Practice of Autonomous Driving

Project II. Traffic Sign Classifier

P46091204

蔡承穎

航太碩 碩一

內容

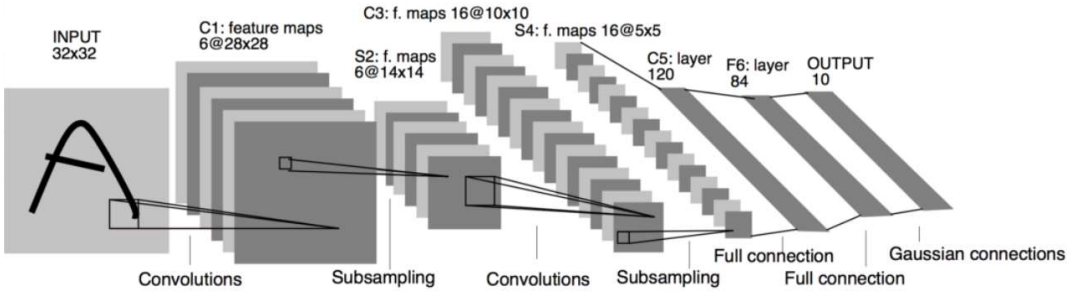
簡介

由於自駕車的概念在近年來受到大眾的關注，其中也得面臨許多挑戰，像是如何建立高精度地圖、在 GPS-denied 的環境下定位、交通號誌辨識等等，此專題是實現如何使用電腦視覺加上深度學習的方式達到辨識交通號誌，深度學習模型選擇 LeNet-5。

將影像丟入網路之前，需要做影像前處理，在這裡會使用到將圖像歸一化再灰度化的方式，歸一化是為了讓網路更 robust，對神經網路有非常大的好處，原因在後面章節講解。

LeNet-5 在 1990 年被提出來，它是個很簡單的 Feature Extractor，基本上就是 Convolution Layer 加上 Max Pooling Layer 的組合。Convolution Layer 可以看成非全連網路，因為圖片的特徵是連續的，不需要連太多神經元。

Convolution 是一個訓練出來的 Kernel 是為了提取圖片特徵，Pooling 是為了降低資料量，物理意義可以看成特徵為連續，取最強的那個數值即可，最後再丟到 Fully-Connected Neural Network。



圖一、LeNet-5

此 Project 已上傳至 Github：

1. Tensorflow 版本

https://github.com/ab458629/Practice-of-Autonomous-Driving/tree/main/Traffic%20Sign%20Classifier/Tensorflow

2. Keras 版本

https://github.com/ab458629/Practice-of-Autonomous-Driving/tree/main/Traffic%20Sign%20Classifier/Keras

此 Project 有寫了兩個版本可以使用，也附上了網路權重可供檢測。

壹、Project 架構

資料分布	張數	影像大小
Training Set	34799	32x32x3
Validation Set	4410	32x32x3
Testing Set	12630	32x32x3

表一、資料分布

在訓練資料網路以前，我們需要準備訓練資料集，這裡切成三個 set，Training Set、Validation Set、Testing Set（表一）。

Tips for Training DNN	
Bad Result on Training Data	(1) New activation function
	(2) Adaptive learning rate
Bad Result on Testing Data	(1) Early stopping
	(2) Regularization
	(3) Drop out

表二、Tips for Training DNN （Hung-yi Lee[1]）

根據前面的資料分布，我們可以根據訓練出來的準確率來調整網路的參數，表二為 Hung-yi Lee 老師提供的訓練網路的 tips，例如，當訓練集的正確率是好的，但驗證集的正確率不好，則可以使用表二提供的方法。

接下來我們使用 LeNet-5 的架構（圖一）作為訓練，訓練圖片時會不斷 minimize cross entropy $C(f(x^n),\hat{y}^n) = -(\hat{y}^n \ln(f^n) + (1-\hat{y}^n)\ln(1-f(x^n)))$，f(x)表示網路輸出，y 是 label，使用 cross entropy 的原因是梯度較平方誤差為明顯，比較快達到目標，因為 update 的變化大。

貳、Project 方法

0. Load the data

```
# Load pickled data
import pickle

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']
```

首先，先將圖片集 load 到 memory，是前面提到的三個 set，X 表示照片，y 表示標籤，也就是圖片種類名稱的 index。

1. Dataset Summary & Exploration

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

```
### Replace each question mark with the appropriate value.
### Use python, pandas or numpy methods rather than hard coding the results
import numpy as np

# Number of training examples
n_train = len(X_train)

# Number of testing examples.
n_test = len(X_test)

# Number of validation examples.
n_valid = len(X_valid)

# What's the shape of an traffic sign image?
image_shape = X_train.shape[1:]

# How many unique classes/labels there are in the dataset.
n_classes = len(np.unique(y_train))

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

```
Number of training examples = 34799
Number of testing examples = 12630
Number of validation examples = 4410
Image data shape = (32, 32, 3)
Number of classes = 43
```

這裡只是將每個資料集的照片個數儲存在變數之中，每張照片的大小為 32x32x3，3 表示 channel 數，共有 43 個種類。

```python
import pandas as pd
df1 = pd.read_csv('./signnames.csv', delimiter=',')
df1.dataframeName = 'signnames.csv'
print(df1.head(20),"\n")
nRow, nCol = df1.shape
# print('There are nRow {nRow}, nCol {nCol}'.format(nRow=nRow, nCol=nCol))
print(f'There are {nRow} rows and {nCol} columns in signnames.csv.')
```

```
    ClassId                                     SignName
0         0                       Speed limit (20km/h)
1         1                       Speed limit (30km/h)
2         2                       Speed limit (50km/h)
3         3                       Speed limit (60km/h)
4         4                       Speed limit (70km/h)
5         5                       Speed limit (80km/h)
6         6                End of speed limit (80km/h)
7         7                      Speed limit (100km/h)
8         8                      Speed limit (120km/h)
9         9                                No passing
10       10  No passing for vehicles over 3.5 metric tons
11       11         Right-of-way at the next intersection
12       12                             Priority road
13       13                                     Yield
14       14                                      Stop
15       15                               No vehicles
16       16      Vehicles over 3.5 metric tons prohibited
17       17                                  No entry
18       18                           General caution
19       19               Dangerous curve to the left

There are 43 rows and 2 columns in signnames.csv.
```

這裡我們可以藉由 panda 模組來整齊顯示出.csv 檔裡頭的東西，並儲存至 df1 中，這裡會顯示出前 20 個種類的 index 與 content。

B. Include an exploratory visualization of the dataset

```
### Data exploration visualization code goes here.
### Feel free to use as many code cells as needed.
# Visualizations will be shown in the notebook.
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import cv2
import random

def getSignNames():
    return pd.read_csv('./signnames.csv').values # [[0 'Speed limit (20km/h)'] [1 'Speed limit (30km/h)'] ... ]


def plotImages(X, y, examples_per_sign=15, squeeze=False, cmap=None):
    samples_per_sign = np.bincount(y) # [ 180 1980 2010 1260 1770 1650  360 1290 1260 1320 1800 1170 ...]
    for sign in getSignNames():
        print(f"{sign[0]}. {sign[1]} - Samples: {samples_per_sign[sign[0]]}")
        sample_indices = np.where(y==sign[0])[0]
        random_samples = random.sample(list(sample_indices), examples_per_sign)
        fig = plt.figure(figsize = (examples_per_sign, 1))
        fig.subplots_adjust(hspace = 0, wspace = 0)
        for i in range(examples_per_sign):
            image = X[random_samples[i]]
            axis = fig.add_subplot(1, examples_per_sign, i+1, xticks=[], yticks=[])
            if squeeze: image = image.squeeze()
            if cmap == None:
                axis.imshow(image)
            else:
                axis.imshow(image.squeeze(), cmap=cmap)
        plt.show()
        print("--------------------------------------------------------------------------------\n")

plotImages(X_train, y_train)
```

若想顯示出種類名稱對應的圖片集，可以使用以上方式，這裡大致講解 plotImages 做了什麼事。

(1) 首先傳入了 X_train 與 y_train，由 np.bincount(y)可以處理標籤的向量，可以算出 y 裡的標籤個數，例如 X=[0,5,3,2,3,1,4,2,4]，print(np.bincount(X))，會顯示出[1,1,2,2,2,1]，表示[indx0 = 1, indx1 = 1, indx2 = 2 …]。

(2) sample_indices = np.where(y==sign[0])[0]，可以找出 y 的哪個位置等於 sign[0]，sign[0]表示前面的 ClassId 的部分。

(3) random_samples = random.sample(list(sample_indices), examples_per_sign)，examples_per_sign 預設 15，表示在這個 list 中，隨機取 15 個。

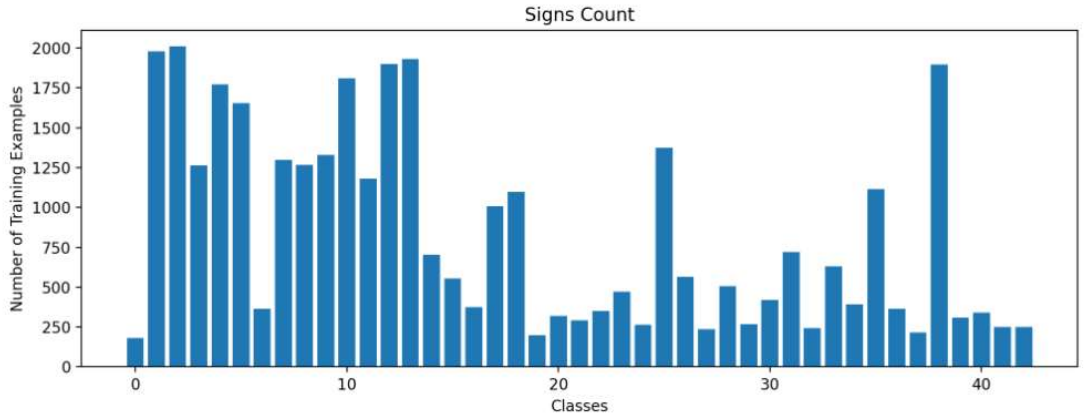
(4) 有了前面 random 取的 index，就能畫圖，畫圖的 code 比較簡單，故省略說明，以下為結果。

```
0. Speed limit (20km/h) - Samples: 180
```

```
--------------------------------------------------------------------------------
1. Speed limit (30km/h) - Samples: 1980
```

```
--------------------------------------------------------------------------------
```

```
# plot the histogram
count=0
cols = 5
hist = np.arange(n_classes)
for i in range(len(y_train)):
    hist[y_train[i]] +=1
fig = plt.figure(figsize=(10, 4), dpi=200, tight_layout=True, linewidth=1)
plt.bar(range(n_classes) , hist)
plt.savefig('plots/bar_chart.png')
plt.title("Signs Count")
plt.xlabel("Classes")
plt.ylabel("Number of Training Examples")
plt.show()
print(df1)
```



接著我們可以很簡單的透過 matplotlib 畫出 bar chart，hist[y_train[i]]就是用來統計每個 bar 的個數，這裡當然可以改寫成用 np.bincount(y)的方式解決。

2. Design and Test a Model Architecture

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

在我們要設計 Model 以前，可以先將 data 做前處理，以免 garbage in garbage out 這件事。

```
1 from sklearn.utils import shuffle
2
3 X_train, y_train = shuffle(X_train, y_train)
```

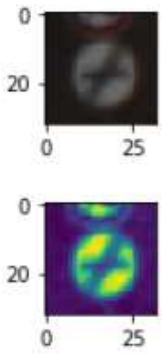
首先先將 training set 打亂

```
import cv2
def prepare_image(image_set):
    """Transform initial set of images so that they are ready to be fed to neural network.

    (1) normalize image
    (2) convert RGB image to gray scale
    """
    # initialize empty image set for prepared images
    new_shape = image_shape[0:2] + (1,)

    prep_image_set = np.empty(shape=(len(image_set),) + new_shape, dtype=int)

    for ind in range(0, len(image_set)):
        # normalize
        norm_img = cv2.normalize(image_set[ind], np.zeros(image_shape[0:2]), 0, 255, cv2.NORM_MINMAX)

        # grayscale
        gray_img = cv2.cvtColor(norm_img, cv2.COLOR_RGB2GRAY)

        # set new image to the corresponding position
        prep_image_set[ind] = np.reshape(gray_img, new_shape)

    return prep_image_set
```

接著寫一個預處理的函數，新的 shape 要變成(32,32,1)，將 RGB 變成 Gray。

(1) cv2.normalize 表示將歸一化，舉個例子，cv2.NORM_MINMAX 算法，設數組原有 $\{A1, A2, \ldots, An\}$，$P = \frac{A_k}{\max(A_i) - \min(A_i)}$，當 A_k 等於 max 則 P=1，當 A_k 等於 min 則 P=0，故假設 $src = \{10, 23, 71\}$ 經過計算後，會得到 $dst = \{0, 0.377, 1\}$。這樣的好處是為了，假設圖片因為 Exposure time、Gain、光圈等影響，或是說 RGB 三個 channel 的圖片是會有差異的，會讓即使原本的圖是同樣物體，但因為這些相機參數不同，若是 RGB 三個通道的分布不同，使得機器認為是不一樣的，讓準確率下降，為了避免此問題，使用歸一化，讓訓練模型更強健。

(2) cv2.cvtColor 則表示將圖片轉換成灰度圖，以下為結果圖。




```python
def equalize_number_of_samples(image_set, image_labels):
    """Make number of samples in each category equal.

    The data set has different number of samples for each category.
    This function will transform the data set in a way that each category
    will contain the number of samples equal to maximum samples per category
    from the initial set. This will provide an equal probability to meet
    traffic sign of each category during the training process.
    """
    num = max([len(np.where(image_labels==cat_id)[0]) for cat_id in signNames.keys()])

    equalized_image_set = np.empty(shape=(num * n_classes,) + image_set.shape[1:], dtype=int)
    equalized_image_labels = np.empty(shape=(num * n_classes,), dtype=int)
    j = 0

    for cat_id in signNames.keys():
        cat_inds = np.where(y_train==cat_id)[0]
        cat_inds_len = len(cat_inds)

        for i in range(0, num):
            equalized_image_set[j] = image_set[cat_inds[i % cat_inds_len]]
            equalized_image_labels[j] = image_labels[cat_inds[i % cat_inds_len]]
            j += 1

    # at this stage data is definitely not randomly shuffled, so shuffle it
    return shuffle(equalized_image_set, equalized_image_labels)
```

這裡因為每個種類分布差異很大，由 bar chart 就能明顯看出來，故此函數是為了讓每個訓練種類的個數相同，讓 training 過程中取到每個種類的機率相同。以下大致講解流程：

(1) num 表示找出最多 sample 數的個數。
(2) equalized_image_set 經由 np.empty 創建一個 shape 為(num*classes, 32, 32, 3)的 array，同理 label 的部分
(3) cat_id 是號誌種類的 index，由 np.where 找出 y_train 裡頭哪些等於 cat_id，並返回此一維索引陣列，這樣就能知道其 index 所在，並且由 len(cat_indx)就能知道個數
(4) 接下來將每個種類一個一個放進 equalized_image_set 與 equalized_image_labels，由 for i in range(num)，traversal 一個一個放進，因為適用%餘數的方式，就能與 num 一樣多，也就是有幾張照片會重複放進。

B. Model Architecture

```
EPOCHS = 30
BATCH_SIZE = 128
KEEP_PROB = 0.5
rate = 0.001

def LeNet(x, keep_prob, channels=1, classes = 43, mu=0, sigma=0.01):

    # Layer 1: Convolutional. Input = 32x32xchannels. Output = 28x28x6.
    conv1_W = tf.Variable(tf.compat.v1.truncated_normal(shape=(5, 5, channels, 6), mean = mu, stddev = sigma))
    conv1_b = tf.Variable(tf.zeros(6))
    conv1   = tf.nn.conv2d(x, conv1_W, strides=[1, 1, 1, 1], padding='VALID') + conv1_b

    # Layer 1: Activation.
    conv1 = tf.nn.relu(conv1)

    # Layer 1: Pooling. Input = 28x28x6. Output = 14x14x6.
    conv1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')

    # Layer 2: Convolutional. Output = 10x10x16.
    conv2_W = tf.Variable(tf.compat.v1.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

    # Layer 2: Activation.
    conv2 = tf.nn.relu(conv2)

    # Layer 2: Pooling. Input = 10x10x16. Output = 5x5x16.
    conv2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='VALID')

    # Layer 2: Flatten. Input = 5x5x16. Output = 400.
    fc0   = tf.compat.v1.layers.flatten(conv2)
    fc0   = tf.nn.dropout(fc0, rate = 1 - keep_prob)

    # Layer 3: Fully Connected. Input = 400. Output = 120.
    fc1_W = tf.Variable(tf.compat.v1.truncated_normal(shape=(400, 120), mean = mu, stddev = sigma))
    fc1_b = tf.Variable(tf.zeros(120))
    fc1   = tf.matmul(fc0, fc1_W) + fc1_b

    # Layer 3: Activation.
    fc1    = tf.nn.relu(fc1)

    # Layer 4: Fully Connected. Input = 120. Output = 84.
    fc2_W  = tf.Variable(tf.compat.v1.truncated_normal(shape=(120, 84), mean = mu, stddev = sigma))
    fc2_b  = tf.Variable(tf.zeros(84))
    fc2    = tf.matmul(fc1, fc2_W) + fc2_b

    # Layer 4: Activation.
    fc2    = tf.nn.relu(fc2)

    # Layer 5: Fully Connected. Input = 84. Output = 10.
    fc3_W  = tf.Variable(tf.compat.v1.truncated_normal(shape=(84, classes), mean = mu, stddev = sigma))
    fc3_b  = tf.Variable(tf.zeros(classes))
    logits = tf.matmul(fc2, fc3_W) + fc3_b

    return logits
```

接下來開始建造我們的網路模型，這裡我們有兩個 Convolution 加 MaxPooing 層，表示我們圖片下取樣了兩次，VALID 表示在做 Convolution 時不會在圖片外圍 zero padding，公式為 $H/W = \frac{H/W - F_{h/w} + 1}{S_{h/w}}$，H 表示高度，W 表示寬度，F 為 Filter 的 size，S 表示 stride 步長，另外 SAME 表示外圍補 0，不會更改圖片大小。Convolution 的 kernel size 為 5x5，stride 為 1，MaxPooling 為 2x2 的格子中選最大值，最後經過 Flatten()丟到一般的全連網路，分別為 400，120，84，最後一層的神經元個數為 class 的個數 43。

其中為了 overfitting，在 400 個神經元的地方加上了 dropout，表示有多少 rate 機率會讓其神經元沒有作用。

這裡設定參數為，迭代次數 Epoch = 20，一個 Batch 大小 Batch size = 128，Drop out 保持神經元機率 Keep prob = 0.5，學習率 rate = 0.0008。

```
### Train your model here.
### Calculate and report the accuracy on the training and validation set.
### Once a final model architecture is selected,
### the accuracy on the test set should be calculated and reported as well.
### Feel free to use as many code cells as needed.
tf.compat.v1.disable_eager_execution()
x = tf.compat.v1.placeholder(tf.float32, shape = (None, 32, 32, 1))
y = tf.compat.v1.placeholder(tf.int32, shape = (None))
keep_prob = tf.compat.v1.placeholder(tf.float32)
one_hot_y = tf.one_hot(y, 43)
```

```
logits = LeNet(x, keep_prob)
cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=one_hot_y, logits=logits)
loss_operation = tf.reduce_mean(cross_entropy)
optimizer = tf.compat.v1.train.AdamOptimizer(learning_rate = rate)
training_operation = optimizer.minimize(loss_operation)

correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(one_hot_y, 1))
accuracy_operation = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
saver = tf.compat.v1.train.Saver()
```

(1) 接下來將輸入與輸出設定成 tensorflow 的佔位符，也就是 placeholder 形式，暫時儲存變量，與前面權重不同的是，它們是 variable 是變量。

(2) 接著將 x 與 y 的格式設定好，輸入 x 大小為(None, 32, 32, 1)，None 在這表示圖片張數，y 則是輸出，要改成 one hot encoding 形式。

(3) cost function 選擇的是 cross entropy，cost operation 就是將網路輸出與 ground truth 標籤所計算出來的 cross entropy 取平均。

(4) 優化器選擇了 Adam 如下式，這也是蠻常使用的優化器，就是加入 momentum 概念的 RMSprop，且在更新中考慮了 bias corection。

$$w_i^{t+1} = w_i^t - \eta \frac{\hat{m}^t}{\sqrt{\hat{v}^t} + \varepsilon}$$

$$where\ m^{t+1} = \beta_1 m^t + (1-\beta_1) g^t,\ v^{t+1} = \beta_2 v^t + (1-\beta_2)(g^t)^2$$

$$and\ \ \hat{m}^t = \frac{m^t}{1-\beta_1},\ \hat{v}^t = \frac{v^t}{1-\beta_2},\ g^t = \frac{\partial L}{\partial w_i}(W^t)$$

其中 learning rate 在這取的是 1e-3，若效果不彰，可以設定成 1e-4，訓練部分就是我們熟悉的梯度下降。

```
def evaluate(X_data, y_data):
    num_examples = len(X_data)
    total_accuracy = 0
    sess = tf.compat.v1.get_default_session()
    for offset in range(0, num_examples, BATCH_SIZE):
        batch_x, batch_y = X_data[offset:offset+BATCH_SIZE], y_data[offset:offset+BATCH_SIZE]
        accuracy = sess.run(accuracy_operation, feed_dict={x: batch_x, y: batch_y, keep_prob: 1})
        total_accuracy += (accuracy * len(batch_x))
    return total_accuracy / num_examples
```

若我們想要計算準確率，可以使用 evaluate 涵式，這裡比較值得一提的是 keep_prob 要設定成 1，因為只有在訓練的時候才要 drop out 神經元。

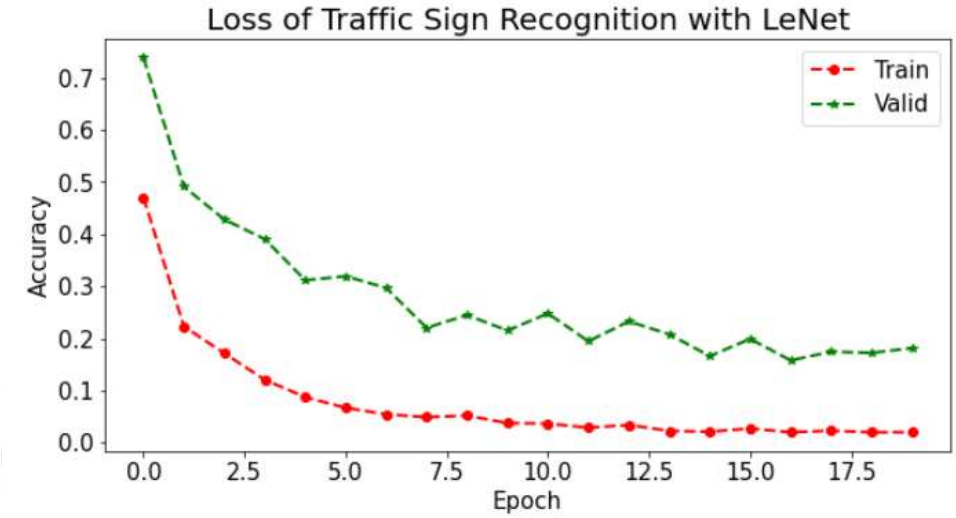
```python
from sklearn.utils import shuffle
loss_array = []
train_acc_array = []
valid_acc_array = []

save_file = "model_data/Lenet.ckpt"
with tf.compat.v1.Session() as sess:
    sess.run(tf.compat.v1.global_variables_initializer())
    num_examples = len(X_train_prep)

    print("Training...")
    print()
    for i in range(EPOCHS):
        X_train_prep, y_train_prep = shuffle(X_train_prep, y_train_prep)
        for offset in range(0, num_examples, BATCH_SIZE):
            end = offset + BATCH_SIZE
            batch_x, batch_y = X_train_prep[offset:end], y_train_prep[offset:end]
            _,loss = sess.run([training_operation, loss_operation], feed_dict={x: batch_x, y: batch_y, keep_prob: KEEP_PROB})

        loss_array.append(loss)
        train_accuracy = evaluate(X_train_prep, y_train_prep)
        train_acc_array.append(train_accuracy)
        validation_accuracy = evaluate(X_valid_prep, y_valid_prep)
        valid_acc_array.append(validation_accuracy)
        print("EPOCH {} ...".format(i+1))
        print("Train Accuracy = {:.3f}".format(train_accuracy))
        print("Validation Accuracy = {:.3f}".format(validation_accuracy))
        print()

    saver.save(sess, save_file)
    print("Model saved")
```

```
Training...

EPOCH 1 ...
Train Accuracy = 0.897
Validation Accuracy = 0.804

EPOCH 2 ...
Train Accuracy = 0.952
Validation Accuracy = 0.877

EPOCH 3 ...
Train Accuracy = 0.969
Validation Accuracy = 0.904

EPOCH 4 ...
Train Accuracy = 0.973
Validation Accuracy = 0.908
```

訓練成果如上，這裡寫了兩個 array 將 training 與 validation 的準確率部分做儲存，與一個 training loss 的 array，以便之後畫圖，當然我們也能使用 tensorboard 來幫助，這裡就不細談。

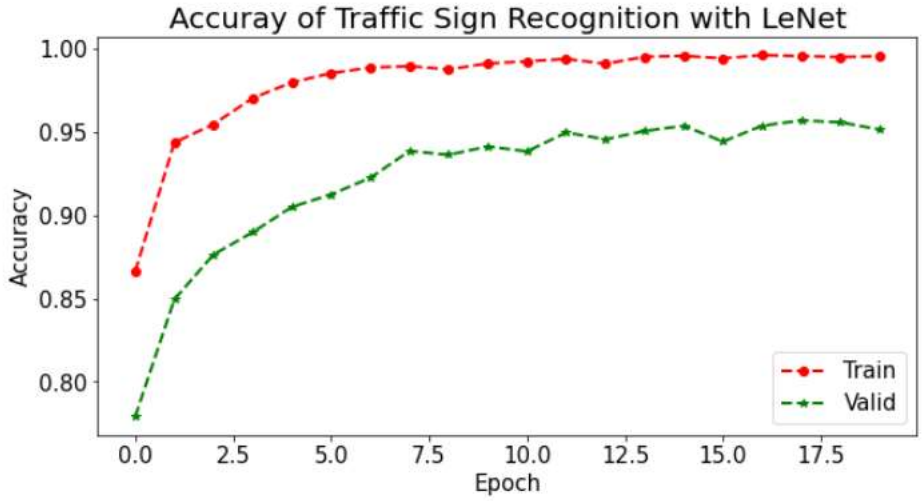
Training Loss 如下，訓練集的 Loss 為 0.02007，驗證集則為 0.18121

```
plt.figure(figsize=(10,5))
plt.plot(train_loss_array,'ro--', linewidth=2)
plt.plot(valid_loss_array,'g*--', linewidth=2)
plt.title('Loss of Traffic Sign Recognition with LeNet', fontsize=20)
plt.ylabel('Accuracy', fontsize=15)
plt.xlabel('Epoch', fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.legend(['Train', 'Valid'], loc='upper right', fontsize=15)
plt.savefig('plots/loss_curve.png')
plt.show()

print("Accuracy of Training = ", train_loss_array[-1])
print("Accuracy of Validation = ", valid_loss_array[-1])
```




```
Accuracy of Training =  0.020067340049058212
Accuracy of Validation =  0.18121259203486889
```

Accuracy 如下，訓練集的準確率為 0.9955，驗證集則為 0.9515，還是有些改進之處，可以使用前面提到的 tips 做改進。




```
Accuracy of Training =  0.9954876778896217
Accuracy of Validation =  0.9514739231728102
```

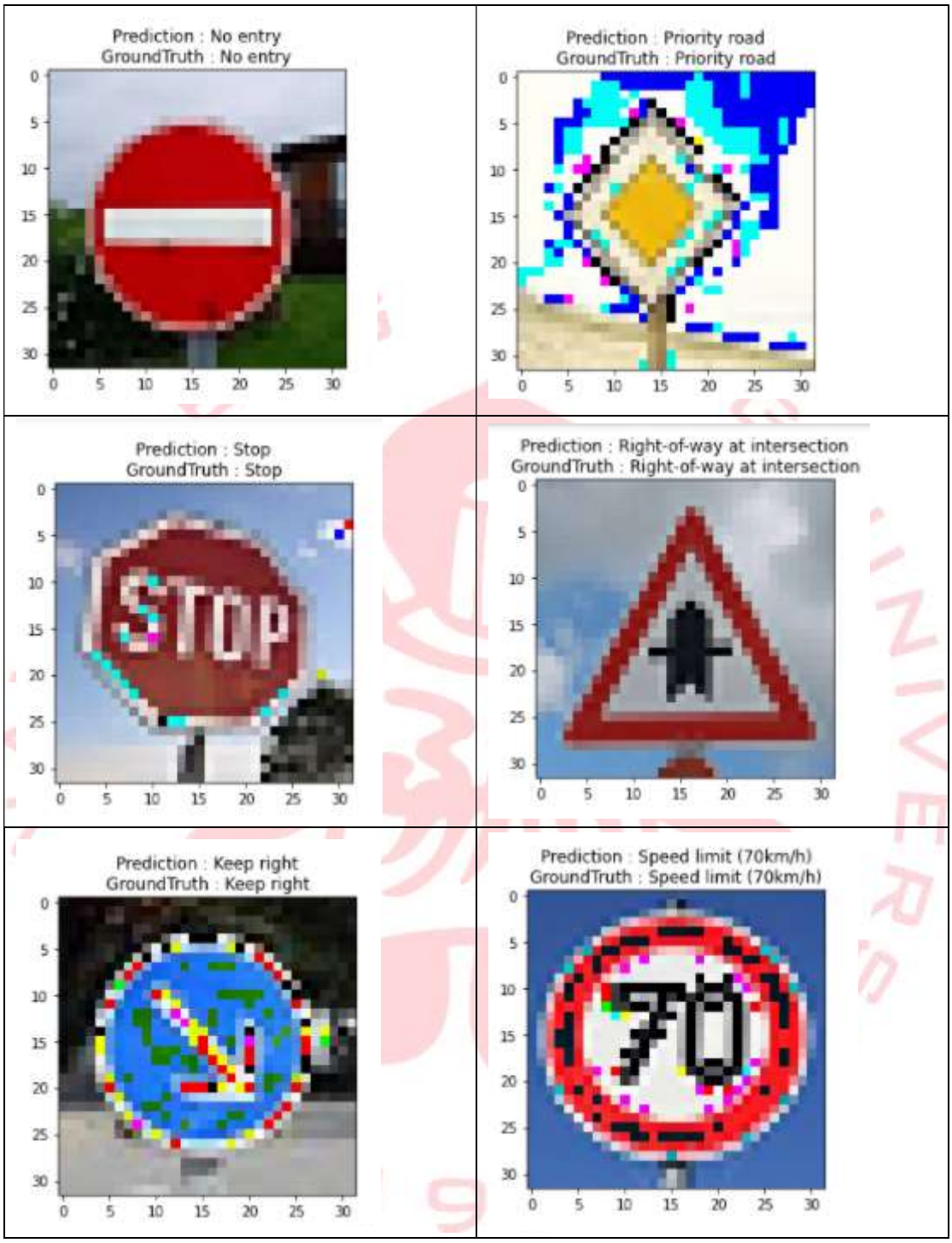
參、結果

1. Predict the Sign Type for Each Image

```
images_prep = prepare_image(X_real)
labels_prep = y_real

# then make a prediction
with tf.compat.v1.Session() as sess:
    saver.restore(sess, "model_data/Lenet.ckpt")
    sign_ids = sess.run(tf.argmax(logits, 1), feed_dict={x: images_prep, y: labels_prep, keep_prob: 1})

# output the results in the table
print('-' * 93)
print("| {p:^43} | {a:^43} |".format(p='PREDICTED', a='ACTUAL'))
print('-' * 93)
for i in range(len(sign_ids)):
    print('| {p:^2} {strp:^40} | {a:^2} {stra:^40} |'.format(
        p=sign_ids[i], strp=signNames[sign_ids[i]], a=y_real[i], stra=signNames[y_real[i]]))
print('-' * 93)

for i in range(len(sign_ids)):
    plt.imshow(X_real[i])
    plt.title('Prediction : ' + signNames[sign_ids[i]] + '\nGroundTruth : ' + signNames[y_real[i]])
    plt.show()
```

```
INFO:tensorflow:Restoring parameters from model_data/Lenet.ckpt
---------------------------------------------------------------------------------------------
|                  PREDICTED                  |                  ACTUAL                   |
---------------------------------------------------------------------------------------------
| 17                No entry                  | 17                No entry                 |
| 12              Priority road               | 12              Priority road              |
| 14                  Stop                    | 14                  Stop                   |
| 11       Right-of-way at intersection       | 11       Right-of-way at intersection      |
| 38              Keep right                  | 38              Keep right                 |
| 4            Speed limit (70km/h)           | 4            Speed limit (70km/h)          |
| 35              Ahead only                  | 35              Ahead only                 |
| 33            Turn right ahead              | 33            Turn right ahead             |
| 25              Road work                   | 25              Road work                  |
| 13                Yield                     | 13                Yield                    |
---------------------------------------------------------------------------------------------
```

這裡我們使用了 10 張 test image 來做測試，左欄表示網路預測結果，右欄則是 Ground truth，這 10 張皆預測正確，結果如下。


Prediction : No entry
GroundTruth : No entry
Prediction : Priority road
GroundTruth : Priority road
Prediction : Stop
GroundTruth : Stop
STOP
Prediction : Right-of-way at intersection
GroundTruth : Right-of-way at intersection
Prediction : Keep right
GroundTruth : Keep right
Prediction : Speed limit (70km/h)
GroundTruth : Speed limit (70km/h)
70

2. Analyze Performance

分析網路在整個 testing set 中的準確率為何，若準確率夠高，表示我們的驗證集接近 testing set。

```
import tensorflow as tf
save_file = "model_data/Lenet.ckpt"
saver = tf.compat.v1.train.Saver()
### Calculate the accuracy for these 5 new images.
### For example, if the model predicted 1 out of 5 signs correctly, it's 20% accurate on these new images.
with tf.compat.v1.Session() as sess:
    saver.restore(sess, save_file)
    test_accuracy, test_loss = evaluate(X_test_prep, y_test_prep, sess)
    print("Test Accuracy = {:.3f}   ".format(test_accuracy))
```

```
INFO:tensorflow:Restoring parameters from model_data/Lenet.ckpt
Test Accuracy = 0.927
```

結果為 0.927，對於此準確率還算堪用，低於 90%以下會考慮重新調整參數，再 train 一次。

3. 自行上網找五張圖片測試

```
INFO:tensorflow:Restoring parameters from model_data/Lenet.ckpt
```

PREDICTED		ACTUAL	
4	Speed limit (70km/h)	4	Speed limit (70km/h)
14	Stop	14	Stop
13	Yield	13	Yield
12	Priority road	40	Roundabout mandatory
38	Keep right	38	Keep right

可以發現這張圖片預測錯誤

肆、心得討論

這裡分兩大部分來討論。

1. 網路深度問題

在 LeNet 提出後，有更多的網路出現，像是古老的 vgg16、vgg19，網路越來越深，但網路越深會有鼎鼎大名的梯度消失問題，故出現了 Residual Block（圖二）的概念，才有 ResNet-50、ResNet-101 更深層的網路，至今還是很強大的網路架構，這裡我也附上我的 Github：https://github.com/ab458629/Computer-vision-and-Deep-learning，裡面包含了用 vgg-16 預測 Cifar10，以及 ResNet-50 處裡貓狗大戰問題。

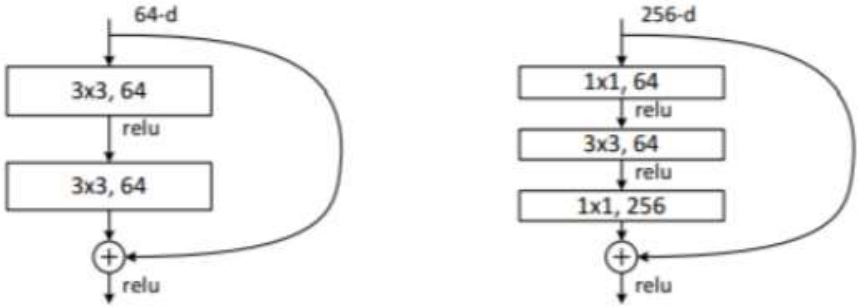
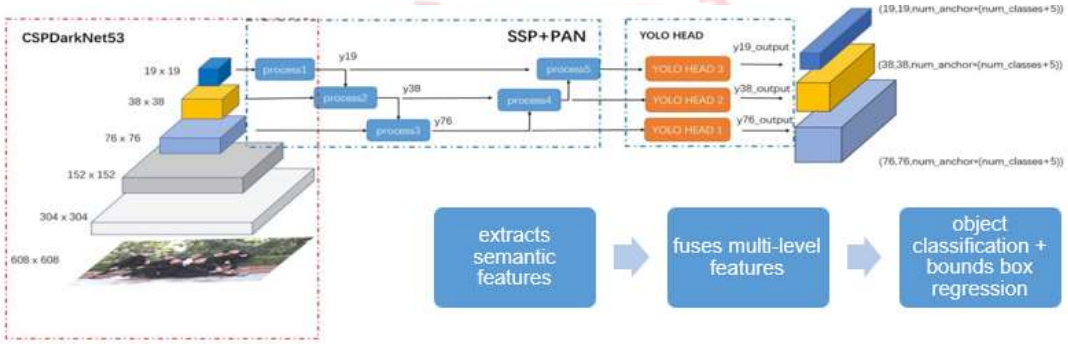


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

圖二、Residual Block

2. 更完整的架構

我們都知道此 Project 只有 backbone 的部分，但影像會有遠近大小的問題，故我們可以加上 neck 的概念之後才丟入 head，也就是 Fuses multi-level features，將影像金字塔的特徵做個合併，這裡 YOLOv4[2]給了一個非常好架構（圖三）。



圖三、Yolo v4 架構

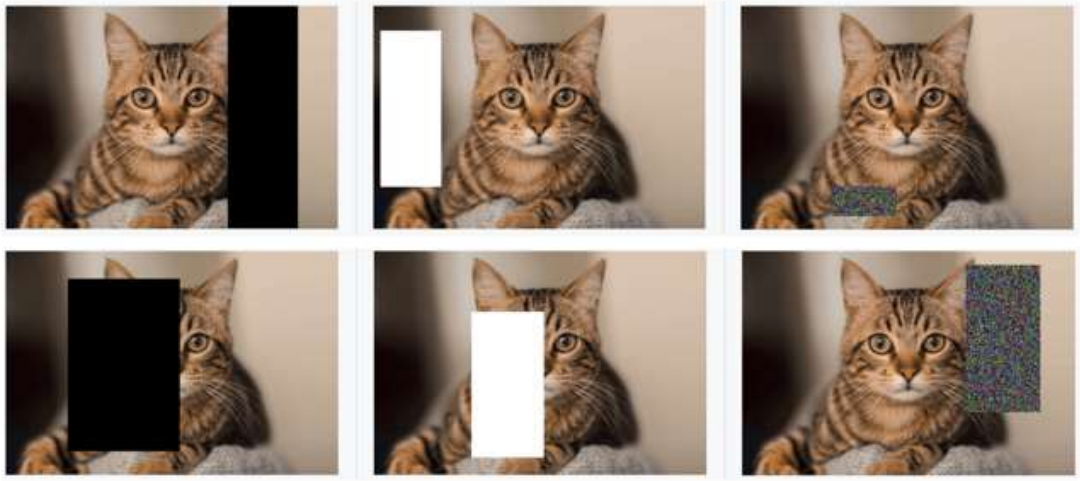
3. Data Argumentation

如 YOLOv4 所提出兩個重點 Bag of Freebies 與 Bag of Specials（圖四），前者為不會增加機器計算量，但會大大提升準確率，後者為增加少許計算量，但也會大大提升準確率的兩個資料擴增的方法。

	Backbone	Detector
Bag of Freebies (BoF)	• CutMix • Mosaic data augmentation • DropBlock • Class label smoothing	• CIoU-loss • Cross mini-Batch Normalization • DropBlock • Mosaic data augmentation • Self-Adversarial Training • Multiple anchors for a single ground truth • Cosine annealing scheduler • Optimal hyperparameters • Random training shapes
Bag of Specials (BoS)	• Mish activation • Cross-stage partial connections (CSP) • Multi-input weighted residual connections (MiWRC)	• Mish activation • SPP-block • SAM-block • PAN path-aggregation block • DIoU-NMS

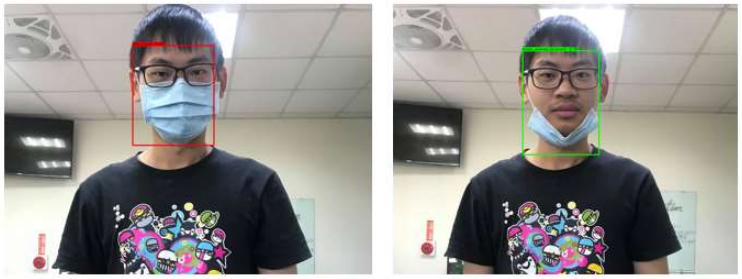
圖四、Bag of Freebies and Specials

我在 train ResNet-50 時，使用了 Random-Erasing 的方式（圖五），提升預測的準確率，其中要注意的是會增加訓練難度，要增加 Epochs 次數。



圖五、Random-Erasing Example

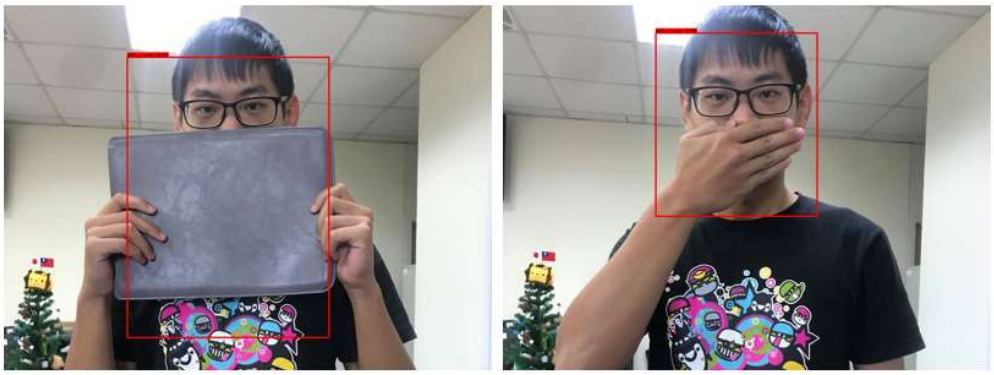
最後附上過去使用 Yolo v4 train 口罩辨識的成果，其中使用 transfer learning，增加訓練速度。

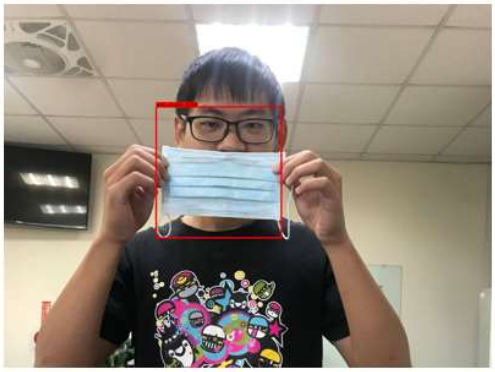


The testing data situation are same as the training dataset, so it can be recognized precisely.

with_mask
mask_weared_incorrect
without_mask

因為訓練資料只有有戴口罩與沒戴口罩兩種，故若使用非口罩來丟入網路是會出現 false-positive 的情況的，故資料的收集在訓練網路是非常重要且必要的。





The training dataset did not contain the block situation, so the result mis-recognized to 'with_mask'

伍、參考資料

[1] Hung-yi Lee , “Tips for Training DNN” , https://www.youtube.com/watch?v=xki61j7z-30&t=659s, Nov 2016

[2] Bochkovskiy, A., C-Y. Wang, and H-Y. Mark Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection(link is external)", arXiv:2004.10934 [cs.CV], 23 April 2020.