**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**

1. from sklearn.utils import shuffle
2. from tensorflow.keras import datasets, layers, models
3. from tensorflow.keras.preprocessing.image import ImageDataGenerator
4. from keras.models import load\_model
5. import matplotlib.pyplot as plt
6. import matplotlib.image as mpimg
7. import tensorflow as tf
8. import pickle
9. import numpy as np
10. import random
11. import pandas as pd
12. import cv2
13. import glob

引入會使用到的函式庫。

**2.數據集影像的讀取及資料**

1. training\_file = 'traffic-signs-data/train.p'
2. validation\_file='traffic-signs-data/valid.p'
3. testing\_file = 'traffic-signs-data/test.p'
4. with open(training\_file, mode='rb') as f:
5. train = pickle.load(f)
6. with open(validation\_file, mode='rb') as f:
7. valid = pickle.load(f)
8. with open(testing\_file, mode='rb') as f:
9. test = pickle.load(f)
10. x\_train, y\_train = train['features'], train['labels']
11. x\_valid, y\_valid = valid['features'], valid['labels']
12. x\_test, y\_test = test['features'], test['labels']
13. print("x\_train shape:", x\_train.shape)
14. print("y\_train shape:", y\_train.shape)
15. print("x\_valid shape:", x\_valid.shape)
16. print("y\_valid shape:", y\_valid.shape)
17. print("X\_test shape:", x\_test.shape)
18. print("y\_test shape:", y\_test.shape)
19. # #
20. # Number of training example
21. n\_train = x\_train.shape[0]
22. # Number of valid example
23. n\_valid=x\_valid.shape[0]
24. # Number of testing examples.
25. n\_test = x\_test.shape[0]
26. 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。

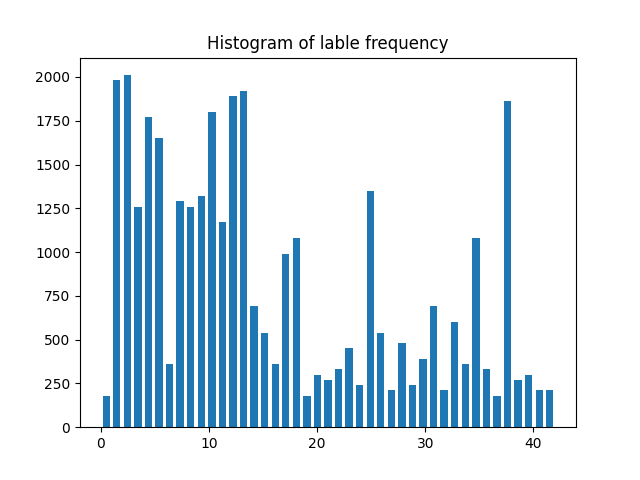
1. def getLabelsCount(labels):
2. d = dict(zip(labels, [0] \* len(labels)))
3. for x in labels:
4. d[x] += 1
5. return d
6. signsDicts = getLabelsCount(y\_train)
7. n\_classes = len(signsDicts)
8. print("Number of training examples =", n\_train)
9. print("Number of validation examples =", n\_valid)
10. print("Number of testing examples =", n\_test)
11. print("Image data shape =", image\_shape)
12. 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**

1. def getSignNamesData():
2. return pd.read\_csv('./signnames.csv').values
3. #return pd.read\_csv('./signnames.csv').as\_matrix()
4. signsNamesData = getSignNamesData()
5. signNames = []
6. i=0
7. for sign in signsNamesData:
8. i=i+1
9. signNames.append(str(i)+'-'+sign[1])
10. print(signNames)
11. # #
12. plt.figure()
13. hist, bins = np.histogram(y\_train, bins = n\_classes)
14. width = 0.7 \* (bins[1] - bins[0])
15. center = (bins[:-1] + bins[1:]) / 2
16. plt.bar(center, hist, align = 'center', width = width)
17. plt.title('Histogram of lable frequency')

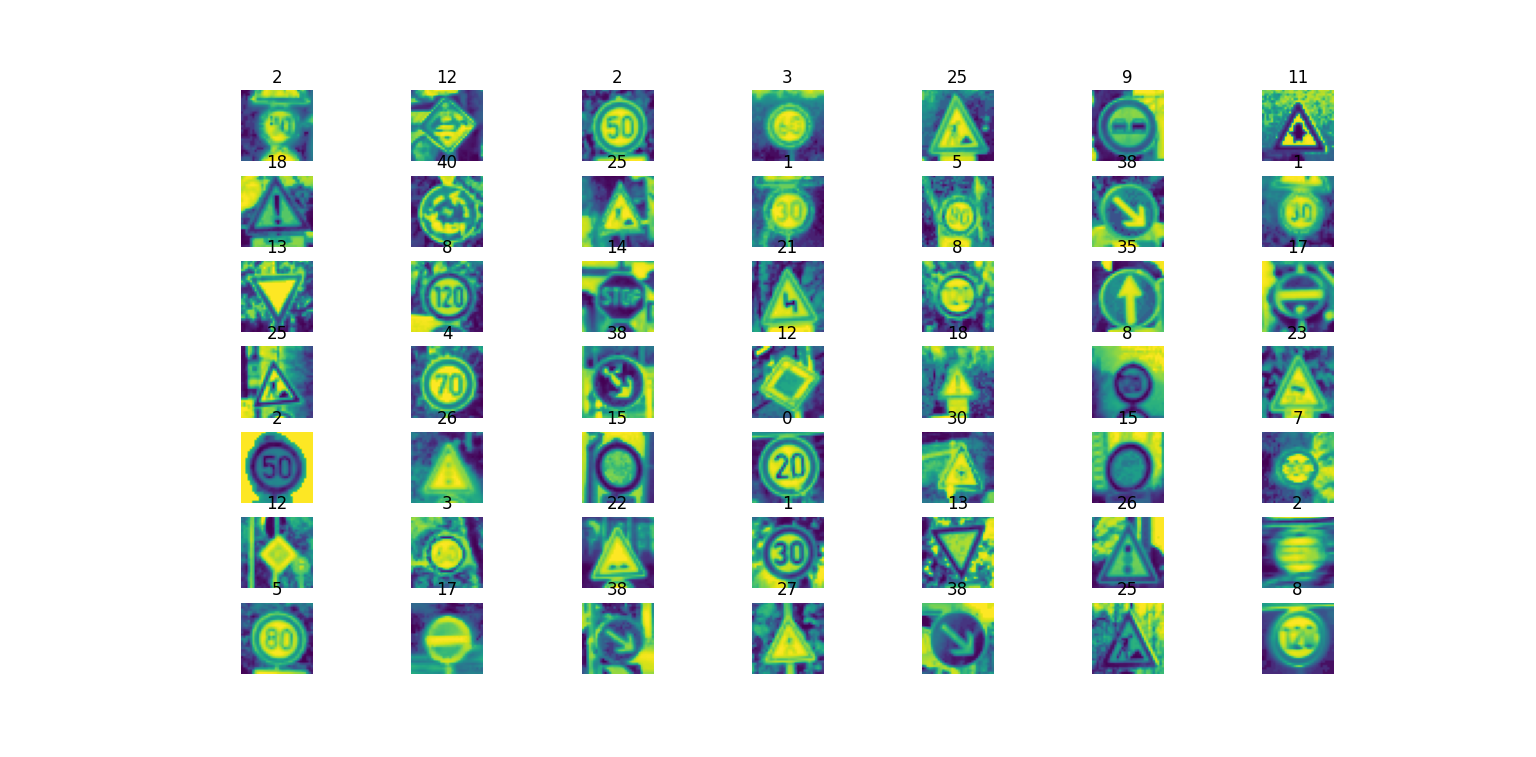
第1~5行為定義讀取標籤名的.csv檔案，第7~10行為輸出讀取的標籤名資料，而第12~17行為統計標籤種類的出現的數量，並繪製呈長條圖。



**4.影像前處理**

1. def gray\_equlize(img):
2. gray = cv2.cvtColor(img, cv2.COLOR\_RGB2GRAY)
3. equ = cv2.equalizeHist(gray)
4. return equ
5. x\_train = np.array([gray\_equlize(img) for img in x\_train])
6. x\_test = np.array([gray\_equlize(img) for img in x\_test])
7. x\_valid = np.array([gray\_equlize(img) for img in x\_valid])
8. #
9. fig, axs = plt.subplots(7, 7, figsize = (15, 12))
10. fig.subplots\_adjust(hspace = 0.2, wspace = 0.001)
11. axs = axs.ravel()
12. for i in range(49):
13. index = random.randint(0, len(x\_train))
14. image = x\_train[index]
15. axs[i].axis('off')
16. axs[i].imshow(image)
17. axs[i].set\_title(y\_train[index])

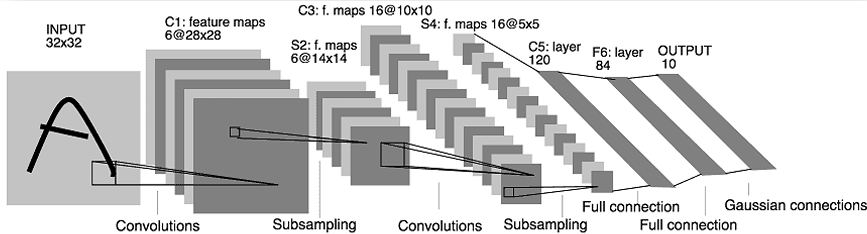
第1~4行為定義一個影像處理的函示，第2行先對影像進行灰階轉換，第3行再對灰階的影像進行直方圖均值化處理。第5~7行則使用函式對訓練集、測試集與驗證集進行影像處理，並在第9~17行將處理完後的訓練集影像顯示出來。

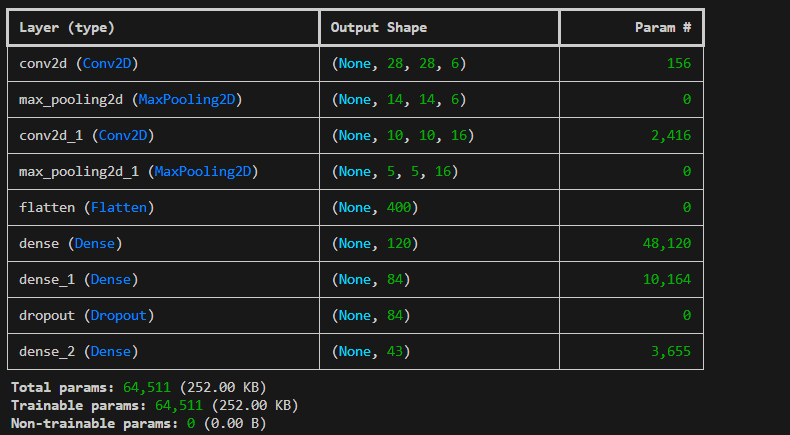


**5.LeNet卷積網路**

1. model = models.Sequential()
2. # Conv 32x32x1 => 28x28x6.
3. 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)))
4. # Maxpool 28x28x6 => 14x14x6
5. model.add(layers.MaxPooling2D((2, 2)))
6. # Conv 14x14x6 => 10x10x16
7. model.add(layers.Conv2D(16, (5, 5), activation='relu'))
8. # Maxpool 10x10x16 => 5x5x16
9. model.add(layers.MaxPooling2D((2, 2)))
10. # Flatten 5x5x16 => 400
11. model.add(layers.Flatten())
12. # Fully connected 400 => 120
13. model.add(layers.Dense(120, activation='relu'))
14. # Fully connected 120 => 84
15. model.add(layers.Dense(84, activation='relu'))
16. # Dropout
17. model.add(layers.Dropout(0.2))
18. # Fully connected, output layer 84 => 43
19. model.add(layers.Dense(43, activation='softmax'))
20. 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。





**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**

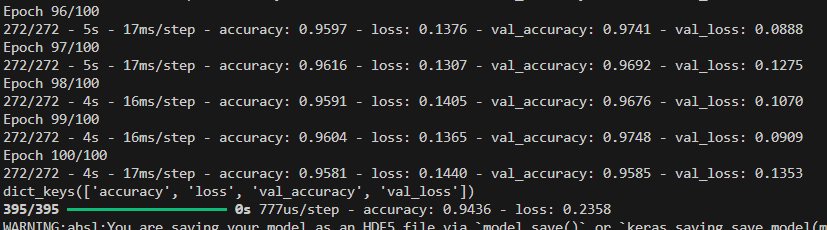
1. # Define a Callback class that stops training once accuracy reaches 98.0%
2. class myCallback(tf.keras.callbacks.Callback):
3. def on\_epoch\_end(self, epoch, logs={}):
4. if(logs.get('accuracy')>0.97):
5. print("\nReached 97.0% accuracy so cancelling training!")
6. self.model.stop\_training = True
7. callbacks = myCallback()
8. # specify optimizer, loss function and metric
9. model.compile(optimizer='adam',loss='sparse\_categorical\_crossentropy',metrics=['accuracy'])

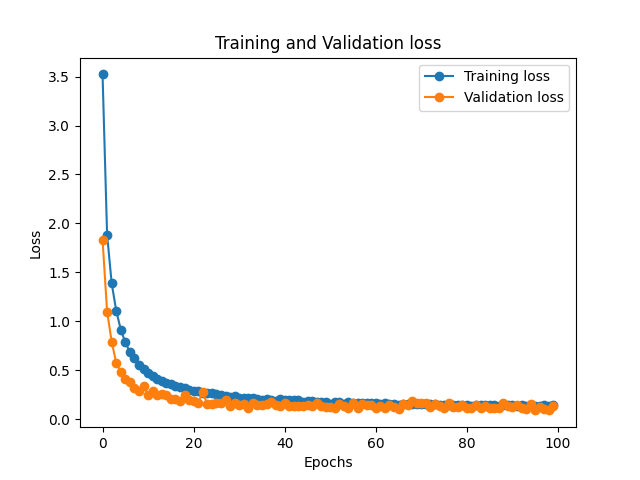
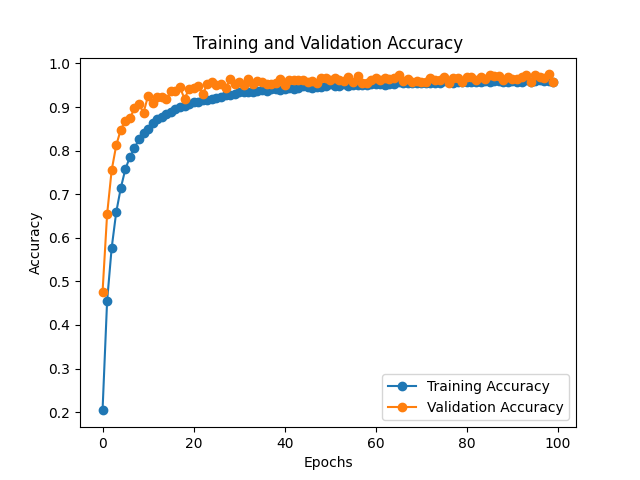
第1~6行的地方定義一個回調函數，當每個訊連回合結束時，便會呼叫此函數，在準確率到達指定的數值時，則停止模型的訓練。第9行說明了模型編譯的相關設定，設定了優化器、損失函數、評估指標，這裡的優化器設定為”Adam”。

**8.模型的執行及準確度、損失率的圖表**

1. conv = model.fit(data\_aug.flow(x\_train, y\_train, batch\_size=128), epochs=100,
2. validation\_data=(x\_valid, y\_valid),verbose = 2,
3. callbacks=[callbacks])
4. print(conv.history.keys())
5. # summarize history for accuracy
6. plt.figure()
7. plt.plot(conv.history['accuracy'],'-o')
8. plt.plot(conv.history['val\_accuracy'],'-o')
9. plt.title('Training and Validation Accuracy')
10. plt.ylabel('Accuracy')
11. plt.xlabel('Epochs')
12. plt.legend(['Training Accuracy', 'Validation Accuracy'], loc='lower right')
13. # summarize history for loss
14. plt.figure()
15. plt.plot(conv.history['loss'],'-o')
16. plt.plot(conv.history['val\_loss'],'-o')
17. plt.title('Training and Validation loss')
18. plt.ylabel('Loss')
19. plt.xlabel('Epochs')
20. plt.legend(['Training loss', 'Validation loss'], loc='upper right')
21. model.evaluate(x=x\_test, y=y\_test)
22. 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。





**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. def getSignNamesData():
9. return pd.read\_csv('./signnames.csv').values
10. # 加载模型
11. model = tf.keras.models.load\_model('traffic\_sign\_detection\_100\_1.hdf5')
12. NImages = 10
13. X\_real = np.zeros((NImages,32,32,3)).astype(np.uint8)
14. y\_real = np.array([17,12,14,11,38,4,35,33,25,13])
15. signsNamesData = getSignNamesData()
16. signNames = []
17. i=-1
18. for sign in signsNamesData:
19. i=i+1
20. signNames.append(str(i)+'-'+sign[1])
21. plt.figure()
22. for i in range(NImages):
23. print(i+1)
24. image = cv2.imread('testImages/'+str(i+1)+'.png')
25. image\_gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY) # 转换为灰度图像
26. prediction = model.predict(np.array([image\_gray]))
27. plt.subplot(2, 5, i+1)
28. plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))
29. plt.title(signNames[np.argmax(prediction)])
30. plt.tight\_layout()
31. 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。

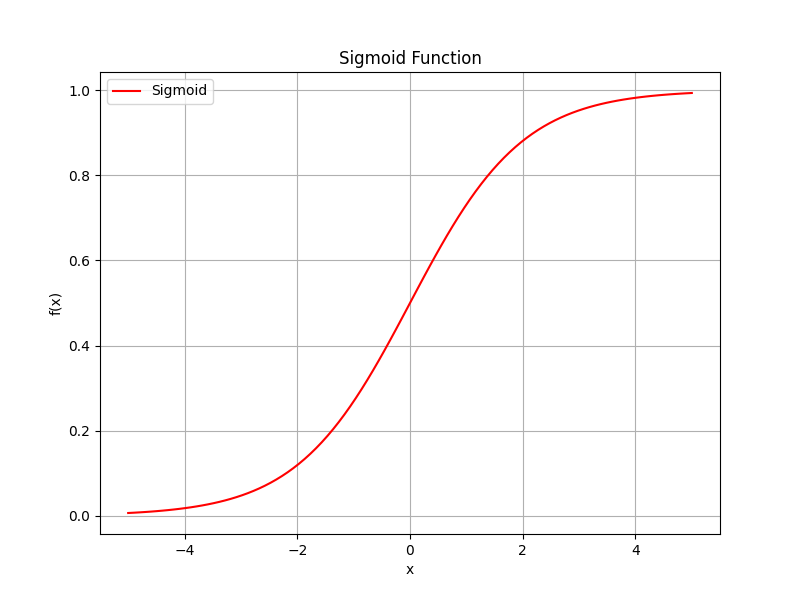
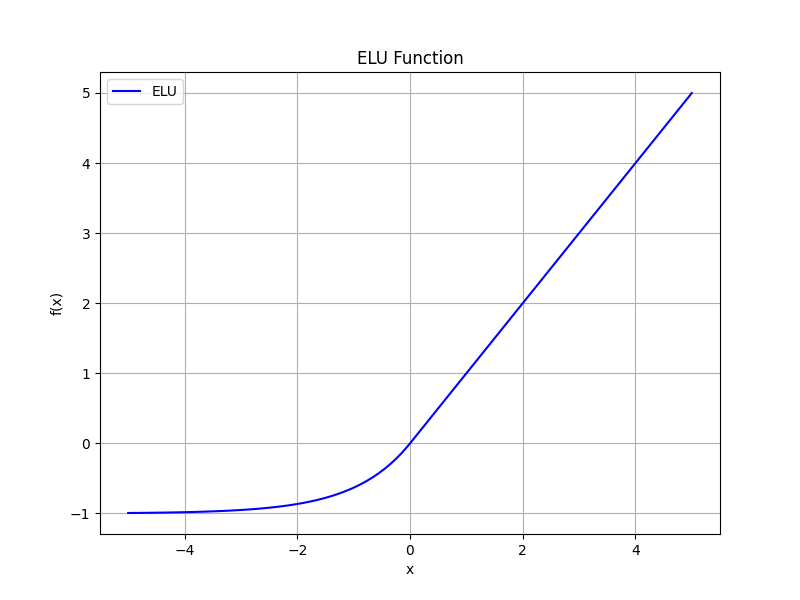


**實驗與結果**

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:***

**ELU:**

*******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

Figure 3 LeNet使用ELU激活函數的訓練準確度及損失率

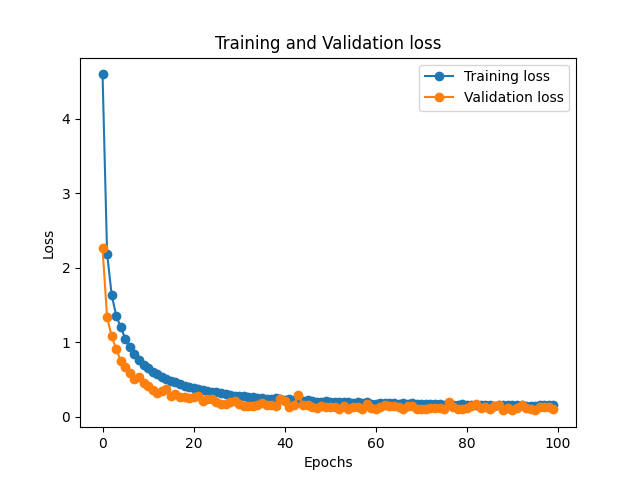
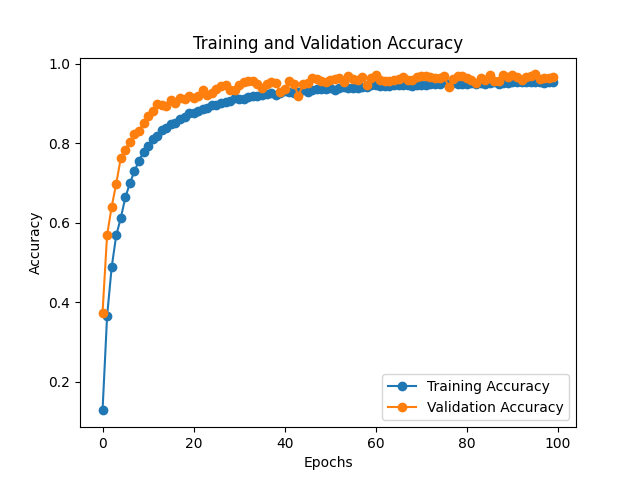


Figure 2 LeNet使用Sigmoid激活函數的訓練準確度及損失率

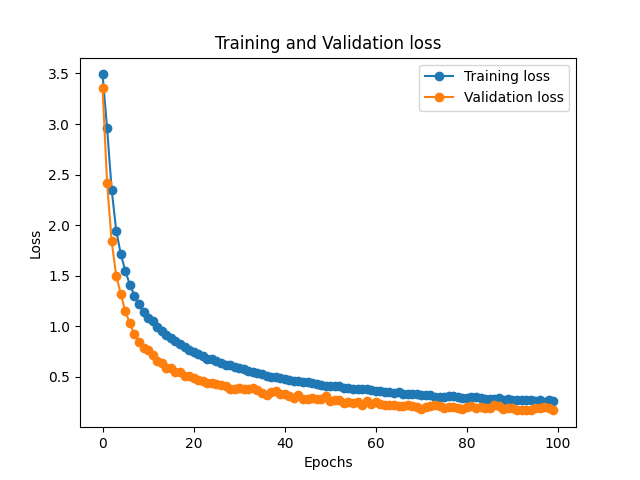
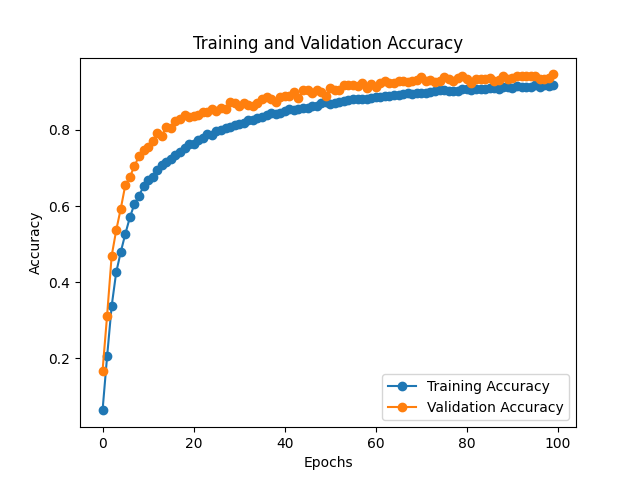
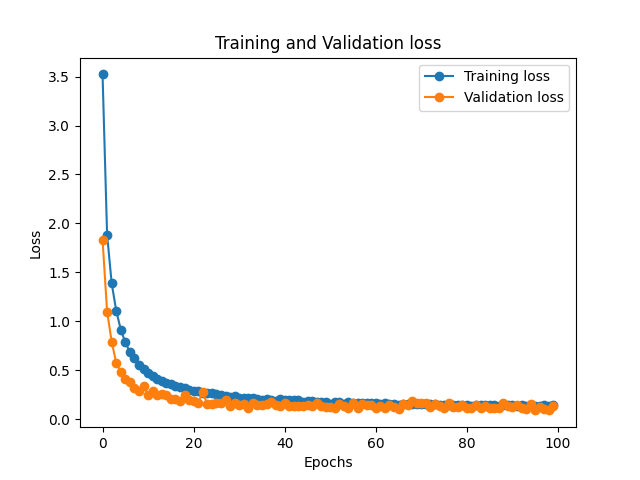
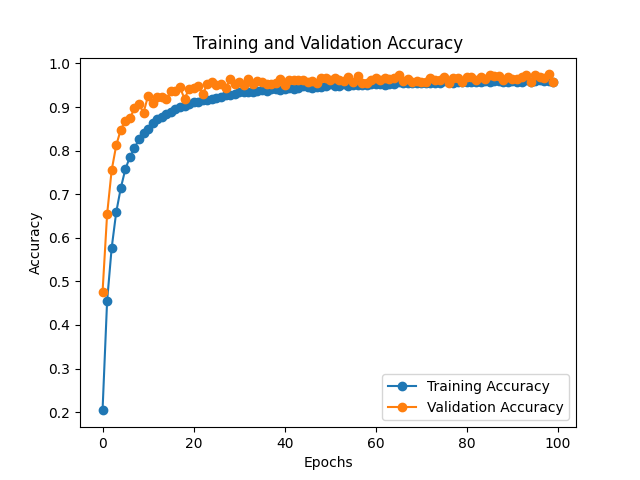


Figure 1 LeNet使用ReLU激活函數的訓練準確度及損失率



|  |  |  |  |
| --- | --- | --- | --- |
|  | ReLU | Sigmoid | ELU |
| Loss | 0.2358 | 0.2741 | 0.2505 |
| Accuracy | 0.9436 | 0.9168 | 0.9371 |

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 6 LeNet使用ELU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 4 LeNet使用ReLU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。