自動駕駛實務 #2 Report

Department	Name	Student ID
電機112	謝宗翰	E14084078

● 遇到的困難

⚠️ 問題1. 套件安裝問題：

在conda環境下，在jupyter notebook上import sklearn時，以確認在該環境下已安裝sklearn的情況下，會持續報 “No module name sklearn” 的錯誤。

☑解決辦法：

使用Stackoverflow上前輩們提供的方法皆無法改善，後來經測試發現，原來是我numpy跟scipy的版本太高(1.18.5)，sklearn不支援這麼高的版本，將numpy降版本至1.13.0後就可以正常import了。

另外，因為上次寫作業也常常卡在安裝套件版本的問題，所以接下來我都使用Colab來編寫程式，省去要安裝套件與卡在套件版本的問題。

⚠️ 問題2. LeNet 輸出矩陣大小問題

因為這是我第一次寫CNN架構的AI模型，在各種Shape的地方卡了很久。原本老師所提供的PPT中的LeNet範例中，最後輸出的結果的大小是84*10，與我們這次要訓練模型的Label數量不一致，導致後面實際訓練時瘋狂報shape不相符的錯誤。

```
InvalidArgumentError: logits and labels must be broadcastable: logits_size=[128,10] labels_size=[128,43]
        [[{{node softmax_cross_entropy_with_logits_sg_12}}]]
```

☑解決辦法：

與同學討論後才發現，LeNet的輸出大小，需要根據Label數(y_train)而定，後來把LeNet中的y向量改成n_class這部分就沒問題了。

```
# SOLUTION: Layer 5: Fully Connected. Input = 84. Output = 10.
fc3_w = tf.Variable(tf.truncated_normal(shape=(84, 10), mean=mu, stddev=sigma))
fc3_b = tf.Variable(tf.zeros(10))
logits = tf.matmul(fc2, fc3_w) + fc3_b    # fc2*fc3_w + fc3_b
return logits
```

```
# SOLUTION: Layer 5: Fully Connected. Input = 84. Output = n_classes.
fc3_w = tf.Variable(tf.truncated_normal(shape=(84, n_classes), mean=mu, stddev=sigma))
fc3_b = tf.Variable(tf.zeros(n_classes))
logits = tf.matmul(fc2, fc3_w) + fc3_b    # fc2*fc3_w + fc3_b
return logits
```

⚠️ 問題3. Shuffle 函式使用問題

```
for i in range(EPOCHS):
    x_train = shuffle(x_train)
    y_train = shuffle(y_train)
    for offset in range(0, example_num, BATCH_SIZE):
        end = offset + BATCH_SIZE
        batch_x = x_train[offset:end]
        batch_y = y_train[offset:end]
```

```
...  Training...Please wait
     EPOCH 1 ...
     Validation Accuracy = 0.060

     EPOCH 2 ...
     Validation Accuracy = 0.044
```

在使用 sklearn 中的 Shuffle 函式時，不可以像上圖那樣寫。他與random中的Shuffle函式不太一樣，他是將 X 和 Y 一一對應後打亂，在亂數排列，上圖的結果會使Valid accuracy降低非常非常多，低至0.044左右，學習取線不升反降，這部分我卡了8小時的時間Debug，需特別注意。

☑ 解決辦法：

```
for i in range(EPOCHS):
    x_train, y_train = shuffle(x_train, y_train)
    for offset in range(0, example_num, BATCH_SIZE):
        end = offset + BATCH_SIZE
        batch_x = x_train[offset:end]
        batch_y = y_train[offset:end]
```

```
Training...Please wait
EPOCH 1 ...
Validation Accuracy = 0.173

EPOCH 2 ...
Validation Accuracy = 0.329

EPOCH 3 ...
Validation Accuracy = 0.482

EPOCH 4 ...
Validation Accuracy = 0.534

EPOCH 5 ...
Validation Accuracy = 0.610
```

正確使用Shuffle函式，使 X 和 Y 一一對應。Valid accuracy 從0.044直接上升到合理範圍，且學習取線變得更漂亮也更合理。

⚠️ 問題4. 資料前處理的Shape問題

因為我設定的 x 的 placeholder 是 [?, 32, 32, 1] ，但是經過資料前處理的 training data只有[?, 32, 32] 比placeholder 少了一維，導致模型與訓練資料大小接不起來。

```
Training...Please wait
---------------------------------------------------------------------------
InvalidArgumentError                      Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/tensorflow/python/client/session.py in _do_call(self, fn, *args)
   1376     try:
-> 1377       return fn(*args)
   1378     except errors.OpError as e:

                                 6 frames
InvalidArgumentError: logits and labels must be broadcastable: logits_size=[128,10] labels_size=[393216,43]
        [[{{node softmax_cross_entropy_with_logits_sg_1}}]]

During handling of the above exception, another exception occurred:

InvalidArgumentError                      Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/tensorflow/python/client/session.py in _do_call(self, fn, *args)
   1394                     '\nsession_config.graph_options.rewrite_options.'
   1395                     'disable_meta_optimizer = True')
-> 1396       raise type(e)(node_def, op, message)  # pylint: disable=no-value-for-parameter
   1397
   1398   def _extend_graph(self):

InvalidArgumentError: Graph execution error
```

☑ 解決辦法：

使用numpy的expand_dims的函式來將處理過的資料再擴大一個維度。

```python
def pre_process_image(image):

    image_gray = np.mean(image, axis=3)
    image_gray = np.expand_dims(image_gray, axis=3)
    image_norm = (image_gray - 128)/128

    return image_norm
```

或

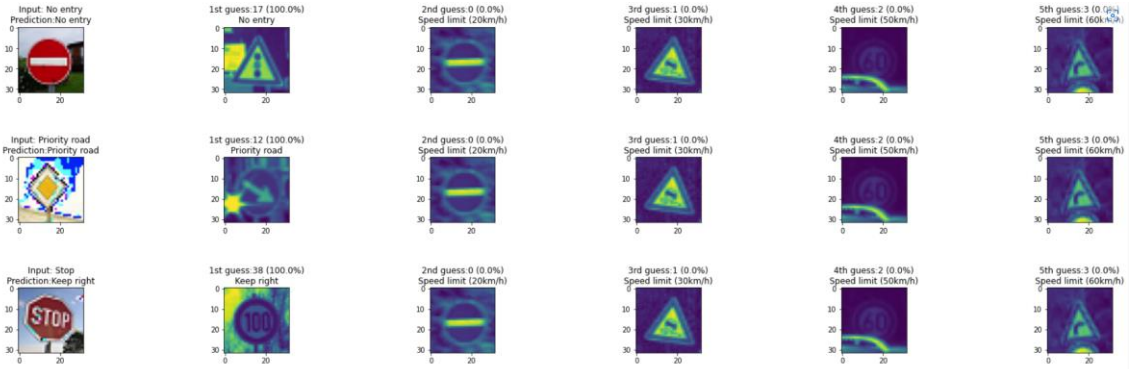
```python
import cv2
import tensorflow.compat.v1 as tf
def grayAndEqu(img):
    gray = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
    equ = cv2.equalizeHist(gray)
    return equ

x_train = np.array([grayAndEqu(image) for image in x_train])
x_test = np.array([grayAndEqu(image) for image in x_test])
x_train = np.expand_dims(x_train,axis = 3)
x_valid = np.expand_dims(x_valid,axis = 3)
x_test = np.expand_dims(x_test,axis = 3)
```

以上兩種方法，經測試都可以解決Shape上的問題。

p.s. 剩下的或要補充的皆在下面有更詳細的說明

⚠️ 問題5. 尚未解決的未知問題

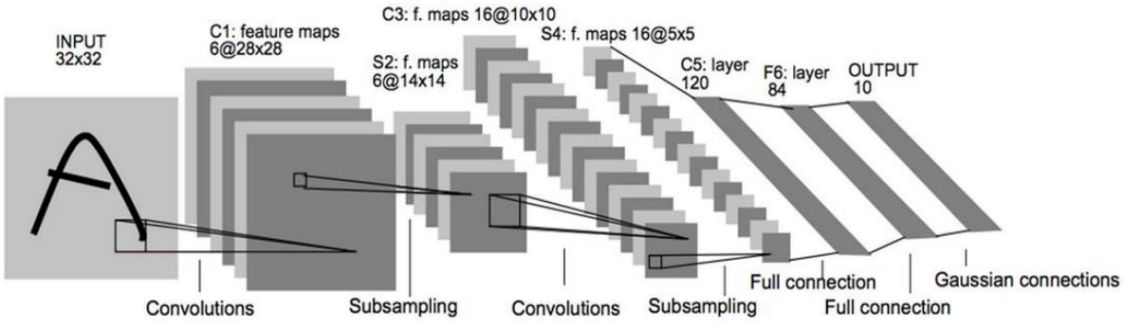
在使用Model Architecture 時，使用 model.predict() 函式時，有時會有資料沒有訓練到的問題，跑出來的predict值有時全部都是0 ，但有時候卻又可以正常運作，此部分尚未找到bug，發生此情況的概率大概是10%左右。



只有第一欄是可以正確預測的，後面的圖片都抓不到預測值，待改進此bug。

- **使用模型架構與參數調整**

此次使用的模型是Lenet



上圖是Lenet的基本的架構，此次Project的重點是將此架構用程式碼實現出來。

I. 第一層Convolution

```
def LeNet(x):
    # Arguments used for tf.truncated_normal, randomly defines variables for the weights and bias
    mu = 0
    sigma = 0.1
    # SOLUTIQN: Layer 1: Convolutional. Input = 32x32x1. Output = 28x28x6.

    # filter size is 5*5
    # filter depth is 1
    # filter output is 6
    # tf.truncated_normal(大小 = 5*5*1*6的矩陣, mean=平均值, stddev=標準差)
    filters = tf.truncated_normal(shape=(5, 5, 1, 6), mean=mu, stddev=sigma)
    initial = tf.zeros(6)    # 因為 filter 的 output 需要是 6

    # 訓練 Filter
    # 需要去學習的權重/加權(weight) -> tf.Variable()內的變數為"可求導的變數" -- 求 weight
    conv1_w = tf.Variable(filters)
    # 需要去學習的權重/加權(weight) -> tf.Variable()內的變數為"可求導的變數" -- 求 bias
    conv1_b = tf.Variable(initial)

    # CNN第一層網路 (第一層捲積)
    conv1 = tf.nn.conv2d(x, conv1_w, strides=[1, 1, 1, 1], padding='VALID') + conv1_b
    # SOLUTION: Activation.
    conv1 = tf.nn.relu(conv1)
    # SOLUTION: Pooling. Input = 28x28x6. Output = 14x14.x6.
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')
```

意在要訓練出一個5*5大小，深度為 1，且輸出為 6 的filter，接著使用tensorflow的函式去打造 weight 跟 bias 。因為步長的關係，捲積會超出大小，所以會需要使用padding補1。最後再取Relu後，做pooling使縮小矩陣丟給第二層捲積。

II. 第二層Convolution

```
# SOLUTION: Layer 2: Convolutional. Output = 10x10x16.
conv2_w = tf.Variable(tf.truncated_normal(shape=(5, 5, 6, 16), mean=mu, stddev=sigma))
conv2_b = tf.Variable(tf.zeros(16))
conv2 = tf.nn.conv2d(conv1, conv2_w, strides=[1, 1, 1, 1], padding='VALID') + conv2_b
# SOLUTION: Activation.
conv2 = tf.nn.relu(conv2)
# SOLUTION: Pooling. Input = 10x10x16. Output = 5x5x16.
conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')

# SOLUTION: Flatten. Input = 5x5x16. Output = 400.
fc0 = tf.compat.v1.layers.flatten(conv2)
```

第二層捲積層的概念與第一層差不多，進行動作的順序都一樣，只是多一個攤平的動作，讓向量展開。

求出 weight 及 bias → padding → Relu → Pooling → Flatten → 丟給第三層

III. 第三層與第四層Convolution

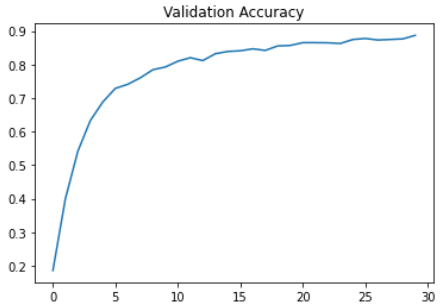
```
# SOLUTION: Layer 3: Fully Connected. Input = 400. Output = 120.
fc1_w = tf .Variable(tf.truncated_normal(shape=(400, 120), mean=mu, stddev=sigma))
fc1_b = tf.Variable(tf.zeros(120))
fc1 = tf.matmul(fc0, fc1_w) + fc1_b
# SOLUTION: Activation.
fc1 = tf.nn.relu(fc1)

# SOLUTION: Layer 4: Fully Connected. Input = 120. Output = 84.
fc2_w = tf.Variable(tf.truncated_normal(shape=(120, 84), mean=mu, stddev=sigma))
fc2_b = tf.Variable(tf.zeros(84))
fc2 = tf.matmul(fc1, fc2_w) + fc2_b
# SOLUTION: Activation.
fc2 = tf.nn.relu(fc2)
```

第三層的構造也很簡單，基本上與第一層一樣

IV. 第五層Convolution

```
# SOLUTION: Layer 5: Fully Connected. Input = 84. Output = n_classes.
fc3_w = tf.Variable(tf.truncated_normal(shape=(84, n_classes), mean=mu, stddev=sigma))
fc3_b = tf.Variable(tf.zeros(n_classes))
logits = tf.matmul(fc2, fc3_w) + fc3_b    # fc2*fc3_w + fc3_b
```

這層就是單純的把訓練完的 weight 與 bias 用線性組合串起來後輸出出去，值得注意的事，n_class 為 y_train 的長度(43 labels)。

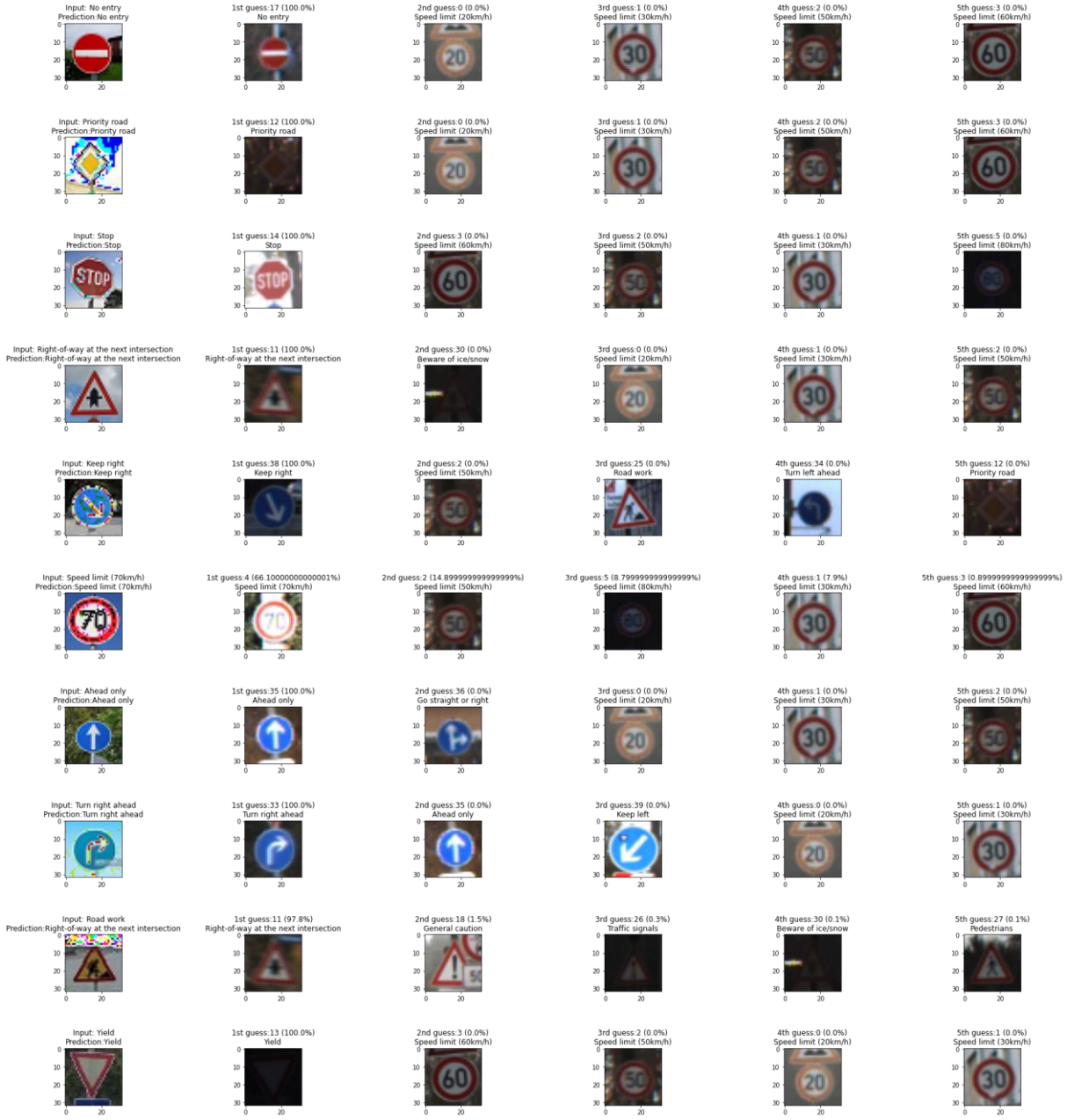


此圖為 LeNet 的學習曲線，從0.136成長至0.86，成長幅度很大。

額外補充：

此次Project除了使用Lenet之外，我還嘗試了老師提供的 Model construction，效果比Lenet好蠻多的，Lenet的最佳預測結果是86%左右，而Model construction的預測準確度高達96.6%，效果十分顯著。

- **交通號誌分類結果**



I. 預測準確度

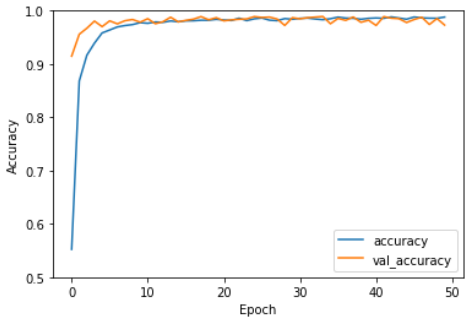
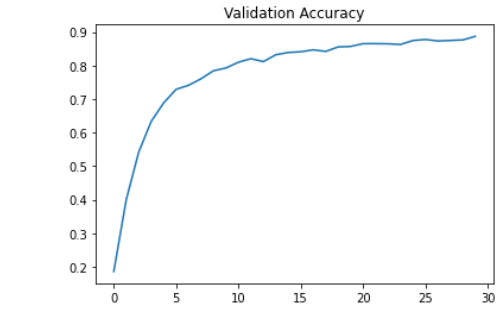
```
with tf.Session() as sess:
    saver.restore(sess, tf.train.latest_checkpoint('.'))

    train_accuracy = evaluate(x_train, y_train)
    print("Train Accuracy = {:.3f}".format(train_accuracy))

    valid_accuracy = evaluate(x_valid, y_valid)
    print("Valid Accuracy = {:.3f}".format(valid_accuracy))

    test_accuracy = evaluate(x_test, y_test)
    print("Test Accuracy = {:.3f}".format(test_accuracy))
```

```
INFO:tensorflow:Restoring parameters from ./Lenet
Train Accuracy = 0.972
Valid Accuracy = 0.887
Test Accuracy = 0.860
```

● 使用工具

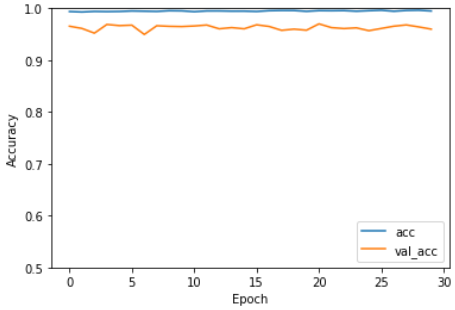
1. 環境工具與IDE：Jupyter notebook、anaconda → Colab
2. 涵式工具：NumPy, sklearn,tensorflor , matplotlib, cv2

● 結果比較

Model Architecture	LeNet
	

可看出Model Architecture 從一開始就比較高，且曲線接近 e。而 LeNet 的學習落差比較大，但最後的成效與 Model Architecture 差不多，都接近 0.96。

注意，在訓練時每次都要重新跑過 Colab，不然會 Training 的東西會重複使用，導致結果變成這樣



● 心得與討論

這次的Project2真的有在訓練模型的感覺了，靠自己寫出一個DNN網路蠻有成就感的，過程中卡了很多莫名其妙的錯誤，覺得自己最粗心的地方應該是在load .p檔時，路徑都用複製的，結果 x_train用到 x_test 的data，訓練資料瞬間少一半，導致 Validation accuracy 少了不少。

再來就是還是會有持續遇到的問題，就是矩陣的大小，在Project1 時其實就遇到了不少關於Shape方面的問題，常常在這邊就卡個老半天，要把模型接起來其實也不容易。

最後也很感謝助教跟老師在課堂上回答我關於Project的問題，不然這兩個禮拜我是無法順利完成的此次Project的，非常感謝。

最後附上Colab連結：
https://colab.research.google.com/drive/1xqS-gI89Hf8pyKyciw7mFLfm8-pSfGBH?usp=sharing