

Udacity Self-Driving Car Engineer

Report

- **Project:** Traffic Sign Classifier
- **Name:** Shao Hongxu

Let me just use some brief sentence to explain the thing I did, basically these are 6 big parts here:

Part 1: import necessary lib

Part 2: identify Main function

Part 3: calculate model validation prediction accuracy

Part 4: using trained model to predict new image from web & provide top 5 predict possibility.

Part 5: Pre-processing techniques used and why these techniques were chosen

Part 6: Summary

To be specific as below, you will find some note on Script, key take away and Area could be improved:

Part 1: import necessary lib:

```
import pickle
import numpy as np
import matplotlib.pyplot as plt
from skimage.color import rgb2gray
import os
import cv2

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
```

Part 2: identify Main function:

```
def Main():

    train_data_sizes = []
    train_data_coords = []
    train_data_features = []
    train_data_labels = []
    try:
        with open('F:/Udacity/Project 2/traffic-signs-data/train.p','rb') as snapFile:
            savedSnaps = pickle.load(snapFile)
    except IOError as e:
        return False
    train_data_sizes=savedSnaps['sizes']
    train_data_coords = savedSnaps['coords']
    train_data_features = savedSnaps['features']
    train_data_labels = savedSnaps['labels']
    train_images = rgb2gray(train_data_features)
```

To archive the goal of deep learning Model, below is what I have done under Main function:

2. 1: Import data of training, validation, testing set. Original data is within '.p' format, here we need to pull them and save into list format for further usage:

```

def Main():

    train_data_sizes = []
    train_data_coords = []
    train_data_features = []
    train_data_labels = []
    try:
        with open('F:/Udacity/Project 2/traffic-signs-data/train.p','rb') as snapFile:
            savedSnaps = pickle.load(snapFile)
    except IOError as e:
        return False
    train_data_sizes=savedSnaps['sizes']
    train_data_coords = savedSnaps['coords']
    train_data_features = savedSnaps['features']
    train_data_labels = savedSnaps['labels']
    train_images = rgb2gray(train_data_features)

    valid_data_sizes = []
    valid_data_coords = []
    valid_data_features = []
    valid_data_labels = []
    try:
        with open('F:/Udacity/Project 2/traffic-signs-data/valid.p','rb') as snapFile:
            savedSnaps = pickle.load(snapFile)
    except IOError as e:
        return False
    valid_data_sizes=savedSnaps['sizes']
    valid_data_coords = savedSnaps['coords']
    valid_data_features = savedSnaps['features']
    valid_data_labels = savedSnaps['labels']
    valid_images = rgb2gray(valid_data_features)

    test_data_sizes = []
    test_data_coords = []
    test_data_features = []
    test_data_labels = []
    try:
        with open('F:/Udacity/Project 2/traffic-signs-data/test.p','rb') as snapFile:
            savedSnaps = pickle.load(snapFile)
    except IOError as e:
        return False
    test_data_sizes=savedSnaps['sizes']
    test_data_coords = savedSnaps['coords']
    test_data_features = savedSnaps['features']
    test_data_labels = savedSnaps['labels']
    test_images = rgb2gray(test_data_features)




```

-> **Key take away:** while pulling original image data, it is necessary to convert colour image into grey image before using it for deep learning model. It will reduce the dimensions of input data. In the end, the benefit would be faster training time.

-> **Area could be improved:** not all the provided data used in this code always, for those data no longer used any more, list to be cleaned to save computing Memory. More importantly, if we could enlarge original data, it would be very helpful, like project spec recommended, rotation, translation, zoom, flips, and/or colour perturbation etc. will decide later whether to do it or not depends on the time & accuracy.

-> **Result:**

Original data:

 test.p	7/5/2018 9:17 PM	P File	37,977 KB
 train.p	7/5/2018 9:17 PM	P File	104,636 KB
 valid.p	7/5/2018 9:17 PM	P File	13,261 KB

```
{'sizes': array([[53, 54],
                [42, 45],
                [48, 52],
                ...,
                [29, 29],
                [48, 49],
                [32, 31]], dtype=uint8), 'coords': array([[ 6,  5, 48, 49],
                [ 5,  5, 36, 40],
                [ 6,  6, 43, 47],
                ...,
                [ 6,  6, 24, 24],
                [ 5,  6, 43, 44],
                [ 6,  5, 27, 26]], dtype=uint8), 'features': array([[[[116, 139, 174],
                [116, 137, 171],
                [118, 138, 172],
                ...,
                [ 98, 114, 143],
                [ 97, 121, 147],
                [ 85, 105, 130]]],
```

Data after pulling into python:

```
[[ 67, 49, 44],
 [ 65, 50, 45],
 [ 65, 49, 45],
 ...,
 [ 57, 42, 39],
 [ 55, 42, 40],
 [ 56, 43, 40]],

...,

[[ 28, 24, 26],
 [ 29, 25, 27],
 [ 29, 27, 30],
 ...,
 [ 27, 24, 24],
 [ 28, 25, 24],
 [ 29, 27, 27]],
```

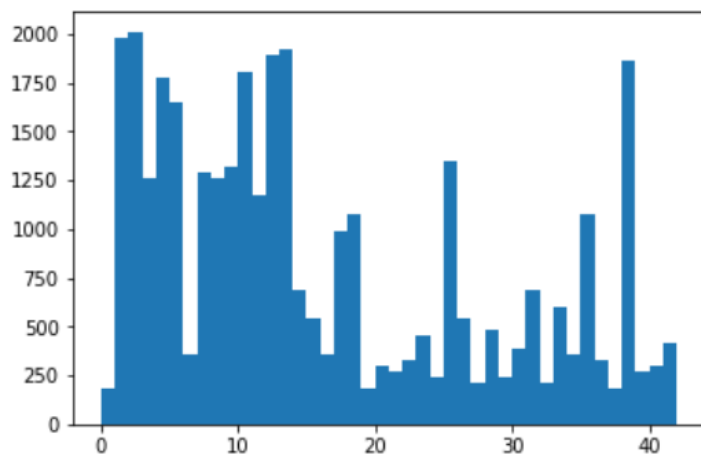
2. 2: Visualize the training data – image label

```
unique_labels = set(train_data_labels)
print(unique_labels)
plt.hist(train_data_labels, 42)
plt.show()
```

-> Result:

Summarise the training label, realized in total there are 42 different label sets, so that 43 output classifiers needed for the classification model:

```
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
34, 35, 36, 37, 38, 39, 40, 41, 42}
```



2. 3: using image size & border points, to crop the image into only traffic sign.

```
for i in range(0,34799):
    train_data_coors[i][0]=int(train_data_coors[i][0] * 32 / train_data_sizes[i][0])
    train_data_coors[i][1]=int(train_data_coors[i][1] * 32 / train_data_sizes[i][1])
    train_data_coors[i][2]=int(train_data_coors[i][2] * 32 / train_data_sizes[i][0])+1
    train_data_coors[i][3]=int(train_data_coors[i][3] * 32 / train_data_sizes[i][1])+1
    crop_img = train_images[i][train_data_coors[i][1]:train_data_coors[i][3], train_data_coors[i][0]:train_data_coors[i][2]]
    train_images[i] = cv2.resize(crop_img,(32,32))
name_values = np.genfromtxt('signnames.csv', skip_header=1, dtype=[('myint','i8'), ('mysring','S55')], delimiter=',')
```

-> **Key take away:** action here helped with increasing accuracy of training model.

2. 4: choose some random images and list down shape and min/max value from image data:

```
# Import the `pyplot` module of `matplotlib`
# Determine the (random) indexes of the train_images that you want to see
traffic_signs = [260, 74, 3515, 2004]

# Fill out the subplots with the random train_images that you defined
for i in range(len(traffic_signs)):
    plt.subplot(1, 4, i+1)
    plt.axis('off')
    plt.imshow(train_images[traffic_signs[i]])
    plt.subplots_adjust(wspace=0.5)
    plt.show()
    print("shape: {0}, min: {1}, max: {2}".format(train_images[traffic_signs[i]].shape,
                                                  train_images[traffic_signs[i]].min(),
                                                  train_images[traffic_signs[i]].max()))
```

-> **Key take away:** as mentioned from project Spec, all images data provided have already been converted into 32*32 shape.

Still I just put a checking point here in case any other similar model has different shape of images, we can have a basic sense of how various the images' shape are.

-> **Area could be improved:** here we can also draw a chart to show how many image there are in total etc.. Just from better understanding data perspective.

-> **Result:**

4 sample Images:



shape: (32, 32), min: 0.09213058823529413, max: 0.41472451439950986



shape: (32, 32), min: 0.05897559627757353, max: 0.23410500268075982



shape: (32, 32), min: 0.07269037990196078, max: 1.0



2. 5: to initiate a training model using TensorFlow:

```
import tensorflow as tf
print("before TF model")

# Initialize placeholders
x = tf.placeholder(dtype = tf.float32, shape = [None, 32, 32])
y = tf.placeholder(dtype = tf.int32, shape = [None])
# Flatten the input data
train_images_flat = tf.contrib.layers.flatten(x)
# Fully connected layer

# Input Layer
input_layer = tf.reshape(train_images_flat, [-1, 32, 32, 1])
print("conv1")
# Convolutional Layer #1
conv1 = tf.layers.conv2d(
    inputs=input_layer,
    filters=32,
    kernel_size=[5, 5],
    padding="VALID",
    activation=tf.nn.relu)
# Pooling Layer #1
pool1 = tf.layers.max_pooling2d(inputs=conv1, pool_size=[2, 2], strides=2)
# Convolutional Layer #2 and Pooling Layer #2
conv2 = tf.layers.conv2d(
    inputs=pool1,
    filters=32,
    kernel_size=[5, 5],
    padding="VALID",
    activation=tf.nn.relu)
pool2 = tf.layers.max_pooling2d(inputs=conv2, pool_size=[2, 2], strides=2)
#Dense Layer
pool2_flat = tf.reshape(pool2, [-1, 5 * 5 * 32])
```

```

# Dense Layer
dense1 = tf.layers.dense(inputs=pool2_flat, units=256, activation=tf.nn.relu)
drop_out1 = tf.nn.dropout(dense1, 0.5)
dense2 = tf.layers.dense(inputs=drop_out1, units=128, activation=tf.nn.relu)
drop_out2 = tf.nn.dropout(dense2, 0.5)
dense3 = tf.layers.dense(inputs=drop_out2, units=64, activation=tf.nn.relu)
drop_out3 = tf.nn.dropout(dense3, 0.5)
logits = tf.contrib.layers.fully_connected(drop_out3, 43, tf.nn.relu)
# Define a loss function
loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(labels = y,
                                                                    logits = logits))

# Define an optimizer
train_op = tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)
# Convert logits to label indexes
correct_pred = tf.argmax(logits, 1)
# Define an accuracy metric
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

saver = tf.train.Saver()

#batch = tf.train.batch([image, label], batch_size=100)
tf.set_random_seed(1234)
sess = tf.Session()
sess.run(tf.global_variables_initializer())

```

-> Key take away:

This is the most important area apparently, architect here obviously will impact the final accuracy in the end. Let's look at the model overview:

- **Input layer:** [-1, 32, 32, 1]
- **1st layer:** flatten layer
- **2nd layer:** Conv2d layer with filters=32, kernel_size=[5, 5], padding="VALID"
- **3rd layer:** max_pooling2d layer with pool_size=[2, 2]
- **4th layer:** Conv2d layer with filters=32, kernel_size=[5, 5], padding="VALID"
- **5th layer:** max_pooling2d layer with pool_size=[2, 2]
- **6th layer:** reshape layer, converting data into [-1, 5 * 5 * 32]
- **7th layer:** 256 units, with 0.5 dropout
- **8th layer:** 128 unit, with 0.5 dropout
- **9th layer:** 64 unit, with 0.5 dropout
- **Output layer:** 43

From input data size 32*32, we use:

- 2 Con2d layer
- 2 max_pooling2d layer
- 1 reshape layer
- 3 normal connecting layer

Reduce data size into 43 classifiers in the end.

-> **Area could be improved:** here is the key area can be improved, as all other machine learning/deep learning model, a bad model will not come out with any high accuracy prediction rate:

- how many levels of layer?

- how many units for each level of layer?
- dropout rate? here to be 0.5
- etc.

This part is the core of Machine learning/ deep learning. Also GPU, TPU play an important role of training out of the better model accuracy.

First time I was using laptop with 16GB CPU (no NAVIA GPU), 98% CPU usage stopped me doing that again since it may destroy the machine.

Second time I was using my desktop with NAVIA GPU 1080. It takes only 40min to complete 5000 echoes. And with not bad accuracy in the end, so the model architect mentioned above can work to a certain extend.

2. 6: Initiate model weight variables, and to set up run Epoch into 5000 times, with Batch size as 2000.

```
batch_size = 2000
hm_epochs = 5000
for epoch in range(1, hm_epochs+1):
    for j in range(int(34799/batch_size)):
        #print(j)
        #print('    Batch', j+1)
        m=j*batch_size
        n=(j+1)*batch_size
        _, accuracy_val = sess.run([train_op, accuracy], feed_dict={x: train_images[m:n], y: train_data_labels[m:n]})

    print('Epoch', epoch, 'completed out of', hm_epochs)

    if epoch % 10000 == 0:
        # Run predictions against the full test set.
        predicted_train = sess.run([correct_pred], feed_dict={x: train_images})[0]
        # Calculate correct matches
        match_count_train = sum([int(y == y_) for y, y_ in zip(train_data_labels, predicted_train)])
        # Calculate the accuracy
        accuracy_train = match_count_train / len(train_data_labels)
        # Print the accuracy
        print("Accuracy_train: {:.3f}".format(accuracy_train))

saver.save(sess, './lenet')
print("Model saved")
```

2. 7: once model training completed, to show the prediction of 10 random images

```
import random

# Pick 10 random train images
sample_indexes = random.sample(range(len(train_images)), 10)
sample_train_images = [train_images[i] for i in sample_indexes]
sample_labels = [train_data_labels[i] for i in sample_indexes]

# Run the "correct_pred" operation
predicted = sess.run([correct_pred], feed_dict={x: sample_train_images})[0]

# Print the real and predicted labels
print(sample_labels)
print(predicted)

# Display the predictions and the ground truth visually.
#fig = plt.figure(figsize=(10, 10))
for i in range(len(sample_train_images)):
    truth = sample_labels[i]
    prediction = predicted[i]
    plt.subplot(5, 2, i+1)
    plt.axis('off')
    color='green' if truth == prediction else 'red'
    plt.text(40, 10, "Truth:      {0}\nPrediction: {1}".format(truth, prediction),
            fontsize=12, color=color)
    plt.imshow(sample_train_images[i], cmap="gray")

plt.show()
```


-> Result:

Epoch 4999 completed out of 5000

Epoch 5000 completed out of 5000

Model saved

[1, 26, 8, 27, 1, 23, 11, 0, 16, 5]

[1 26 8 27 1 23 11 0 16 5]

<Figure size 640x480 with 10 Axes>

Part 3: calculate model prediction accuracy:

```
#crop(train_images[1],train_data_coors[1])
valid_data_sizes = []
valid_data_coors = []
valid_data_features = []
valid_data_labels = []
try:
    with open('F:/Udacity/Project 2/traffic-signs-data/valid.p', 'rb') as snapFile:
        #with open('C:/Users/alsshxu/Desktop/2018 HX/Udacity/Project 2/traffic-signs-data/valid.p', 'rb') as snapFile:
            savedSnaps = pickle.load(snapFile)
except IOError as e:
    return False
#print(savedSnaps)
valid_data_sizes=savedSnaps['sizes']
valid_data_coors = savedSnaps['coors']
valid_data_features = savedSnaps['features']
valid_data_labels = savedSnaps['labels']

valid_images = rgb2gray(valid_data_features)
for i in range(0,4410):
    valid_data_coors[i][0]=int(valid_data_coors[i][0] * 32 / valid_data_sizes[i][0])
    valid_data_coors[i][1]=int(valid_data_coors[i][1] * 32 / valid_data_sizes[i][1])
    valid_data_coors[i][2]=int(valid_data_coors[i][2] * 32 / valid_data_sizes[i][0])*1
    valid_data_coors[i][3]=int(valid_data_coors[i][3] * 32 / valid_data_sizes[i][1])*1
    crop_img = valid_images[i][valid_data_coors[i][1]:valid_data_coors[i][3], valid_data_coors[i][0]:valid_data_coors[i][2]]
    valid_images[i] = cv2.resize(crop_img,(32,32))

a1,b1 = np.shape(valid_data_sizes)
print (a1)
print (b1)

predicted_valid = sess.run([correct_pred], feed_dict={x: valid_images})[0]
# Calculate correct matches
match_count_valid = sum([int(y == y_) for y, y_ in zip(valid_data_labels, predicted_valid)])
# Calculate the accuracy
accuracy_valid = match_count_valid / len(valid_data_labels)
# Print the accuracy
print("Accuracy_valid: {:.3f}".format(accuracy_valid))
```

```

try:
    with open('F:/Udacity/Project 2/traffic-signs-data/test.p', 'rb') as snapFile:
        #with open('C:/Users/alsshxu/Desktop/2018 HX/UdaCity/Project 2/traffic-signs-data/test.p', 'rb') as snapFile:
            savedSnaps = pickle.load(snapFile)
    except IOError as e:
        return False
    #print(savedSnaps)
    test_data_sizes=savedSnaps['sizes']
    test_data_coords = savedSnaps['coords']
    test_data_features = savedSnaps['features']
    test_data_labels = savedSnaps['labels']
    test_images = rgb2gray(test_data_features)

    a1,b1,c1 = np.shape(test_images)
    print (a1)
    print (b1)
    print (c1)
    test_data_sizes[2847][0]=229
    test_data_sizes[6373][0]=232
    test_data_sizes[6612][0]=225

    for i in range(0,12630):
        test_data_coords[i][0]=int(test_data_coords[i][0] * 32 / test_data_sizes[i][0])
        test_data_coords[i][1]=int(test_data_coords[i][1] * 32 / test_data_sizes[i][1])
        test_data_coords[i][2]=int(test_data_coords[i][2] * 32 / test_data_sizes[i][0])+1
        test_data_coords[i][3]=int(test_data_coords[i][3] * 32 / test_data_sizes[i][1])+1
        crop_img = test_images[i][test_data_coords[i][1]:test_data_coords[i][3], test_data_coords[i][0]:test_data_coords[i][2]]

        test_images[i] = cv2.resize(crop_img,(32,32))

    predicted_test = sess.run([correct_pred], feed_dict={x: test_images})[0]

    # Calculate correct matches
    match_count_test = sum([int(y == y_) for y, y_ in zip(test_data_labels, predicted_test)])
    # Calculate the accuracy
    accuracy_test = match_count_test / len(test_data_labels)
    # Print the accuracy
    print("Accuracy_test: {:.3f}".format(accuracy_test))

```

Result:

After 5000 echoes:

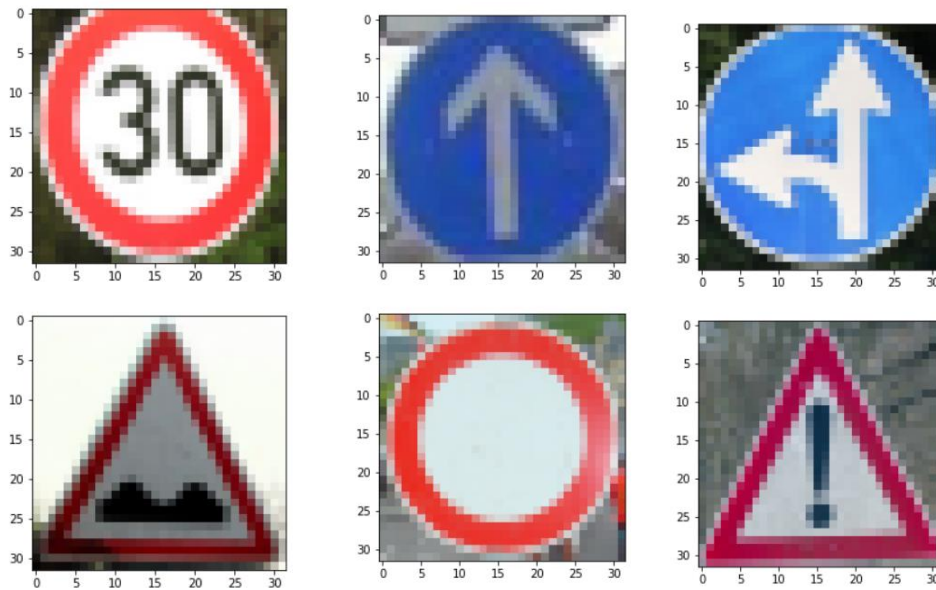
```

Accuracy_valid: 0.951
12630
32
32
Accuracy_test: 0.930
_ _ _ _ _

```

Part 4: using trained model to predict new image from web & provide top 5 predict possibility.

Selected image from website:



- ➔ Image has already been cropped into traffic sign only, and the image quality is not bad as in brightness. size, resolution has already been changed into standard from original image, now is 32*32. So should not be so difficult to predict.

```
import glob

my_images = sorted(glob.glob('./web_image/*.png'))
my_labels = np.array([1, 22, 35, 15, 37, 18])

figures = {}
labels = {}
my_signs = []
index = 0
for my_image in my_images:
    img = cv2.cvtColor(cv2.imread(my_image), cv2.COLOR_BGR2RGB)
    my_signs.append(img)
    figures[index] = img
    labels[index] = name_values[my_labels[index]][1].decode('ascii')
    index += 1

def plot_figures(figures, nrows = 1, ncols=1, labels=None):
    fig, axs = plt.subplots(ncols=ncols, nrows=nrows, figsize=(12, 14))
    axs = axs.ravel()
    for index, title in zip(range(len(figures)), figures):
        axs[index].imshow(figures[title], plt.gray())
        if labels != None:
            axs[index].set_title(labels[index])
        else:
            axs[index].set_title(title)
        axs[index].set_axis_off()

    plt.tight_layout()

my_signs = np.array(my_signs)
my_signs_normalized = rgb2gray(my_signs)
n_train = my_signs.shape[1:]
print(n_train)

m_train = my_signs_normalized.shape[1:]
print(m_train)
```

```

k_size = 5
softmax_logits = tf.nn.softmax(logits)
top_k = tf.nn.top_k(softmax_logits, k=k_size)
keep_prob = tf.placeholder(tf.float32)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    saver.restore(sess, "./lenet")
    my_top_k = sess.run(top_k, feed_dict={x: my_signs_normalized, keep_prob: 1.0})

    for i in range(6):
        figures = {}
        labels = {}
        figures[0] = my_signs[i]
        labels[0] = "Original"
        print('Image {} :'.format(i+1))
        for j in range(k_size):
            print('    Possibility {} : ({:.0f}%)' .format(j+1, 100*my_top_k[0][i][j]))
            labels[j+1] = 'Guess {} : ({:.0f}%)' .format(j+1, 100*my_top_k[0][i][j])
    return savedSnaps

```

Result: (100%)

```

[37 18  1 22 35 15]
[37 18  1 22 35 15]
Image 1 :
    Possibility 1 : (100%)
    Possibility 2 : (0%)
    Possibility 3 : (0%)
    Possibility 4 : (0%)
    Possibility 5 : (0%)
Image 2 :
    Possibility 1 : (100%)
    Possibility 2 : (0%)
    Possibility 3 : (0%)
    Possibility 4 : (0%)
    Possibility 5 : (0%)
Image 3 :
    Possibility 1 : (100%)
    Possibility 2 : (0%)
    Possibility 3 : (0%)
    Possibility 4 : (0%)
    Possibility 5 : (0%)
Image 4 :
    Possibility 1 : (95%)
    Possibility 2 : (3%)
    Possibility 3 : (2%)
    Possibility 4 : (0%)
    Possibility 5 : (0%)
Image 5 :
    Possibility 1 : (100%)
    Possibility 2 : (0%)
    Possibility 3 : (0%)
    Possibility 4 : (0%)
    Possibility 5 : (0%)
Image 6 :
    Possibility 1 : (100%)
    Possibility 2 : (0%)
    Possibility 3 : (0%)
    Possibility 4 : (0%)
    Possibility 5 : (0%)

```

Part 5: Pre-processing techniques used and why these techniques were chosen:

- Con2d layer:

Reason/ Benefit: to reduce input data size, but not impact image information. Weight as a 5*5 matrix. This weight shall now run across the image such that all the pixels are covered at least once, to give a convolved output. The weight matrix behaves like a filter in an image extracting particular information from the original image matrix. Therefore weights are learnt to extract features from the original image which help the network in correct prediction.

- max_pooling2d layer:

Reason/ Benefit: Pooling is done for purpose of reducing the spatial size of the image. And also independently on each depth dimension, the depth of the image remains unchanged. The most common form of pooling layer generally applied is the max pooling.

Part 6: Summary

These are the steps I could think of for project 2, as a base line. Still a lot of improvement needed to achieve a better accuracy.

The key area could be improved is:

1. Original data expansion

if we could enlarge original data, it would be very helpful, like project spec recommended, rotation, translation, zoom, flips, and/or colour perturbation etc.

2. Model architecture enhancement

- how many levels of layer?
- how many units for each level of layer?
- dropout rate? here to be 0.5
- etc.

3. In the end, after 5000 echoes, validation accuracy becomes 95.1% using desktop GPU.