guide and the TensorFlow installation on the Raspberry Pi Zero as well, we found a method to use NumPy to directly calculate the forward inference procedure and output the predicted result directly. Because the System of Raspberry Pi have some fundamental python libraries like **NumPy, PIL**(Python Image Library), so we can process the captured photos and make computation.

The detail steps for this process are as follows:

(1) Train the model for Digit Recognition, with dataset MNIST. The Network structure is as Graph 3:

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 28, 28, 3)         30
_________________________________________________________________
activation_3 (Activation)    (None, 28, 28, 3)         0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 3)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 588)               0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                18848
_________________________________________________________________
activation_4 (Activation)    (None, 32)                0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                330
=================================================================
Total params: 19,208
```

Graph 3 Trained Network Structure

By this way, we defined functions for computation of involved matrix for fully connected network , convolutional procedure for CNN, and Activation function like “RELU” and the output layer “SoftMax”, and the MaxPooling as well.

This means, we **no longer need TensorFlow** anymore, and we **no longer need OpenCV Module**(which was also of unsuccessful installation on Raspberry Pi for Vision Kit), we just analog the process of forward inference, and try to use PIL library to read the photos rather than OpenCV, use NumPy rather than TensorFlow to get the prediction result. Finally, we just need the communication circuit mentioned in Part 1 to send the prediction result to our WeChat chatting room.

Part 3. Side Story of *Real-Number* Recognition

3.1 Comparison between MNIST and Real-Number

The resource from the Internet are almost about “How to build the Machine Learning Mode for MNIST ”. However, few people have curiosity about recognition of real handwritten number by us.

Since we have Google Vision Kit, we naturally want to make fully use of it, that is, once we shoot the **real number** we write on the paper/iPad, the Kit can automatically calculate the prediction result, as the process of Part 2.3, and output the class of number through the WeChat. For this task, we invite some of our friends to write the real digits to test the model, the number of real numbers is 524.

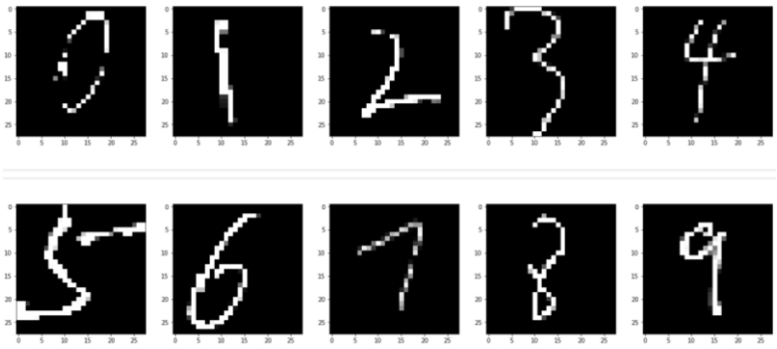
Not that easy, we soon found that the trained mature neural network model with highly accuracy on MNIST(99% or above) performed **poor** on the real numbers, as the Table 1 shows.

Datasets	Loss	Accuracy
MNIST	0.0687	0.9798
REAL NUMBER	**2.4393**	**0.41129**

Table 1 Result Comparison of Raw Model

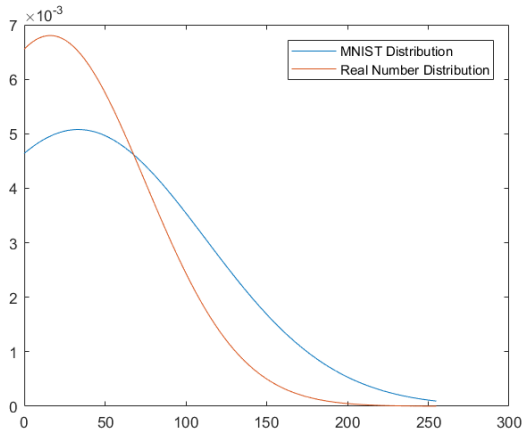
The following Graphs show the comparison between MNIST dataset and Real Hand-Written Numbers.

Graph 4 MNIST Number



Graph 5 Real Hand-Written Number

The problem is at this. After calculating the distribution of these numbers and the difference about the two datasets, we find that, the mean value of MNIST is 33.38596, the standard deviation is 78.567444. However, the mean value of Real Hand-Written Number is 16.05849, and the standard deviation is 58.651166. Assuming that the distributions of these datasets are both Gaussian distributions, we plot the curves of them:



Graph 6 Probability Density Function of both Distribution

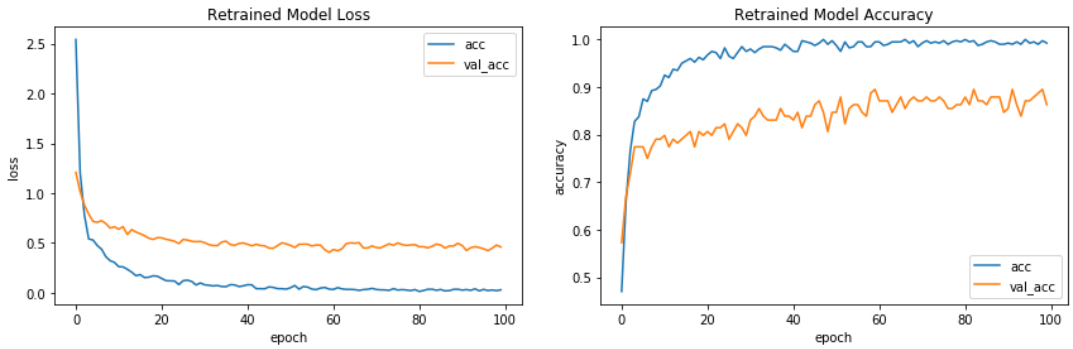
Hence, we clearly see that, the different distribution of respective dataset, that's **why** we cannot get the same performance as that of the MNIST images----in machine learning, we should keep the training set and testing set the IID condition(Independent and identically distributed).

3.2 Solution to the Problem: Transfer Learning!

Due to the big difference of their accuracy, we coped with the problem by transfer the raw model to retrain it on the Brand-New Dataset, named "***Hand-Written Dataset***", we will call it ***HWD*** from now on.

The HWD consists of 524 hand written numbers, written by 4 people. Three of the authors wrote the training set, and another one wrote the number for validation set, which means the validation set of HWD from a total new person, and the training set **NOT** consist of any number from the person that wrote the validation number. This is the background.

Then, we load the trained raw model of MNIST, deploy the HWD, and retrain the model. As has been mentioned before(Table 1), the raw model performed poor on the HWD with only 40% accuracy, however it jumped to 88% at the end of the retrain station, as Graph 7 shows, the decrease of the loss value and the increase of the accuracy demonstrate the stronger capability of generalization.



Graph 7 The training accuracy of retrained model

3.3 Realization of Real-Time Recognition

This is the final stage of this demo. We made the synthesis of the above all units. In this stage, we let the Google Vision Kit shoot the real-time digit Written Number and then output the result to the WeChat chatting room.

As the Graph 8 shows, once we shoot the photo by Kit, the Raspberry Pi will make inference calculation of the model forwardly and return the result in a short latency.

Based on the principle of considering all the details, we also upload the ***Video*** of our test scene by shooting and the inference procedure on the YouTube: *https://youtu.be/ZyAYFuCDyrI*. Feel free to reach it. The test result is relatively good, but we must point out some problems, which will be prohibited later in the fruit recognition:

(1) During the procedure of making HWD (Real digit numbers by hand), we scratch the square size of the picture, and then resize to 28*28. However, in the normal cases, we cannot get the so perfect size by simply shooting the photos in a casual way. So, maybe the training result is so

good, but once we test the model as the *YouTube Video* like, we must fix the position of the Vision Kit and then take pictures. Once we take photos in a random angle or posture, the accuracy will drop down. So, for next task on fruit recognition, we will prohibit it by making datasets in a random angle and different size.

(2) During the procedure of making HWD (Real digit numbers by hand), even though from different people, all the digit numbers come from iPad, which means the dataset are all absolute binary image, however the image when shooting is of gray scale, and the image effect will also decay by effect of the indoor lighting or sunshine or reflection of the light.

Hence, in order to achieve the better result, we must make sure the distribution of the image keep identity, i.e. take photos by Google Vision Kit to make training set, thus the similar pictures can be correctly recognized by the device.

Graph 8 Test Scene

Part 4. Conclusion

In this project, we design an integrated, device-based recognizer Vision Kit model. Firstly, we use the python library “itchat” and local Wi-Fi network to build a platform in order to make communication. Then we form a new database HWD to retrain the network based on the raw one.

Finally, the Vision Kit can recognize the real digit number and send the prediction result to the chatting room in a short latency, that is what we have achieved. And some technical problems will be resolved and dealt with in later task. Next, we may form a new goal and new objects, to have a total combination of the all modules, from dataset, algorithm, to the final result.

Appendix: Inference Engine

In this Module, the python program achieves the forward inference, as mentioned in Part 2.3.

```
import numpy as np
class Recognizer():
    def __init__(self):
        self.conv_weights = np.load('model/weights/conv_weights.npy')    # load weights
        self.conv_bias = np.load('model/weights/conv_bias.npy')
        self.fc1_weights = np.load('model/weights/fc1_weights.npy')
        self.fc1_bias = np.load('model/weights/fc1_bias.npy')
        self.fc2_weights = np.load('model/weights/fc2_weights.npy')
        self.fc2_bias = np.load('model/weights/fc2_bias.npy')
```

```
def predict(self, x, img_enhance=False):
    x = x.reshape((28,28,1))
    x = x / 255.0
    if img_enhance:
        x = self.enhancement(x)
    output = self.Inference(x)
    label = int(np.where(output==np.max(output))[1])

    return    output,label

def Inference(self,x):
    x = self.Conv2D(x, self.conv_weights, self.conv_bias, strides=(1,1))
    x = self.Relu(x)
    x = self.Max_pooling(x, kernel_size=(2,2), strides=(2,2))
    x = self.Flatten(x)
    x = self.Dense(x, self.fc1_weights, self.fc1_bias)
    x = self.Relu(x)
    x = self.Dense(x, self.fc2_weights, self.fc2_bias)
    x = self.Normalization(x)
    return x

def Relu(self, x):
    x[x<0] = 0
    return x

def Flatten(self, x):
    # make input tensor resized as (1,n) shape
    x = x.reshape(1,-1)
    return x

def Normalization(self, x):
    max_x = np.max(x)
    min_x = np.min(x)
    y = (x - min_x) / (max_x - min_x)
    return y

def Conv2D(self, inputs, kernels, biases, strides=(1,1)):
    # input tensor size as[heigh,width,channel],
    # kernel size size as [heigh,width,channel,k]
    input_size = inputs.shape
    kernel_size = kernels.shape
    # padding input tensor as the size of kernel
    h_pad = int(kernel_size[0] / 2)
    w_pad = int(kernel_size[1] / 2)
```

```
            inputs=np.pad(inputs,pad_width=((h_pad,h_pad),(w_pad,w_pad),(0,0)),
mode='constant')
            # compute the size of output tensor, and initialize the output tensor
            w_new = int((input_size[0] - kernel_size[0] + 2*h_pad) / strides[0] + 1)
            h_new = int((input_size[1] - kernel_size[1] + 2*w_pad) / strides[1] + 1)
            outputs = np.zeros((w_new,h_new,kernels.shape[3]))
            # do convolutional computation by channels
            for k in range(kernel_size[3]):
                # (i_new,j_new) is the position in output tensor, (i,j) is the center position of input
tensor
                for i_new,i in enumerate(range(h_pad, inputs.shape[0]-h_pad, strides[0])):
                    for j_new,j in enumerate(range(w_pad, inputs.shape[1]-w_pad, strides[1])):
                        # temporary window value after multiply and make sum
                        temp = inputs[i-h_pad:i+h_pad+1, j-w_pad:j+w_pad+1, :]
                        temp = self.Flatten(temp)
                        # kernel is the k th kernel
                        kernel = self.Flatten(kernels[:,:,:,k])
                        # compute the convolutional value and output
                        value = np.dot(temp, kernel.T) + biases[k]
                        outputs[i_new,j_new, k] = value
            return outputs

    def Max_pooling(self, x, kernel_size=(2,2), strides=(2,2)):
            # compute the size of output tensor
            width, heigh, channel = x.shape
            # creat output tensor as kernel size strides
            w_new = int((width - kernel_size[0]) / strides[0] + 1)
            h_new = int((heigh - kernel_size[1]) / strides[1] + 1)
            c_new = channel
            outputs = np.zeros((w_new,h_new, c_new))
            # pooling by channel
            for k in range(channel):
                # (i_new,j_new) coordinates of output tensor，(i,j) is input tensor coordinates
                for i_new,i in enumerate(range(0, width, strides[0])):
                    for j_new,j in enumerate(range(0, heigh, strides[1])):
                        # (i,j) is coordinates of upper left
                        temp = x[i:i+kernel_size[0],j:j+kernel_size[1]]
                        # extract the maximum value inside this window
                        outputs[i_new,j_new,k] = np.max(temp)
            return outputs

    def Dense(self, inputs, kernel, bias):
            # shape the size of input tensor and bias as (1,n)
            inputs = inputs.reshape(1,-1)
```

```
        bias = bias.reshape(1,-1)
        # vectorize the output
        output = np.dot(inputs, kernel) + bias
        return output

def enhancement(self, x): # image enhancement
        mean = np.mean(x)
        x[x>=mean] = 1.0
        x[x<mean] = 0.0
        return x
```