自動駕駛實務

Project_2
Traffic Sign Classifier

電機所 碩一
廖聲宇
N26121143

程式碼說明

1.Import Library

```
from sklearn.utils import shuffle
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.models import load_model
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import tensorflow as tf
import pickle
import numpy as np
import random
import pandas as pd
import cv2
import glob
```

引入會使用到的函式庫。

2. 數據集影像的讀取及資料

```
training_file = 'traffic-signs-data/train.p'
validation_file='traffic-signs-data/valid.p'
testing_file = 'traffic-signs-data/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)

# #
# Number of training example
n_train = x_train.shape[0]
# Number of valid example
n_valid=x_valid.shape[0]
# Number of testing examples.
n_test = x_test.shape[0]
image_shape = [x_train.shape[1],x_train.shape[2],x_train.shape[3]]
```

第 1~9 行為讀取模型需要的訓練集、驗證集及測試集。第 11~13 行為分離出訓練集、驗證集及測試集的影像數據與標籤名，並分別命名代號 x 與 y，並在第 14~19 行輸出個影像集的與標籤名的數量以及影像的大小與維度。第 22~27 行輸出各影像集的影像數量，第 28 行獲得影像的長、寬像素大小與影像的通道數。

```
def getLabelsCount(labels):
    d = dict(zip(labels, [0] * len(labels)))
    for x in labels:
        d[x] += 1
```

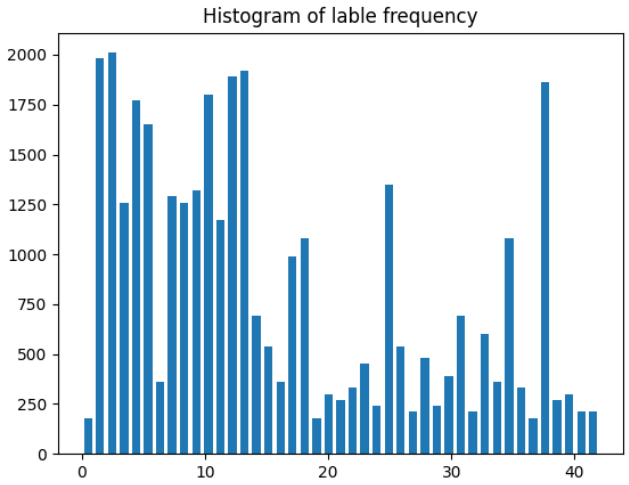
```
    return d
signsDicts = getLabelsCount(y_train)
n_classes = len(signsDicts)

print("Number of training examples =", n_train)
print("Number of validation examples =", n_valid)
print("Number of testing examples =", n_test)
print("Image data shape =", image_shape)
print("Number of classes =", n_classes)
```

第 1~6 行定義獲得標籤名的數量函示，並在第 6、7 行使用函式獲得標籤名數量，及訓練集的種類數量。在第 9~13 行輸出上面獲得到的各項資訊。

3. 讀取訓練集標籤名

```
def getSignNamesData():
    return pd.read_csv('./signnames.csv').values
    #return pd.read_csv('./signnames.csv').as_matrix()
signsNamesData = getSignNamesData()
signNames = []
i=0
for sign in signsNamesData:
    i=i+1
    signNames.append(str(i)+'-'+sign[1])
print(signNames)
# #
plt.figure()
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.title('Histogram of lable frequency')
```

第 1~5 行為定義讀取標籤名的.csv 檔案，第 7~10 行為輸出讀取的標籤名資料，而第 12~17 行為統計標籤種類的出現的數量，並繪製呈長條圖。

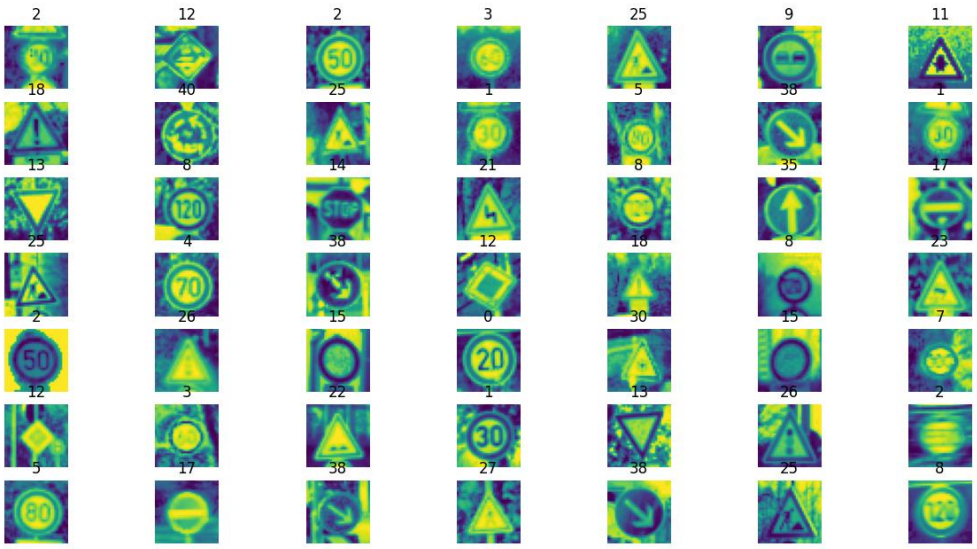


4. 影像前處理

```
def gray_equlize(img):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    equ = cv2.equalizeHist(gray)
    return equ
x_train = np.array([gray_equlize(img) for img in x_train])
```

```
x_test = np.array([gray_equlize(img) for img in x_test])
x_valid = np.array([gray_equlize(img) for img in x_valid])
#
fig, axs = plt.subplots(7, 7, figsize = (15, 12))
fig.subplots_adjust(hspace = 0.2, wspace = 0.001)
axs = axs.ravel()
for i in range(49):
    index = random.randint(0, len(x_train))
    image = x_train[index]
    axs[i].axis('off')
    axs[i].imshow(image)
    axs[i].set_title(y_train[index])
```

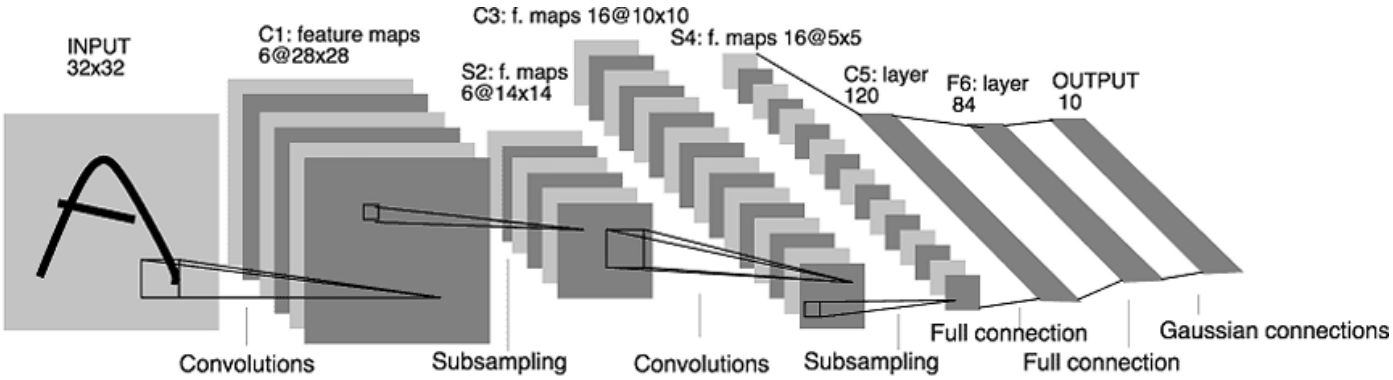
第 1~4 行為定義一個影像處理的函示，第 2 行先對影像進行灰階轉換，第 3 行再對灰階的影像進行直方圖均值化處理。第 5~7 行則使用函式對訓練集、測試集與驗證集進行影像處理，並在第 9~17 行將處理完後的訓練集影像顯示出來。



5. LeNet 卷積網路

```
model = models.Sequential()
# Conv 32x32x1 => 28x28x6.
model.add(layers.Conv2D(filters = 6, kernel_size = (5, 5), strides=(1, 1), padding='valid', activation='relu', data_format = 'channels_last', input_shape = (32, 32, 1)))
# Maxpool 28x28x6 => 14x14x6
model.add(layers.MaxPooling2D((2, 2)))
# Conv 14x14x6 => 10x10x16
model.add(layers.Conv2D(16, (5, 5), activation='relu'))
# Maxpool 10x10x16 => 5x5x16
model.add(layers.MaxPooling2D((2, 2)))
# Flatten 5x5x16 => 400
model.add(layers.Flatten())
# Fully connected 400 => 120
model.add(layers.Dense(120, activation='relu'))
# Fully connected 120 => 84
model.add(layers.Dense(84, activation='relu'))
# Dropout
model.add(layers.Dropout(0.2))
# Fully connected, output layer 84 => 43
model.add(layers.Dense(43, activation='softmax'))
model.summary()
```

這邊指定了模型的卷積層、最大池化層、展開層及全連接層的數量，並且給定激活函數與各層的大小，這裡的激活函數選擇為”ReLu”，最後顯示模型的各層的參數數量和輸出形狀；LeNet 具有兩個捲積層、兩個最大池化層、兩個全連接層及一個 Dropout 層，如下圖所示。



Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 6)	156
max_pooling2d (MaxPooling2D)	(None, 14, 14, 6)	0
conv2d_1 (Conv2D)	(None, 10, 10, 16)	2,416
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 16)	0
flatten (Flatten)	(None, 400)	0
dense (Dense)	(None, 120)	48,120
dense_1 (Dense)	(None, 84)	10,164
dropout (Dropout)	(None, 84)	0
dense_2 (Dense)	(None, 43)	3,655

Total params: 64,511 (252.00 KB)
Trainable params: 64,511 (252.00 KB)
Non-trainable params: 0 (0.00 B)

6. 影像數據增強處理

```
1.  x_train = np.expand_dims(x_train, axis=-1)
2.  data_aug = ImageDataGenerator(
3.          featurewise_center=False,
4.          featurewise_std_normalization=False,
5.          rotation_range= 10,
6.          zoom_range=0.2,
7.          width_shift_range=0.1,
8.          height_shift_range=0.1,
9.          shear_range=0.11,
10.         horizontal_flip=False,
11.         vertical_flip=False)
```

第 1 行為了要符合輸入的影像尺寸資訊，從 3 維轉換為 4 維。第 2~11 行為影像數據增強處理，其中包括旋转、缩放、平移、剪切與水平、垂直翻轉等，如此可以使模型學習到更多不同的角度和變換後的特徵。

7. 模型準確率及優化器設定

```
1.  # Define a Callback class that stops training once accuracy reaches 98.0%
```

```
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('accuracy')>0.97):
      print("\nReached 97.0% accuracy so cancelling training!")
      self.model.stop_training = True
callbacks = myCallback()
# specify optimizer, loss function and metric
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

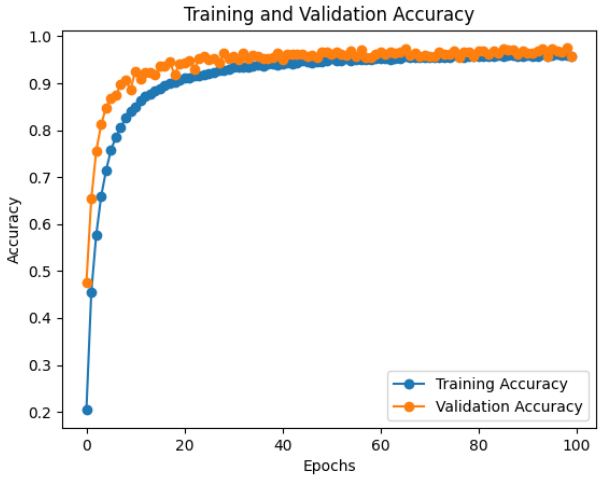
第 1~6 行的地方定義一個回調函數，當每個訊連回合結束時，便會呼叫此函數，在準確率到達指定的數值時，則停止模型的訓練。第 9 行說明了模型編譯的相關設定，設定了優化器、損失函數、評估指標，這裡的優化器設定為”Adam”。

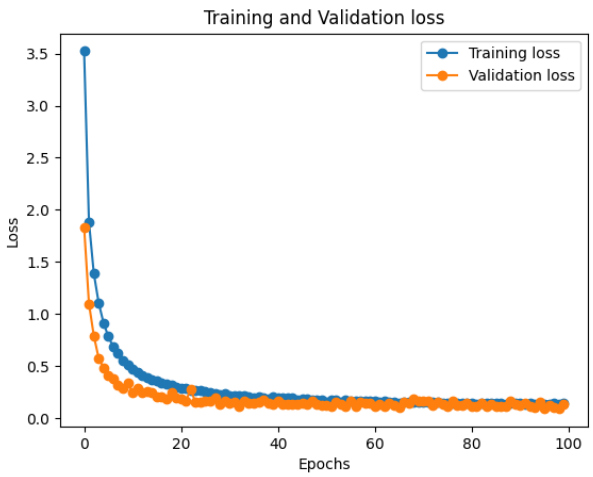
8. 模型的執行及準確度、損失率的圖表

```
conv = model.fit(data_aug.flow(x_train, y_train, batch_size=128), epochs=100,
                 validation_data=(x_valid, y_valid),verbose = 2,
                callbacks=[callbacks])
print(conv.history.keys())
# summarize history for accuracy
plt.figure()
plt.plot(conv.history['accuracy'],'-o')
plt.plot(conv.history['val_accuracy'],'-o')
plt.title('Training and Validation Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower right')
# summarize history for loss
plt.figure()
plt.plot(conv.history['loss'],'-o')
plt.plot(conv.history['val_loss'],'-o')
plt.title('Training and Validation loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['Training loss', 'Validation loss'], loc='upper right')
model.evaluate(x=x_test, y=y_test)
model.save('traffic_sign_detection_100_1.hdf5')
```

第 1~3 行輸入數據增強後的數據結果，並且設定想要訓練的 batch_size、epochs，在每個回合結束時以驗證集影像來做驗證，最後輸出詳細的訓練資料。第 4 行顯示模型的先前訓練的資料，其中包含準確度、驗證準確度、損失率及驗證損失率。在第 6~20 行設定繪製出準確度及損失率的數據。第 21 行則輸出每一個回合測試集的準確度及損失率，第 22 行為輸出本次訓練完成的模型。本次模型的訓練次數為 100 epoch。

```
Epoch 96/100
272/272 - 5s - 17ms/step - accuracy: 0.9597 - loss: 0.1376 - val_accuracy: 0.9741 - val_loss: 0.0888
Epoch 97/100
272/272 - 5s - 17ms/step - accuracy: 0.9616 - loss: 0.1307 - val_accuracy: 0.9692 - val_loss: 0.1275
Epoch 98/100
272/272 - 4s - 16ms/step - accuracy: 0.9591 - loss: 0.1405 - val_accuracy: 0.9676 - val_loss: 0.1070
Epoch 99/100
272/272 - 4s - 16ms/step - accuracy: 0.9604 - loss: 0.1365 - val_accuracy: 0.9748 - val_loss: 0.0909
Epoch 100/100
272/272 - 4s - 17ms/step - accuracy: 0.9581 - loss: 0.1440 - val_accuracy: 0.9585 - val_loss: 0.1353
dict_keys(['accuracy', 'loss', 'val_accuracy', 'val_loss'])
395/395 ━━━━━━━━━━━━━━━━━━━━ 0s 777us/step - accuracy: 0.9436 - loss: 0.2358
```

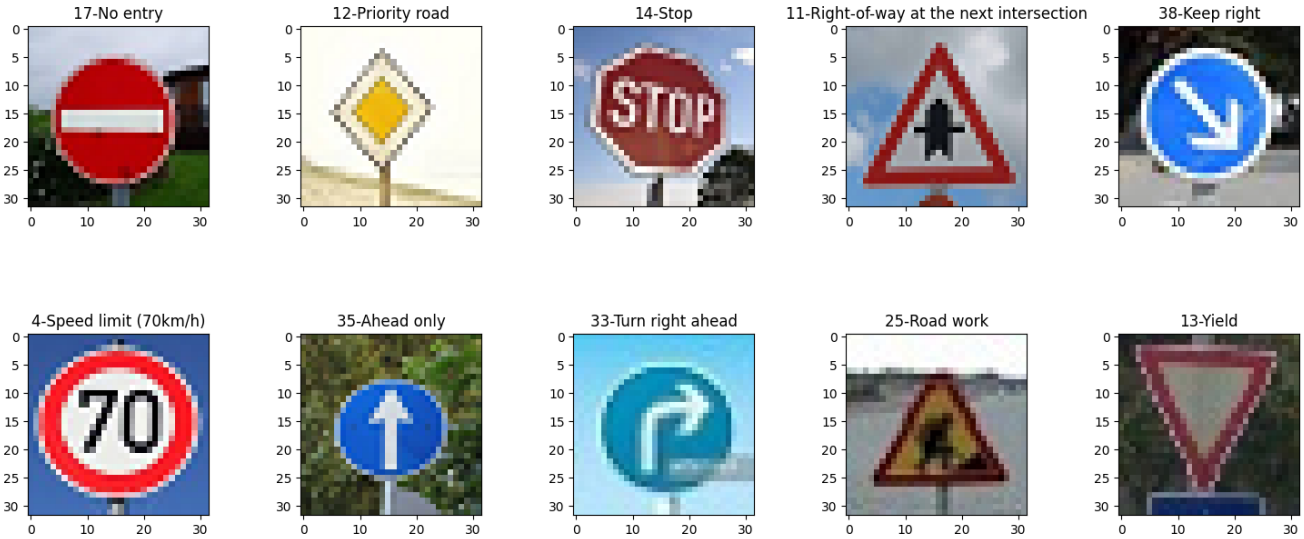




9. 引入訓練好的模型及分類新資料

```
1. import os
2. import cv2
3. import numpy as np
4. import tensorflow as tf
5. import matplotlib.pyplot as plt
6. import matplotlib.image as mpimg
7. import pandas as pd
8. 
9. def getSignNamesData():
10.     return pd.read_csv('./signnames.csv').values
11. # 加載模型
12. model = tf.keras.models.load_model('traffic_sign_detection_100_1.hdf5')
13. NImages = 10
14. X_real = np.zeros((NImages,32,32,3)).astype(np.uint8)
15. y_real = np.array([17,12,14,11,38,4,35,33,25,13])
16. signsNamesData = getSignNamesData()
17. signNames = []
18. i=-1
19. for sign in signsNamesData:
20.     i=i+1
21.     signNames.append(str(i)+'-'+sign[1])
22. plt.figure()
23. for i in range(NImages):
24.     print(i+1)
25.     image = cv2.imread('testImages/'+str(i+1)+'.png')
26.     image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # 转换为灰度图像
27.     prediction = model.predict(np.array([image_gray]))
28.     plt.subplot(2, 5, i+1)
29.     plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
30.     plt.title(signNames[np.argmax(prediction)])
31. plt.tight_layout()
32. plt.show()
```

先在第 12 行呼叫想要引入的模型，接著第 25 行呼叫想要分類的影像資料，最後將分類完的結果繪製成一張圖表。下圖為訓練完後的結果，經過確認後可以得知此次分類結果皆為正確。



實驗與結果

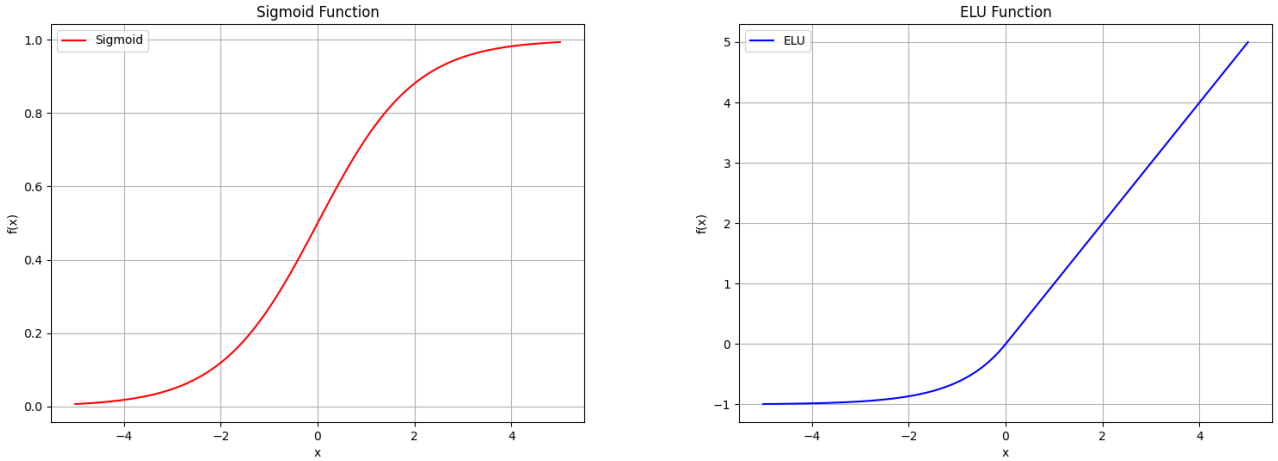
在神經網路中，每個神經元都有一個激活函數，激活函數可使上一層節點做非線性轉換，從而獲取充分的特徵組合，可以使神經網路學習到更多特徵；如果使用非線性轉換，神經網路學習就會有限。本次模型的激活函數預設是使用非線性的”ReLU”來進行運算，我們試著使用其他激活函數來進行結果的比較。

Sigmoid:

$$Sigmoid(x) = \frac{1}{1+e^{-x}}$$

ELU:

$$ELU(x) = \begin{cases} x, x > 0 \\ \alpha(e^{x}-1), x \leq 0 \end{cases}$$



Sigmoid 函數圖形參考左上圖，**ELU** 函數圖形參考右上圖。

接著進行三種激活函數的模型訓練，模型皆使用 LeNet 神經網路，訓練次數皆設為 100 epoch，其餘設定皆不調整。

Table 1 三種激活函數的準確度、損失率比較

	ReLU	Sigmoid	ELU
Loss	0.2358	0.2741	0.2505
Accuracy	0.9436	0.9168	0.9371

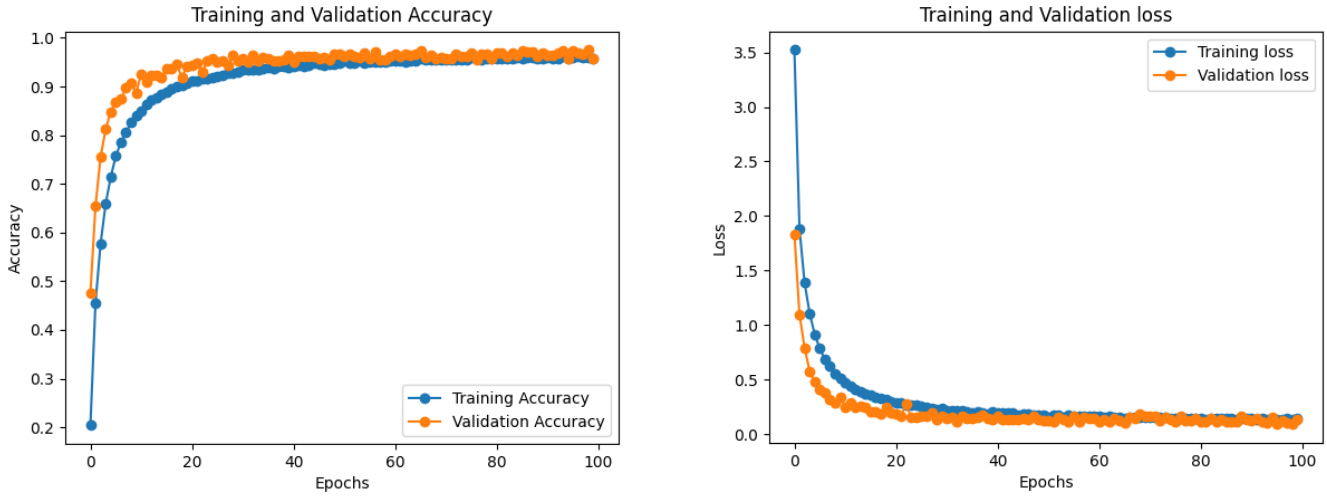


Figure 1 LeNet 使用 ReLU 激活函數的訓練準確度及損失率

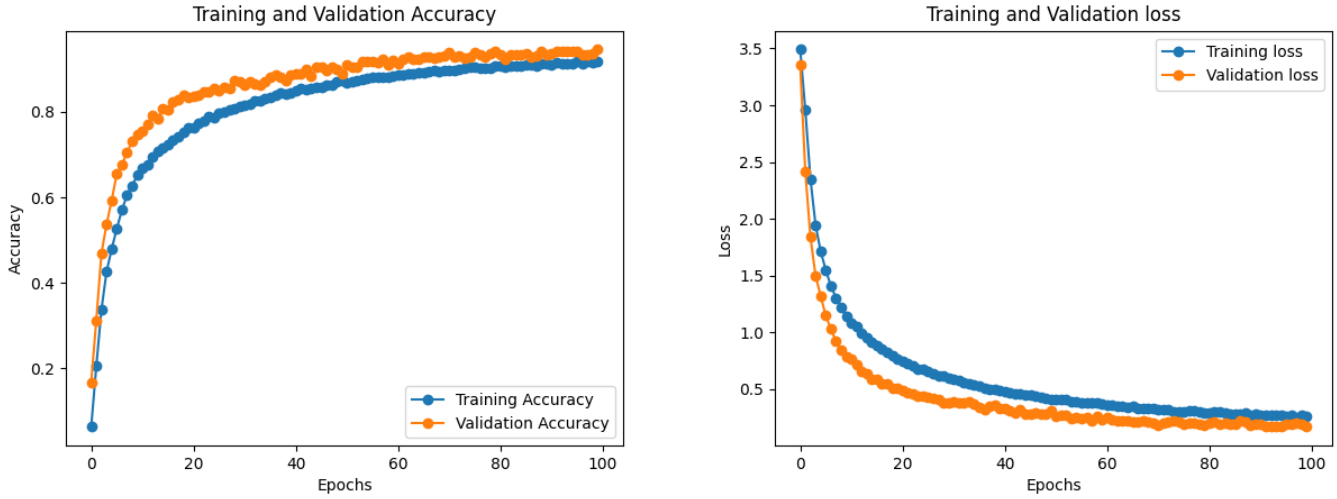


Figure 2 LeNet 使用 Sigmoid 激活函數的訓練準確度及損失率

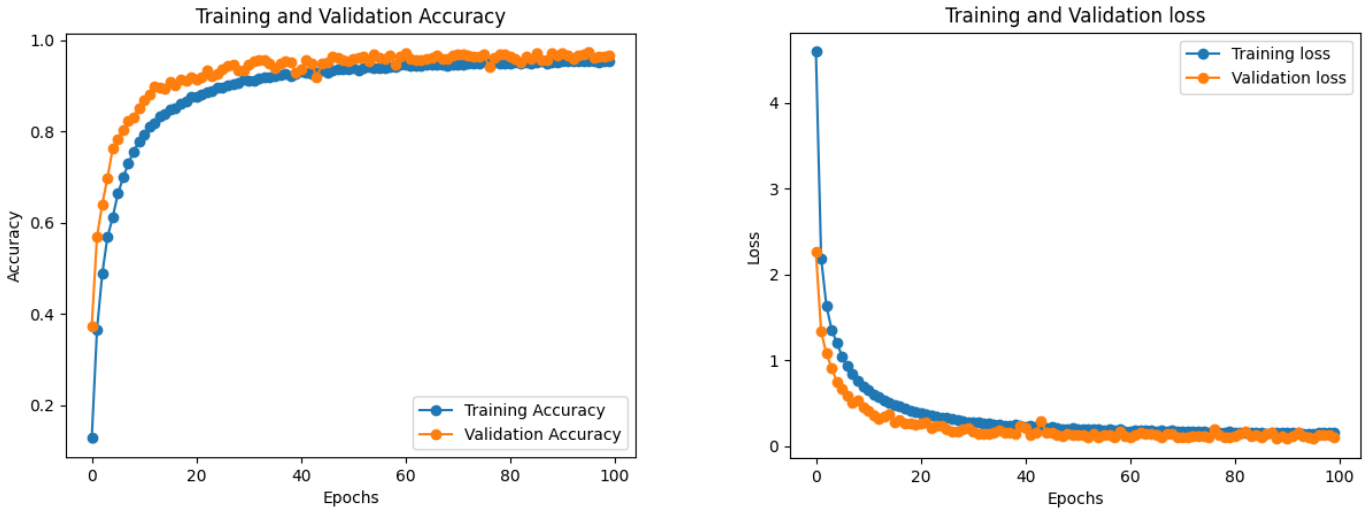


Figure 3 LeNet 使用 ELU 激活函數的訓練準確度及損失率

我們可以從 table 1 中看出在同樣的訓練次數中，ReLU 激活函數訓練的模型具有最好的準確率以及最低的損失率，而 Sigmoid 激活函數訓練的模型在兩項數據中皆為表現最差的。

Figure 4　LeNet 使用 ReLU 激活函數的訓練結果

Figure 5　LeNet 使用 Sigmoid 激活函數的訓練結果

Figure 6　LeNet 使用 ELU 激活函數的訓練結果

從上面三張圖可以看出模型分類後的結果，可以看出在 ReLU、Sigmoid 的模型上分類結果皆為正確，而在 ELU 的分類結果，在限速 70km/h 標誌的分類上誤分類成”限速 20km/h”，如上 Figure 6 紅框所示。

結論

這次的作業為使用 LeNet 進行分類模型的訓練，而使用的數據為已預先處理好分為訓練、驗證與測試集。從上面的結果可以看出經過 100 回合的訓練已經可以有不錯的結果，之後可以嘗試增加其他影像前處理的方法，或是調整影響處理的大小，使得訓練的次數回合可以更短。除了 LeNet 以外也有許多神經網路可以使用，像是 ResNet、AlexNet 等深度卷積神經網路，可以試著使用並比較分類的結果，或是可以自行設計卷積層的數量以及相關的參數。