Project02_TrafficSign Classifier

P16084251 系統所 周禮宏

● Project 目標

透過**交通號誌**的影像資料集來訓練 CNN 網路架構 **LeNet5** 使其能對影像中交通號誌進行**分類**。

● 採用的資料集

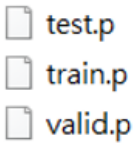
Project 採用 **The German Traffic Sign Benchmark** 的資料集，其特色如下:

1. 單一影像資料中含有單一交通號誌
2. 超過 40 種不同交通號誌
3. 影像資料超過 50,000 筆
4. 影像資料貼近現實場景

本 Project 將資料集**切割**成三部分:

1. 訓練資料集: 共 34799 筆
2. 驗證資料集: 共 4410 筆
3. 測試資料集: 共 12630 筆

資料集檔案如下圖所示:



- **Code 的詳細描述解說**

Step 0: Load The Data

Download the dataset & unzip

In [1]:
```
import pickle
training_file = "train.p"
validation_file = "valid.p"
testing_file = "test.p"

with open(training_file, mode='rb') as f:
    train = pickle.load(f)
with open(validation_file, mode='rb') as f:
    valid = pickle.load(f)
with open(testing_file, mode='rb') as f:
    test = pickle.load(f)

x_train, y_train = train['features'], train['labels']
x_valid, y_valid = valid['features'], valid['labels']
x_test, y_test = test['features'], test['labels']
print("x_train shape:", x_train.shape)
print("y_train shape:", y_train.shape)
print("x_valid shape:", x_valid.shape)
print("y_valid shape:", y_valid.shape)
print("X_test shape:", x_test.shape)
print("y_test shape:", y_test.shape)
```

```
x_train shape: (34799, 32, 32, 3)
y_train shape: (34799,)
x_valid shape: (4410, 32, 32, 3)
y_valid shape: (4410,)
X_test shape: (12630, 32, 32, 3)
y_test shape: (12630,)
```

此區塊使用 pickle 套件來引入本 Project 使用的資料集，並顯示其維度。

從結果可以看出資料集為 4 維，
其意義為:（資料筆數, 影像寬, 影像高, RGB 三維空間）

由於 pickle 套件本身便是 Python 當中內建的模組，所以我們不需要使用額外的指令來安裝它，直接在程式中匯入即可。

Step 1: Dataset Summary & Exploration

Provide a Basic Summary of the Data Set Using Python, Numpy and/or Pandas

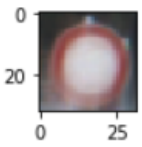
In [2]:
```
import numpy as np
n_train = len(x_train)
n_test = len(x_test)
image_shape = x_train[0].shape
n_classes = len(np.unique(y_train))
print("Number of training examples = ", n_train)
print("Number of testing examples = ", n_test)
print("Image data shape = ", image_shape)
print("Number of classes = ", n_classes)
```

```
Number of training examples =  34799
Number of testing examples =  12630
Image data shape =  (32, 32, 3)
Number of classes =  43
```

套件版本： Numpy 1.16.4
此區塊為顯示訓練資料集與測試資料集筆數、單一影像的維度以及資料集中的交通號誌種類數量。

In [3]:

```
import matplotlib.pyplot as plt
import random
%matplotlib inline

index = random.randint(0,len(x_train))
image = x_train[index].squeeze()

plt.figure(figsize=(1,1))
plt.imshow(image)
```



套件版本: Matplotlib 3.1.0

此區塊為顯示訓練資料集中隨機的一張影像資料。

plt.show() 用來顯示影像

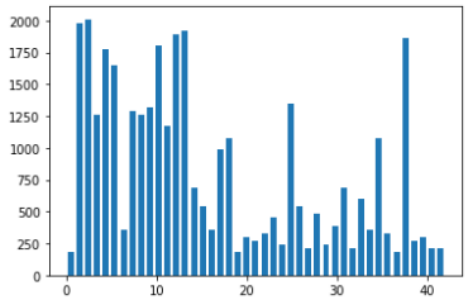
In [5]:

```
show_each_class = {}

for label in range(43):
    index = 0
    for data_label in y_train:
        if data_label == label:
            image = x_train[index]
            show_each_class[label] = image
            break
        index += 1

fig, axs = plt.subplots(4,11,figsize = (15,7))
plt.subplots_adjust(wspace =0.2, hspace =0)
axs = axs.ravel()

for i in range(44):
    axs[i].axis('off')

for i in range(len(show_each_class)):
    image = show_each_class[i]
    axs[i].imshow(image)
    axs[i].set_title(i)
```

此區塊為顯示資料集中各類各一筆的影像資料，讓我們更了解資料集的全貌。

plt.subplots() 使多個影像可以用子圖一同呈現，由於共有 43 類，我使用 4(row) x 11(column) 的呈現。

In [6]:
```
hist, bins = np.histogram(y_train, bins = n_classes)
width = 0.7 * (bins[1] - bins[0])
center = (bins[:-1] + bins[1:]) / 2
plt.bar(center, hist, align = "center", width = width)
plt.show()
```



此區塊顯示訓練資料集各種類的資料筆數。
plt.bar() 可將資料以長直條圖表呈現。

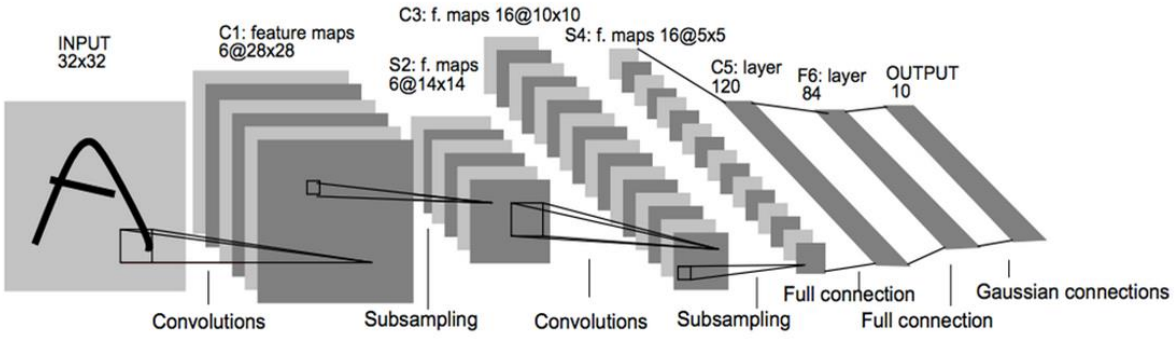
Step 2: Design and Test a Model Architecture

Pre-process the Data Set (normalization, grayscale, etc.)

In [31]:
```
import tensorflow as tf
from tensorflow.keras import Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.layers import Dense, Flatten, Conv2D, AveragePooling2D

class LeNet(Sequential):
    def __init__(self, input_shape, nb_classes):
        super().__init__()

        self.add(Conv2D(6, kernel_size=(5, 5) , strides=(1, 1),activation='tanh', input_shape=input_shape, padding='valid'))
        self.add(AveragePooling2D())
        self.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='tanh', padding='valid'))
        self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
        self.add(Flatten())
        self.add(Dense(120, activation='tanh'))
        self.add(Dense(84, activation='tanh'))
        self.add(Dense(nb_classes, activation='softmax'))

        # 測試 learning_rate 0.001 0.01 0.1

        opt = tf.keras.optimizers.Adam(learning_rate=0.001)

        self.compile(optimizer= opt,
                     loss='sparse_categorical_crossentropy',
                     metrics=['accuracy'])
```

套件版本: tensorflow 2.0.0

使用 tensorflow.keras 建立 CNN 網路架構 **LeNet5** 如下圖所示:



Layer1 (卷積層)

目的:

對資料提取特徵

程式碼:

```
self.add(Conv2D(6, kernel_size=(5, 5) , strides=(1, 1),activation='tanh',
input_shape=input_shape, padding='valid'))
```

參數說明:

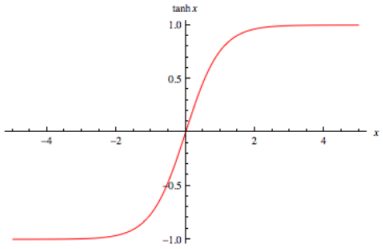
filters：卷積核的數目，即輸出維度，此設 6

kernel_size: 卷積核大小，此設 5x5

strides: 卷積的步數，此設 1

activation: 激勵函數，此用 tanh，如下圖所示

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$



input_shape: 輸入的維度，此為(32,32,3)

padding: 填充，此設 valid(不採用)

Layer2 (平均池化層)

目的:

降低資料維度

程式碼:

```
self.add(AveragePooling2D())
```

參數說明:

pool_size：池化的大小，默認為 2

strides: 池化的步數，默認為 2

padding: 填充，此設 valid(不採用)

Layer3 (卷積層)

目的:

對資料提取特徵

程式碼:

```
self.add(Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activation='tanh',
padding='valid'))
```

參數說明:

filters：卷積核的數目，即輸出維度，此設 16

kernel_size: 卷積核大小，此設 5x5

strides: 卷積的步數，此設 1

activation: 激勵函數，此用 tanh

padding: 填充，此設 valid(不採用)

Layer4 (平均池化層)

目的:

降低資料維度

程式碼:

```
self.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding='valid'))
```

參數說明:

pool_size：池化的大小，此設 2

strides: 池化的步數，此設 2

padding: 填充，此設 valid(不採用)

Layer5 (平坦層)

目的:

將矩陣打平成一維的陣列作為輸入

程式碼:

```
self.add(Flatten())
```

Layer6 (全連接層)

目的:

使用多層感知器進行判斷

程式碼:

```
self.add(Dense(120, activation='tanh'))
```

參數說明:

units：神經元數量，此設 120

activation: 激勵函數，此用 tanh

Layer7 (全連接層)

目的:

使用多層感知器進行判斷

程式碼:

```
self.add(Dense(84, activation='tanh'))
```

參數說明:

units：神經元數量，此設 84

activation: 激勵函數，此用 tanh

Layer8 (輸出層)

目的:

輸出判斷結果，將每個輸入轉化成各類別的機率

程式碼:

```
self.add(Dense(nb_classes, activation='softmax'))
```

參數說明:

units：神經元數量，此設 43，即交通號誌的類別總數

activation: 激勵函數，此用 softmax

訓練模型配置

程式碼:

```
opt = tf.keras.optimizers.Adam(learning_rate=0.001)
self.compile(optimizer= opt,
             loss='sparse_categorical_crossentropy',
             metrics=['accuracy'])
```

參數說明:

optimizer：優化器，在此使用 Adam，learning rate 設 0.001

loss: 損失函數，此用 sparse_categorical_crossentropy

metrics: 在訓練和測試期間的模型評估標準。

```
In [32]: 1 model = LeNet(x_train[0].shape, n_classes)
```

此區塊為將訓練資料集的維度及類別總數放入 CNN 網路架構 LeNet5。

```
In [33]: model.summary()
```

```
Model: "le_net_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 28, 28, 6)         456
_________________________________________________________________
average_pooling2d_8 (Average (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_9 (Average (None, 5, 5, 16)          0
_________________________________________________________________
flatten_4 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_12 (Dense)             (None, 120)               48120
_________________________________________________________________
dense_13 (Dense)             (None, 84)                10164
_________________________________________________________________
dense_14 (Dense)             (None, 43)                3655
=================================================================
Total params: 64,811
Trainable params: 64,811
Non-trainable params: 0
_________________________________________________________________
```

此區塊為顯示 CNN 網路架構 LeNet5 的詳細資訊，包含各層輸出維度及參數數量。

```
In [34]:
```

```python
import datetime
import os

log_dir = os.path.join(
    "logs",
    "fit",
    datetime.datetime.now().strftime("%Y%m%d-%H%M%S"),
)

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
```

此區塊為建立回調參數，以便日後查看訓練模型的內在狀態和統計。
TensorBoard 是由 Tensorflow 提供的視覺化工具。

程式碼:

`tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)`

參數說明:

log_dir: 用來保存被 TensorBoard 分析的檔名

histogram_freq: 對於模型中各層計算啟動值和模型權重長條圖的頻率

In [35]:
```
model.fit(x_train, y=y_train,
          epochs=20,
          batch_size=128,
          validation_data=(x_valid, y_valid),
          callbacks=[tensorboard_callback],
          verbose=1)
# verbose:  0 = silent, 1 = progress bar, 2 = one line per epoch
```

```
Train on 34799 samples, validate on 4410 samples
Epoch 1/20
  128/34799 [..............................] - ETA: 1:48 - loss: 3.9343 - accuracy: 0.0078WARNING:tensorflow:Method (on_train_batch_end) is slow compared to the batch update (0.116122). Check your callbacks.
34799/34799 [==============================] - 11s 321us/sample - loss: 1.2235 - accuracy: 0.6994 - val_loss: 0.5466 - val_accuracy: 0.8474
Epoch 2/20
34799/34799 [==============================] - 10s 290us/sample - loss: 0.2783 - accuracy: 0.9384 - val_loss: 0.3766 - val_accuracy: 0.8918
Epoch 3/20
34799/34799 [==============================] - 10s 297us/sample - loss: 0.1525 - accuracy: 0.9673 - val_loss: 0.2965 - val_accuracy: 0.9172
Epoch 4/20
34799/34799 [==============================] - 10s 291us/sample - loss: 0.1100 - accuracy: 0.9752 - val_loss: 0.2812 - val_accuracy: 0.9156
Epoch 5/20
34799/34799 [==============================] - 10s 291us/sample - loss: 0.0784 - accuracy: 0.9826 - val_loss: 0.2487 - val_accuracy: 0.9297
Epoch 6/20
34799/34799 [==============================] - 11s 308us/sample - loss: 0.0557 - accuracy: 0.9880 - val_loss: 0.2323 - val_accuracy: 0.9331
Epoch 7/20
34799/34799 [==============================] - 11s 311us/sample - loss: 0.0443 - accuracy: 0.9908 - val_loss: 0.2442 - val_accuracy: 0.9286
Epoch 8/20
34799/34799 [==============================] - 10s 300us/sample - loss: 0.0347 - accuracy: 0.9934 - val_loss: 0.2209 - val_accuracy: 0.9354
```

此區塊為訓練模型。

程式碼:

```
model.fit(x_train, y=y_train,
          epochs=20,
          batch_size=128,
          validation_data=(x_valid, y_valid),
          callbacks=[tensorboard_callback],
          verbose=1)
```

參數說明:

x: 訓練資料集

y: 訓練集標籤

epochs: 訓練的總次數，此設 20

batch_size: 指定進行梯度下降時每個 batch 包含的樣本數，此設 128

validation_data: 驗證數據集

callbacks: 回調函數

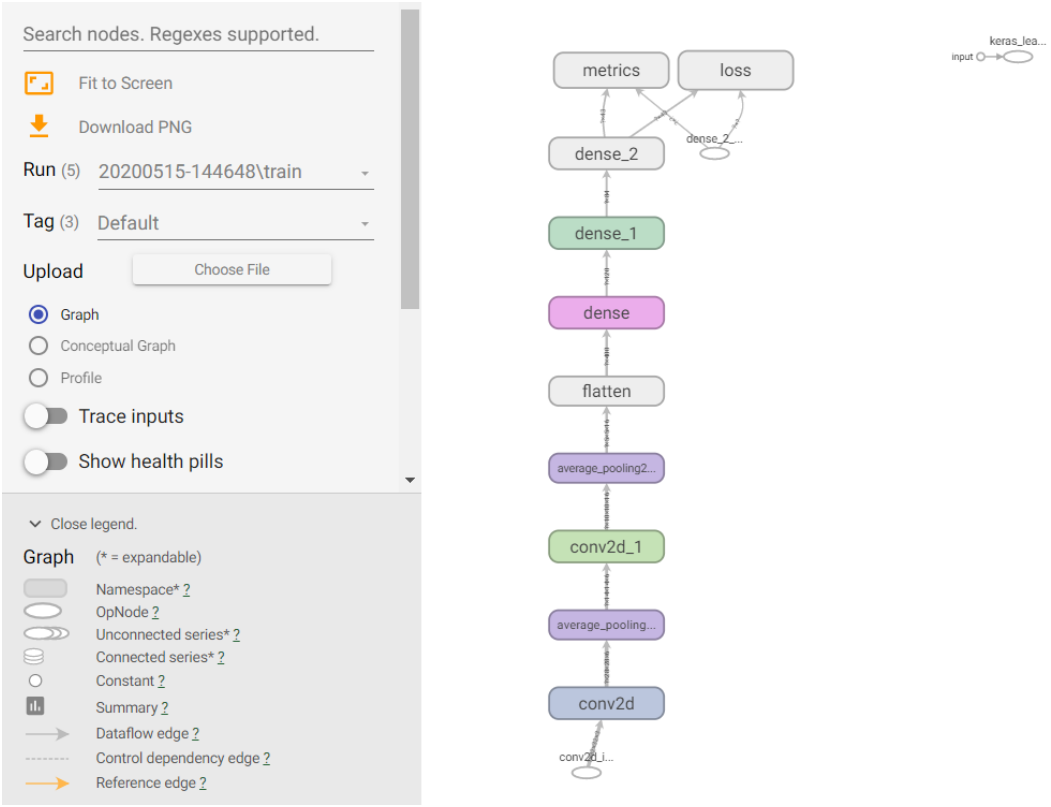
verbose: 設 1 為輸出進度條記錄

In [23]:
```
%load_ext tensorboard
```

In [25]:
```
# 解決 localhost拒絕連線
# remove the .tensorboard-info directory (C:\Users\username\AppData\Local\Temp\.tensorboard-info )
%tensorboard --logdir logs/fit
```

```
Reusing TensorBoard on port 6006 (pid 7260), started 0:01:41 ago. (Use '!kill 7260' to kill it.)
```

此區塊為開啟 tensorboard，並接上訓練好的資料紀錄



In [36]:
```
model.save('path_to_saved_model', save_format='tf')
```

```
WARNING:tensorflow:From C:\Users\NCKU KCLee\Anaconda3\lib\site-packages\tensorflow_core\python\ops\resource_variable_ops.py:178
1: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and w
ill be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
INFO:tensorflow:Assets written to: path_to_saved_model\assets
```

此區塊為儲存訓練好的模型。

Step 3: Test a Model on New Images

In [38]:
```
# 載入模型並建立模型物件（不需要模型的類別宣告原始碼，即可完成）
loaded_model = tf.keras.models.load_model('path_to_saved_model')

# 不除255會有error
x_test_normalize = x_test / 255

prediction_values = loaded_model.predict_classes(x_test_normalize)
```

此區塊為對測試資料集進行分類預測。

tf.keras.models.load_model() 為載入訓練好的模型

loaded_model.predict_classes(x_test_normalize) 對測試資料集進行分類預測

In [47]:

```
fig = plt.figure(figsize=(15, 7))
fig.subplots_adjust(left=0, right=0.8, bottom=0, top=1, hspace=0.6, wspace=0.05)

for i in range(50):
    ax = fig.add_subplot(5, 10, i + 1, xticks=[], yticks=[])
    ax.imshow(x_test_normalize[i,:])
    title = 'pred= ' + str(prediction_values[i])

    if prediction_values[i] == y_test[i]:
        title += '(o)'
        title += '\nlabel= ' + str(y_test[i])
        ax.set_title(title,fontsize = 12)
    else:
        title += '(x)'
        title += '\nlabel= ' + str(y_test[i])
        ax.set_title(title,fontsize = 12, color = "r")

scores = loaded_model.evaluate(x_test_normalize,y_test, verbose=0)
print('test accuracy = ',str(scores[1]))
```

此區塊為顯示本模型在測試資料集的正確率，及測試資料集的前 50 筆資料預測成效如何，若預測成功則以黑色顯示，否則以紅色顯示。

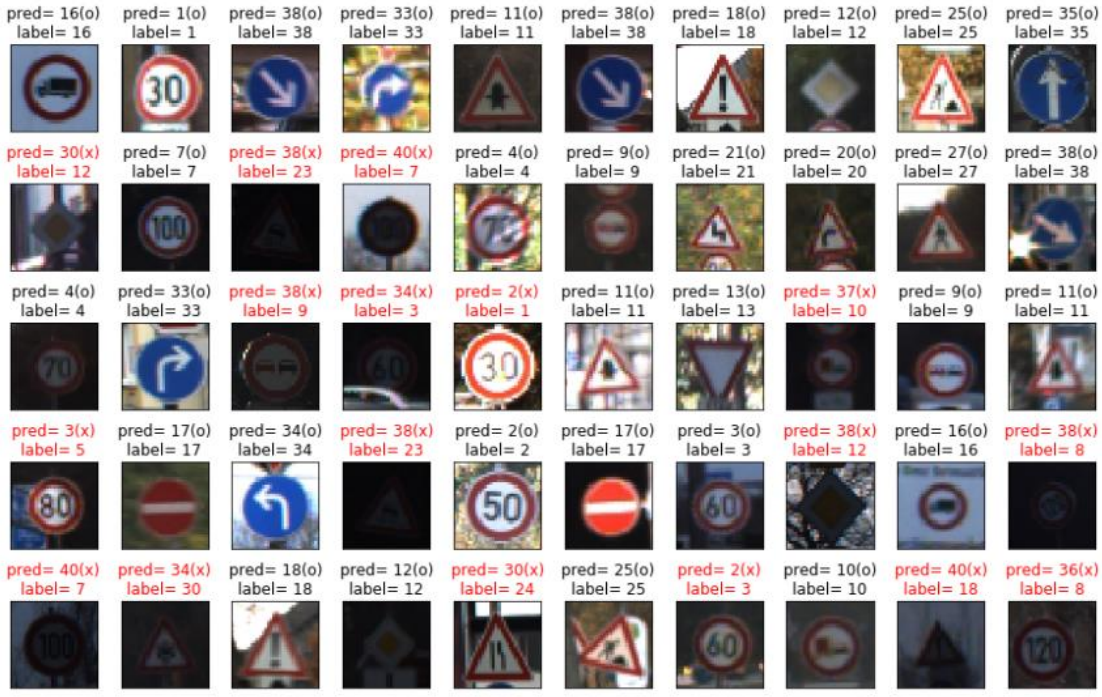
程式碼:

loaded_model.evaluate(x_test_normalize,y_test, verbose=0)

說明:

計算在輸入資料上模型的數據資料，在此求預測正確率。

```
test accuracy =  0.6252573
```

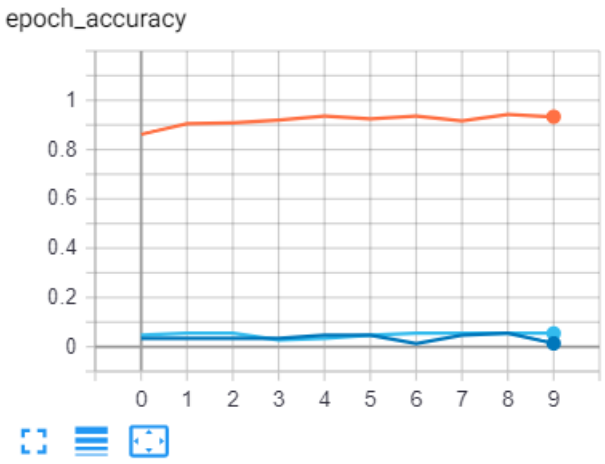


- **參數調整比較（使用驗證資料集的紀錄）**

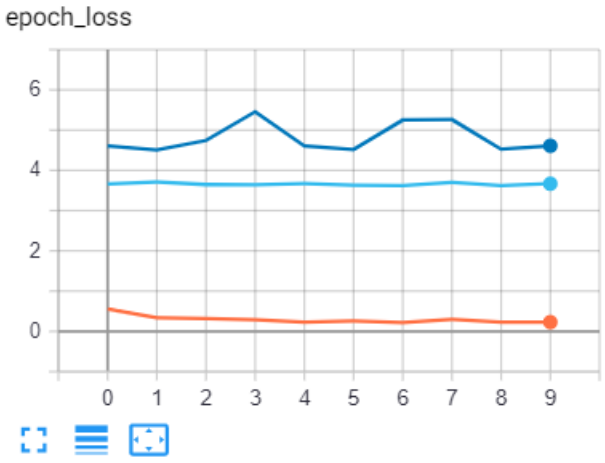
Learning rate 不同

在 epochs 固定(10)下，改變 learning rate

epoch_accuracy



epoch_loss



深藍色: learning rate = 0.1

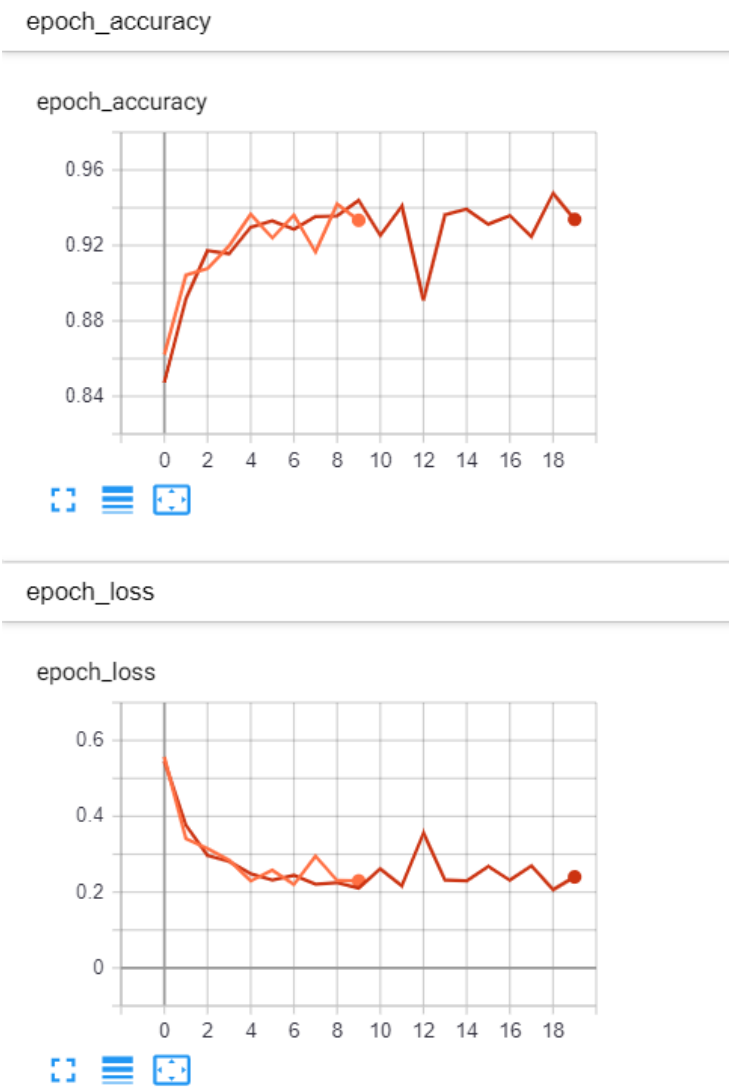
淺藍色: learning rate = 0.01

橘色: learning rate = 0.001

從圖中可以看出 learning rate 在 0.001(橘色)有最好的訓練結果。

Epochs 不同

在 learning rate 固定(0.001)下，改變 epochs



橘色: epochs = 10
深紅色: epochs = 20
從結果來看 epochs = 10 就已取得很好的訓練結果，epochs = 20 的訓練過程甚至出現 overfitting 的現象。

- **Github 完整程式碼 :**

https://github.com/howard1352h/TrafficSign-Classifier/blob/master/P2_final.ipynb

- **可以改善之處**

雖然在訓練資料集和驗證資料集上都可以得到高達 90%的正確率，然而在測試資料集只有不到 70%的正確率，個人推斷 overfitting 造成的，也許在參數調整上做更多的嘗試可以改善這個問題。