CNN 之環境景觀影像分類識別

109034548 吳仲人

智慧化企業整合 期末個人報告

摘要

自動駕駛的關鍵技術之一的人工智慧技術，近幾年來越來越成熟，本研究至 Kaggle 公開數據集中蒐集一些基礎之環境景觀影像，並利用 CNN 機器學習影像辨識系統來自動辨識駕駛人行車過程可能會經過之環境景觀影像，經測試後準確率最高可達 84.57%。

關鍵字：自動駕駛、CNN、機器學習

目標

壹、 背景

1.1 研究背景

隨著 5G、大數據時代來臨，自駕車已成為近年來全球的顯學，各國政府、各大企業皆以發展自駕車為重要目標。目前全球各地正積極進行自駕車的測試，跨國企業亦陸續宣布推出共享自駕車服務的時間表。隨著人類科技的腳步加快，創新科技被廣泛採用的時程越來越短，一般預期自駕車將會以驚人的速度在全球道路上大量出現，人們也很快的將之視為平常。

1.2 研究動機

美國汽車書籍作者兼評論家 Dan Albert 日前於新書中指出，近來包括配備自駕功能的特斯拉（Tesla）電動車等事故，皆造成死亡案件，指出「自駕車技術尚未成熟」，並質疑自駕車業者宣稱自駕車有助減少交通事故的說法。

而在台灣，也發生過類似的事故：

（2020-06-01 聯合報 / 記者陳苡葳／雲林縣即時報導）

國道 1 號南向 268.3 公里處，昨天上午 6 時 35 分一輛載滿食品的營業小貨車倒在內側車道上，車上 34 歲葉姓男駕駛自行脫困，站在中央分隔島等待救援時，6 時 44 分突然後方來了一輛特斯拉疑似煞車不及，直接撞進貨車廂，第四公路警察大隊表示，駕駛特斯拉為 53 歲黃姓男子，他供稱當時有開啓車輛輔助系統，原以為車輛會偵測到障礙物而降速或停駛，但車子仍定速前進，因此當在最後一刻要煞車時已來不及才會釀禍。

由以上新聞得知，該名駕駛認為車輛會偵測到前方的障礙物而自動閃避。然而，車輛或許可以辨識正常行駛的貨車，但卻無法辨識處於翻車狀態的貨車。因此，要做到自動駕駛的首要問題就是克服行車路況影像辨識。

1.3 研究目的

因此，本研究乃自網際網路蒐集基本的環境景觀影像，期望能使用卷積神經網絡(Convolutional Neural Network)模型正確辨別此些基本環境景觀影像，以作為後續行車路況影像辨識之基礎。

1.4 5W1H 分析法

項目	內容
What	自駕車無法辨別某些特定場景
Where	可供車輛行駛之道路（如高速公路、快速道路、一般道路等）
Who	具自駕功能之車輛的駕駛人
When	當駕駛人行駛於道路上時
Why	使配備自駕功能的車輛在行車時更為安全
How	各類基本環境景觀影像的資料前處理及 CNN 模型訓練

貳、 文獻回顧

2.1 先進駕駛輔助系統 (Advanced Driver Assistance System, ADAS)

先進駕駛輔助系統 (Advanced Driver Assistance System, ADAS) 是應用於車輛駕駛的先進技術，由於 駕駛人本身可能偶有不專注或是其他判定錯誤之因素，造成交通事故，因此若能在事前提醒駕駛者道路使 用情況、提供警示，則有助於減少事故發生，這些技術一直以來是業界及學界所重視的議題。ADAS 其中的 一個功能即是偵測車輛前方的障礙物，如行人、機車、腳踏車、車輛。為達成此目的，電腦視覺 (Computer Vision)偵測以及辨識技術已被廣泛地使用於 ADAS，用以提供障礙物提示並提醒駕駛人道路環境情形，改善行車安全。

2.2 機器學習於智慧車輛應用

電腦視覺，是讓電腦模擬人眼如何去「看」的一門科學，利用攝影機與電腦代替人眼對目標物偵測和識別，箇中細節會因欲偵測物體或環境不同，而有不同的實現方式。如 ADAS 需要在複雜的行車道路場景中，偵測多種不同類型障礙物，則是採用近年來熱門且深具發展潛力的深度學習 (Deep Learning)。深度學習是機器學習 (Machine Learning) 中近年來備受重視的一支，深度學習根源於人工類神經網路 (Artificial Neural Network) 模型，目前最好的語音辨識和影像辨識系統都是以深度學習技術來完成，在許多場合能有各式驚人應用，好比日前很紅的 AlphaGo 便是一經典例子。

類神經網路主要在於模擬生物神經網路的結構和功能，像是動物的中樞神經系統，特別是大腦。利用數學模型模擬生物的神經連結，仿造人腦完成複雜的圖型辨識動作，以分析學習的機制來理解大量的資料。此方法可應用於影像、聲音和文字等多種領域，而影像應用則是仰賴卷積神經網路 (Convolution Neural Network，CNN) 技術，CNN 是類神經網路的一種，最常應用於影像偵測，由於是模擬人腦的運作，其主要擁有自主學習不同類別特徵的能力。以 ADAS 主要欲偵測的障礙物為例，我們人眼之所以看到人或車輛可以立即分辨出不同，是這兩個類別具備明顯不同的特徵。像是人有眼睛、鼻子、手和腳等特徵，我們可以立即知道是人；而汽車本身有輪胎或是車窗等特徵，進而能使人眼立即判斷。

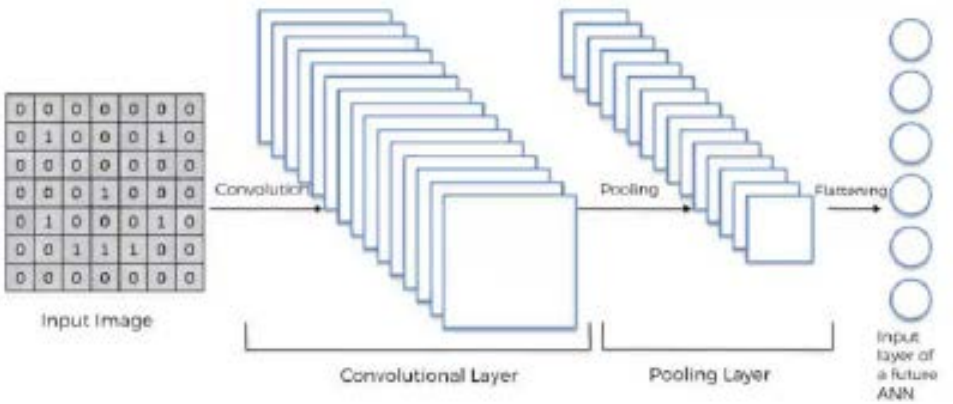
2.3 CNN 如何應用於行車障礙物偵測

CNN 如何應用於行車障礙物偵測？就如同人類在學習分類的過程一樣，需要準備很多我們想要偵測的障礙物照片，例如不同角度的機車騎士或汽車照片，使 CNN 學習這些照片中的規律性。而 CNN 達到自主學習各種類別特徵以及正確分類的關鍵在於其具有多層次的結構，就如同我們欲分辨前方的是行人、腳踏車騎士、汽車還是機車騎士之前，會經過人眼和大腦好幾關的判定一樣。像是第一關是有沒有人的特徵，如 果有我們就會將汽車這個類別先刪去；第二關有無輪胎之特徵，如果有行人這個可能性也會被剔除；最後一關則可能是有無機車或腳踏車特徵才判定結果。

我們給予 CNN 大量的照片，希望讓其模擬上述分類邏輯，CNN 的多層次就如同我們大腦判定時的多個條件一樣，每一階層是學習不同的條件。假如今天有個 CNN 包含五層的結構，給予許多照片去學習的結果，可以像是蓋房子一樣：第一層可能是在找有無直線或是弧線等比較底層的特徵；到了第二層則可以利用前一層的特徵結合出較大的特徵，如完整的正方型或是圓形等形狀；最後一層，則可以完整描述出整個物體的特徵，如整個人臉或是車子造型。

2.4 卷積神經網路(Convolutional Neural Network, CNN)

CNN（Convolutional Neural Network，卷積神經網絡）是模仿人類大腦認知方式的一種深度學習方法，例如我們辨識一個圖像，會先注意到顏色鮮明的點、線、面，之後再將它們構成一個個不同的形狀(眼睛、鼻子、嘴巴...)，這種抽象化的過程就是 CNN 演算法建立模型的方式。 CNN 透過卷積層(Convolution Layer)、池化層(Pooling Layer)、全連接層(Fully Connected Layer) 這三個主要步驟來達到辨識圖片的功能。卷積層就是由點的比對轉成局部的比對，透過一塊塊的特徵研判，逐步堆疊綜合比對結果，Max Pooling 的概念是挑出矩陣當中的最大值，使得若圖片平移幾個 Pixel 對判斷上完全不會造成影響，以及有很好的抗雜訊功能，最後則是透過全連接層將 Matrix 拉直。



參、 資料整理

3.1 資料集照片來源

由 Kaggle 公開數據集中蒐集環境景觀影像（建築物、森林、冰川、山、海、街道）共 24335 張。其檔案如下所示

- seg_pred：未分類之環境景觀影像共 7301 筆
- seg_train：已分類之環境景觀影像共 14034 筆
- seg_test：已分類之環境景觀影像共 3000 筆

3.2 資料前處理過程

一、定義 get_images 函式

(1) 將放置在各個不同的資料夾的影像進行讀取。

(2) 將讀取之影像依其所屬之資料夾給予編碼（從建築物資料夾載入的影像作為第 0 類，森林作為第 1 類，冰川作為第 2 類，山作為第 3 類，海作為第 4 類，街道作為第 5 類）。

(3) 將所有圖片調整為相同大小（150×150）。

(4) 將所有已讀取之影像進行打亂（shuffle）的動作，避免有 overfitting 的情形發生，也能避免同一個組合的 batch 反覆出現，使的模型記住這些順序。

```
def get_images(directory):
    Images = []
    Labels = []   # 0 for Building , 1 for forest, 2 for glacier, 3 for mountain, 4 for Sea , 5 for Street
    label = 0

    for labels in os.listdir(directory): #Main Directory where each class label is present as folder name.
        if labels == 'glacier': #Folder contain Glacier Images get the '2' class label.
            label = 2
        elif labels == 'sea':
            label = 4
        elif labels == 'buildings':
            label = 0
        elif labels == 'forest':
            label = 1
        elif labels == 'street':
            label = 5
        elif labels == 'mountain':
            label = 3

        for image_file in os.listdir(directory+labels): #Extracting the file name of the image from Class Label folder
            image = cv2.imread(directory+labels+r'/'+image_file) #Reading the image (OpenCV)
            image = cv2.resize(image,(150,150)) #Resize the image, Some images are different sizes. (Resizing is very Important)
            Images.append(image)
            Labels.append(label)

    return shuffle(Images,Labels,random_state=817328462) #Shuffle the dataset you just prepared.

def get_classlabel(class_code):
    labels = {2:'glacier', 4:'sea', 0:'buildings', 1:'forest', 5:'street', 3:'mountain'}

    return labels[class_code]
```

二、執行 get_images 函式（seg_train），以作為訓練資料

Shape of Images: (14034, 150, 150, 3)表示共有 14034 張圖像，尺寸為 150 150，RGB。

```
Images, Labels = get_images('../content/seg_train/seg_train/') #Extract the training images from the folders.

Images = np.array(Images) #converting the list of images to numpy array.
Labels = np.array(Labels)
```

3.3 以視覺化方式呈現影像資料

```
f,ax = plot.subplots(5,5)
f.subplots_adjust(0,0,3,3)
for i in range(0,5,1):
    for j in range(0,5,1):
        rnd_number = randint(0,len(Images))
        ax[i,j].imshow(Images[rnd_number])
        ax[i,j].set_title(get_classlabel(Labels[rnd_number]))
        ax[i,j].axis('off')
```

肆、 模型設定、訓練及測試

4.1 模型架構 model architecture

包含了六個卷積層、兩個最大池化層、一個扁平層及四個全連接層，其參數設置如下所示：

CNN parameter	Value
Number of convolution layers	6
Filter size	3×3
Number of pooling layers	2
Pooling filter size	5×5
Number of fully connected layers	4
Activation function	relu
Regularization method	dropout
Classification function of the output layer	sigmoid
Loss function	sparse_categorical_crossentropy
Optimizer	adam
Learning rate	0.001
Epoch	10

```
[33] model = Models.Sequential()

     model.add(Layers.Conv2D(200,kernel_size=(3,3),activation='relu',input_shape=(150,150,3)))
     model.add(Layers.Conv2D(180,kernel_size=(3,3),activation='relu'))
     model.add(Layers.MaxPool2D(5,5))
     model.add(Layers.Conv2D(180,kernel_size=(3,3),activation='relu'))
     model.add(Layers.Conv2D(140,kernel_size=(3,3),activation='relu'))
     model.add(Layers.Conv2D(100,kernel_size=(3,3),activation='relu'))
     model.add(Layers.Conv2D(50,kernel_size=(3,3),activation='relu'))
     model.add(Layers.MaxPool2D(5,5))
     model.add(Layers.Flatten())
     model.add(Layers.Dense(180,activation='relu'))
     model.add(Layers.Dense(100,activation='relu'))
     model.add(Layers.Dense(50,activation='relu'))
     model.add(Layers.Dropout(rate=0.5))
     model.add(Layers.Dense(6,activation='sigmoid'))

     model.compile(optimizer=Optimizer.Adam(lr=0.001),loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 200)     5600
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 146, 146, 180)     324180
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 29, 29, 180)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 27, 27, 180)       291780
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 25, 25, 140)       226940
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 23, 23, 100)       126100
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 21, 21, 50)        45050
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 50)          0
_________________________________________________________________
flatten (Flatten)            (None, 800)               0
_________________________________________________________________
dense (Dense)                (None, 180)               144180
_________________________________________________________________
dense_1 (Dense)              (None, 100)               18100
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout (Dropout)            (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 306
=================================================================
Total params: 1,187,286
Trainable params: 1,187,286
Non-trainable params: 0
```

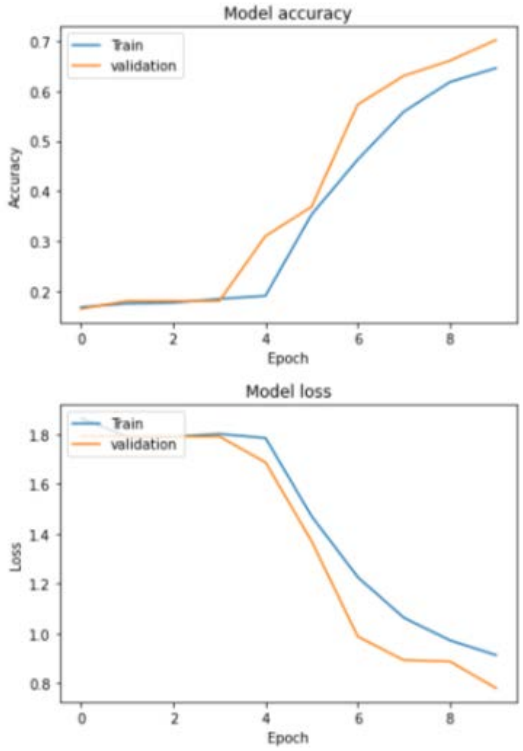
4.2 訓練模型

Epochs 設為 10，Batch size 設為 32，並將訓練資料的 30%分割為驗證資料，用來驗證此模型是否過度擬合(over-fitted)。

訓練集準確度為 64.58%，驗證集準確度 70.2%，發現不管是在訓練集或是驗證集資料都無法達到一定的準度。

```
trained = model.fit(Images, Labels, epochs=10, batch_size=32, validation_split=0.30)
```

```
Epoch 1/10
307/307 [==============================] - 91s 267ms/step - loss: 2.1572 - accuracy: 0.1674 - val_loss: 1.7923 - val_accuracy: 0.1643
Epoch 2/10
307/307 [==============================] - 84s 274ms/step - loss: 1.7904 - accuracy: 0.1766 - val_loss: 1.7920 - val_accuracy: 0.1800
Epoch 3/10
307/307 [==============================] - 84s 274ms/step - loss: 1.7899 - accuracy: 0.1835 - val_loss: 1.7917 - val_accuracy: 0.1800
Epoch 4/10
307/307 [==============================] - 83s 269ms/step - loss: 1.8133 - accuracy: 0.1940 - val_loss: 1.7916 - val_accuracy: 0.1802
Epoch 5/10
307/307 [==============================] - 82s 268ms/step - loss: 1.7923 - accuracy: 0.1772 - val_loss: 1.6873 - val_accuracy: 0.3099
Epoch 6/10
307/307 [==============================] - 83s 272ms/step - loss: 1.5666 - accuracy: 0.3266 - val_loss: 1.3700 - val_accuracy: 0.3683
Epoch 7/10
307/307 [==============================] - 83s 270ms/step - loss: 1.2865 - accuracy: 0.4281 - val_loss: 0.9875 - val_accuracy: 0.5728
Epoch 8/10
307/307 [==============================] - 83s 270ms/step - loss: 1.0633 - accuracy: 0.5585 - val_loss: 0.8925 - val_accuracy: 0.6303
Epoch 9/10
307/307 [==============================] - 83s 269ms/step - loss: 0.9714 - accuracy: 0.6160 - val_loss: 0.8874 - val_accuracy: 0.6604
Epoch 10/10
307/307 [==============================] - 83s 269ms/step - loss: 0.9215 - accuracy: 0.6458 - val_loss: 0.7799 - val_accuracy: 0.7020
```



4.3 測試模型

執行 get_images 函式（seg_test），以作為測試資料，結果測試集準確度為：71%。

```
[11] test_images,test_labels = get_images('../content/seg_test/seg_test/')
     test_images = np.array(test_images)
     test_labels = np.array(test_labels)
     model.evaluate(test_images,test_labels, verbose=1)

     94/94 [==============================] - 8s 81ms/step - loss: 0.7725 - accuracy: 0.7103
     [0.7725214958190918, 0.7103333473205566]
```

4.4 小結

原始模型之參數設定及執行結果表格如下：

CNN parameter	Value
Number of convolution layers	6
Filter size	3×3
Number of pooling layers	2
Pooling filter size	5×5
Number of fully connected layers	4
Activation function	relu
Regularization method	dropout
Classification function of the output layer	sigmoid
Loss function	sparse_categorical_crossentropy
Optimizer	adam
Learning rate	0.001
Epoch	10
Batch size	32
訓練集準確度	64.58%
驗證集準確度	70.2%
測試集準確度	71%

伍、 改善方法與過程

(1) 改善方法 1：調整學習率與輸出層分類函數（將學習率設為 0.0001，輸出層之分類函數設為 softmax 表現最佳）

```
model = Models.Sequential()

model.add(Layers.Conv2D(200,kernel_size=(3,3),activation='relu',input_shape=(150,150,3)))
model.add(Layers.Conv2D(180,kernel_size=(3,3),activation='relu'))
model.add(Layers.MaxPool2D(5,5))
model.add(Layers.Conv2D(180,kernel_size=(3,3),activation='relu'))
model.add(Layers.Conv2D(140,kernel_size=(3,3),activation='relu'))
model.add(Layers.Conv2D(100,kernel_size=(3,3),activation='relu'))
model.add(Layers.Conv2D(50,kernel_size=(3,3),activation='relu'))
model.add(Layers.MaxPool2D(5,5))
model.add(Layers.Flatten())
model.add(Layers.Dense(180,activation='relu'))
model.add(Layers.Dense(100,activation='relu'))
model.add(Layers.Dense(50,activation='relu'))
model.add(Layers.Dropout(rate=0.5))
model.add(Layers.Dense(6,activation='softmax'))

model.compile(optimizer=Optimizer.Adam(lr=0.0001),loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

改善後之結果：

```
trained = model.fit(Images,Labels,epochs=10,batch_size=32,validation_split=0.30)
```

```
Epoch 1/10
307/307 [==============================] - 91s 296ms/step - loss: 1.9215 - accuracy: 0.2862 - val_loss: 1.0482 - val_accuracy: 0.5868
Epoch 2/10
307/307 [==============================] - 90s 293ms/step - loss: 1.2010 - accuracy: 0.5194 - val_loss: 0.8936 - val_accuracy: 0.6564
Epoch 3/10
307/307 [==============================] - 90s 294ms/step - loss: 1.0321 - accuracy: 0.6025 - val_loss: 0.8417 - val_accuracy: 0.6787
Epoch 4/10
307/307 [==============================] - 90s 293ms/step - loss: 0.9330 - accuracy: 0.6471 - val_loss: 0.7556 - val_accuracy: 0.7324
Epoch 5/10
307/307 [==============================] - 90s 293ms/step - loss: 0.8320 - accuracy: 0.6936 - val_loss: 0.6675 - val_accuracy: 0.7647
Epoch 6/10
307/307 [==============================] - 90s 293ms/step - loss: 0.7403 - accuracy: 0.7344 - val_loss: 0.7271 - val_accuracy: 0.7374
Epoch 7/10
307/307 [==============================] - 90s 294ms/step - loss: 0.7197 - accuracy: 0.7476 - val_loss: 0.5911 - val_accuracy: 0.7858
Epoch 8/10
307/307 [==============================] - 90s 294ms/step - loss: 0.6438 - accuracy: 0.7788 - val_loss: 0.6011 - val_accuracy: 0.7867
Epoch 9/10
307/307 [==============================] - 90s 294ms/step - loss: 0.5905 - accuracy: 0.8009 - val_loss: 0.5970 - val_accuracy: 0.7896
Epoch 10/10
307/307 [==============================] - 90s 293ms/step - loss: 0.5367 - accuracy: 0.8173 - val_loss: 0.5248 - val_accuracy: 0.8233
```

訓練資料集之準確度為 81.73%，驗證資料集之準確度為 82.33%

```
test_images,test_labels = get_images('../content/seg_test/seg_test/')
test_images = np.array(test_images)
test_labels = np.array(test_labels)
model.evaluate(test_images,test_labels,batch_size=32,verbose=1)
```

```
94/94 [==============================] - 8s 87ms/step - loss: 0.5127 - accuracy: 0.8363
[0.5127487182617188, 0.8363333344459534]
```

測試資料集之準確度為 83.63%，發現準確度有顯著提升，因此利用此模型為基礎進行下一階段之改善。

(2) 改善方法 2：Batch size 調整為 64

```
trained = model.fit(Images, Labels, epochs=10, batch_size=64, validation_split=0.30)
```

改善後之結果：

```
Epoch 1/10
154/154 [==============================] - 118s 755ms/step - loss: 0.4752 - accuracy: 0.8431 - val_loss: 0.4907 - val_accuracy: 0.8243
Epoch 2/10
154/154 [==============================] - 114s 743ms/step - loss: 0.4345 - accuracy: 0.8556 - val_loss: 0.4930 - val_accuracy: 0.8340
Epoch 3/10
154/154 [==============================] - 115s 747ms/step - loss: 0.4139 - accuracy: 0.8602 - val_loss: 0.5065 - val_accuracy: 0.8262
Epoch 4/10
154/154 [==============================] - 115s 748ms/step - loss: 0.3787 - accuracy: 0.8755 - val_loss: 0.5380 - val_accuracy: 0.8148
Epoch 5/10
154/154 [==============================] - 115s 745ms/step - loss: 0.3678 - accuracy: 0.8788 - val_loss: 0.4791 - val_accuracy: 0.8407
Epoch 6/10
154/154 [==============================] - 115s 746ms/step - loss: 0.3536 - accuracy: 0.8813 - val_loss: 0.5020 - val_accuracy: 0.8342
Epoch 7/10
154/154 [==============================] - 115s 749ms/step - loss: 0.3217 - accuracy: 0.8918 - val_loss: 0.5568 - val_accuracy: 0.8271
Epoch 8/10
154/154 [==============================] - 116s 751ms/step - loss: 0.3013 - accuracy: 0.9015 - val_loss: 0.5183 - val_accuracy: 0.8321
Epoch 9/10
154/154 [==============================] - 115s 749ms/step - loss: 0.2725 - accuracy: 0.9083 - val_loss: 0.5281 - val_accuracy: 0.8428
Epoch 10/10
154/154 [==============================] - 115s 748ms/step - loss: 0.2545 - accuracy: 0.9143 - val_loss: 0.5524 - val_accuracy: 0.8416
```

訓練資料集之準確度為 91.43%，驗證資料集之準確度為 84.16%

```
test_images, test_labels = get_images('../content/seg_test/seg_test/')
test_images = np.array(test_images)
test_labels = np.array(test_labels)
model.evaluate(test_images, test_labels, batch_size=32, verbose=1)
```

```
94/94 [==============================] - 8s 80ms/step - loss: 0.5375 - accuracy: 0.8453
[0.5375233292579651, 0.8453333377838135]
```

測試資料集之準確度為 84.53%，較上次改善之測試準確度多了 0.9%，並無顯著提升，利用此模型為基礎進行下一階段之改善。

(3) 改善方法 3：Convolution layer 之數量改成 7 。

```
[46] model = Models.Sequential()

     model.add(Layers.Conv2D(200,kernel_size=(3,3),activation='relu',input_shape=(150,150,3)))
     model.add(Layers.Conv2D(180,kernel_size=(3,3),activation='relu'))
     model.add(Layers.Conv2D(140,kernel_size=(3,3),activation='relu'))
     model.add(Layers.MaxPool2D(5,5))
     model.add(Layers.Conv2D(180,kernel_size=(3,3),activation='relu'))
     model.add(Layers.Conv2D(140,kernel_size=(3,3),activation='relu'))
     model.add(Layers.Conv2D(100,kernel_size=(3,3),activation='relu'))
     model.add(Layers.Conv2D(50,kernel_size=(3,3),activation='relu'))
     model.add(Layers.MaxPool2D(5,5))
     model.add(Layers.Flatten())
     model.add(Layers.Dense(180,activation='relu'))
     model.add(Layers.Dense(100,activation='relu'))
     model.add(Layers.Dense(50,activation='relu'))
     model.add(Layers.Dropout(rate=0.5))
     model.add(Layers.Dense(6,activation='softmax'))

     model.compile(optimizer=Optimizer.Adam(lr=0.0001),loss='sparse_categorical_crossentropy',metrics=['accuracy'])
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_78 (Conv2D)           (None, 148, 148, 200)     5600
_________________________________________________________________
conv2d_79 (Conv2D)           (None, 146, 146, 180)     324180
_________________________________________________________________
conv2d_80 (Conv2D)           (None, 144, 144, 140)     226940
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 28, 28, 140)       0
_________________________________________________________________
conv2d_81 (Conv2D)           (None, 26, 26, 180)       226980
_________________________________________________________________
conv2d_82 (Conv2D)           (None, 24, 24, 140)       226940
_________________________________________________________________
conv2d_83 (Conv2D)           (None, 22, 22, 100)       126100
_________________________________________________________________
conv2d_84 (Conv2D)           (None, 20, 20, 50)        45050
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 4, 4, 50)          0
_________________________________________________________________
flatten_13 (Flatten)         (None, 800)               0
_________________________________________________________________
dense_52 (Dense)             (None, 180)               144180
_________________________________________________________________
dense_53 (Dense)             (None, 100)               18100
_________________________________________________________________
dense_54 (Dense)             (None, 50)                5050
_________________________________________________________________
dropout_10 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_55 (Dense)             (None, 6)                 306
=================================================================
Total params: 1,349,426
Trainable params: 1,349,426
Non-trainable params: 0
```

改善後之結果：

```
trained = model.fit(Images,Labels,epochs=10,batch_size=32,validation_split=0.20)

Epoch 1/10
351/351 [==============================] - 162s 455ms/step - loss: 1.6032 - accuracy: 0.3518 - val_loss: 1.0363 - val_accuracy: 0.6067
Epoch 2/10
351/351 [==============================] - 159s 454ms/step - loss: 1.1315 - accuracy: 0.5618 - val_loss: 0.8519 - val_accuracy: 0.6879
Epoch 3/10
351/351 [==============================] - 159s 454ms/step - loss: 0.9905 - accuracy: 0.6291 - val_loss: 0.7833 - val_accuracy: 0.7246
Epoch 4/10
351/351 [==============================] - 160s 455ms/step - loss: 0.8524 - accuracy: 0.6840 - val_loss: 0.6735 - val_accuracy: 0.7481
Epoch 5/10
351/351 [==============================] - 160s 455ms/step - loss: 0.7452 - accuracy: 0.7341 - val_loss: 0.6290 - val_accuracy: 0.7795
Epoch 6/10
351/351 [==============================] - 159s 454ms/step - loss: 0.7089 - accuracy: 0.7644 - val_loss: 0.5510 - val_accuracy: 0.8119
Epoch 7/10
351/351 [==============================] - 159s 454ms/step - loss: 0.6226 - accuracy: 0.7937 - val_loss: 0.6200 - val_accuracy: 0.7813
Epoch 8/10
351/351 [==============================] - 159s 453ms/step - loss: 0.5802 - accuracy: 0.8074 - val_loss: 0.5434 - val_accuracy: 0.8090
Epoch 9/10
351/351 [==============================] - 160s 455ms/step - loss: 0.5332 - accuracy: 0.8211 - val_loss: 0.5357 - val_accuracy: 0.8212
Epoch 10/10
351/351 [==============================] - 159s 454ms/step - loss: 0.5022 - accuracy: 0.8329 - val_loss: 0.4949 - val_accuracy: 0.8422
```

訓練資料集之準確度為 83.29%，驗證資料集之準確度為 84.22%。

```
test_images,test_labels = get_images('../content/seg_test/seg_test/')
test_images = np.array(test_images)
test_labels = np.array(test_labels)
model.evaluate(test_images, test_labels,batch_size=32,verbose=1)

94/94 [==============================] - 12s 129ms/step - loss: 0.4691 - accuracy: 0.8457
[0.46908560395240784, 0.8456666469573975]
```

測試資料集之準確度為 84.57%。

(4) 改善結果

	原型	方法一	方法二	方法三
Number of convolution layers	6	6	6	7
Number of pooling layers	2	2	2	2
Number of fully connected layers	4	4	4	4
Activation function	Relu	Relu	Relu	Relu
Classification function of the output layer	Sigmoid	Softmax	Softmax	Softmax
Optimizer	Adam	Adam	Adam	Adam
Learning rate	0.001	0.0001	0.0001	0.0001
Batch size	32	32	64	64
Test accuracy	71%	83.63%	84.53%	84.57%

透過以上的結果可以看出，方法 3 使用之參數設定改善是有較佳的結果，準確度達 84.57%；此外，本研究也嘗試改變如激活函數、輸出層分類函數、優化器等參數，但是成效並不理想。

陸、 結論

6.1 分析與改善小結

因為本次的期末報告所使用的資料集較為簡單，皆為基礎的影像，所以在訓練模型及測試模型時，均能達到較高的準確度，如若想進一步提升測試準確度，可嘗試其他參數之調整或增加訓練及測試的資料量。

6.2 模型及研究限制

此次研究所訓練出的模型，經過了多次的改善、測試，仍無法使測試集的準確率提高到 85%以上，其可能之原因如下：

(1) 六種環境景觀之影像張數不平均，且數量太少。

(2) 由於訓練模型的時間過長，導致此次訓練模型的過程中，無法嘗試所有參數之調整，因此本研究所得之結果僅為所有試驗過的模型中表現最好的，而非所有參數之最佳組合。

6.3 未來改進方向

未來若欲用此 CNN 模型來辨識更為複雜的環境景觀影像，如障礙物照片、不同角度的機車騎士或汽車照片、實際路況等，準確度可能會下降非常多，而為避免此問題，可能需於資料前處理過程中進行更多處理動作，並增加訓練、測試模型的複雜度。

柒、 參考文獻

(1) https://scitechvista.nat.gov.tw/c/sTkg.htm

(2) https://kknews.cc/zh-tw/news/yzar3rk.html

(3) https://www.artc.org.tw/upfiles/ADUpload/knowledge/tw_knowledge_624411812.pdf

(4) https://www.kaggle.com/uzairrj/beg-tut-intel-image-classification-93-76-accur/data

(5) https://www.moneydj.com/kmdj/wiki/wikiviewer.aspx?KeyID=5cd9a0f7-e44e-44a3-af4f-a301acdc6103

(6) https://www.artc.org.tw/upfiles/EditUpload/file/ecHo/201608/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92%E6%96%BC%E6%99%BA%E6%85%A7%E8%BB%8A%E8%BC%9B%E6%87%89%E7%94%A8.pdf