此次作業是使用 CNN 的 model 去跑 cifar10 的資料集，Keras 本身就內建提供了 cifar10 的訓練資料集，所以 import 後再透過 load_data()下載資料集後，就可使用。

```
import keras
from keras.datasets import cifar10
```

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

cifar10 的 train data 有 50000 筆，每筆資料為 32x32 的大小，且為 RGB 三原色的彩色圖片，它的 test data 有 10000 筆，每筆資料也是 32x32 的大小，也是 RGB 三原色的彩色圖片。

將資料集載入後，要對圖片進行預先的處理，cifar 的圖片每筆本身都是三維陣列，CNN 輸入層要求每筆資料都是三維陣列，所以不用進行轉換；但因為是彩色圖片，我們要轉換成黑白以提高訓練模型準確率，故會做/255，因為 RGB 的顏色數值為 0～255，黑白圖片的顏色數值為 0～1。

```
x_train = x_train/255
x_test = x_test/255
```

再來要建立 Convolutional Neural Network 的模型，在一開始會先載入所需的相關套件，接著建立 Sequential 順序模組。

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D, Activation
```

我建立的模型是先加入兩層輸入層與卷積層，filter 設為 64，filter 的大小設為 3x3 去做特徵的比對，激活函數使用 relu；再來加入一層池化層以 2x2 進行縮減取樣，圖片的大小就會變為原本的四分之一；接著加入一層 Dropout，其中的隨機 25% Neural 進行隱藏以避免 overfitting；之後我將上述過程中的 filter 大小改為 128 後再重複兩次。

```
#model
model = Sequential()
model.add(Conv2D(64, (3, 3), border_mode='same', activation='relu', input_shape=(32, 32, 3)))
model.add(Conv2D(64, (3, 3), border_mode='same', activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
#model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(128, (3, 3), border_mode='same', activation='relu', input_shape=(32, 32, 3)))
model.add(Conv2D(128, (3, 3), border_mode='same', activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), border_mode='same', activation='relu', input_shape=(32, 32, 3)))
model.add(Conv2D(128, (3, 3), border_mode='same', activation='relu', input_shape=(32, 32, 3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
```

接著模型加入平坦層，將多維度的陣列拉平轉換成一維陣列；再來加入隱藏層，有 1024 個神經元，激活函數使用 relu；最後加入輸出層，因為 class 只有 10 種(airplane、automobile、bird、cat、deer、dog、frog、horse、ship、truck)，所以只有 10 個神經元，激活函數使用 softmax，至此模型建立完畢。

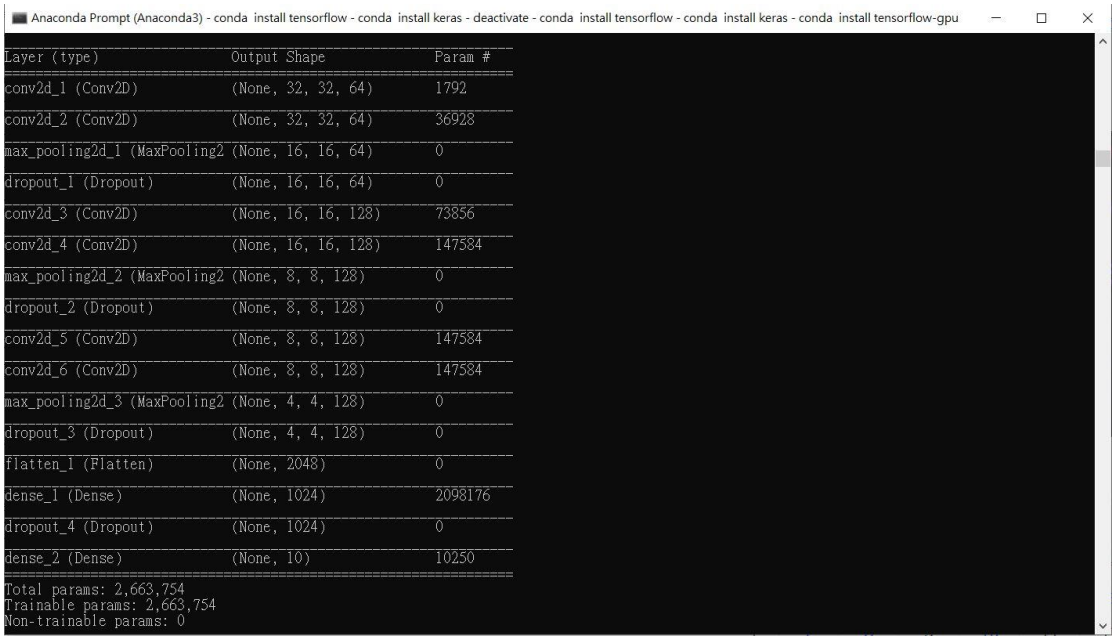
```
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(num_classes, activation='softmax'))
print(model.summary())
```

建立完模型後就要利用模型進行訓練，首先設定模型的訓練方式，設定 Loss 損失函數為 categorical_crossentropy，Optimizer 最佳化方法為 adam，Model 評估準確率方法為 accuracy；再來開始訓練，我的訓練次數設定 10 次，訓練時每批次有 128 筆，設定有 10000 筆作為驗證資料，最後會將 test data 的準確率印出。

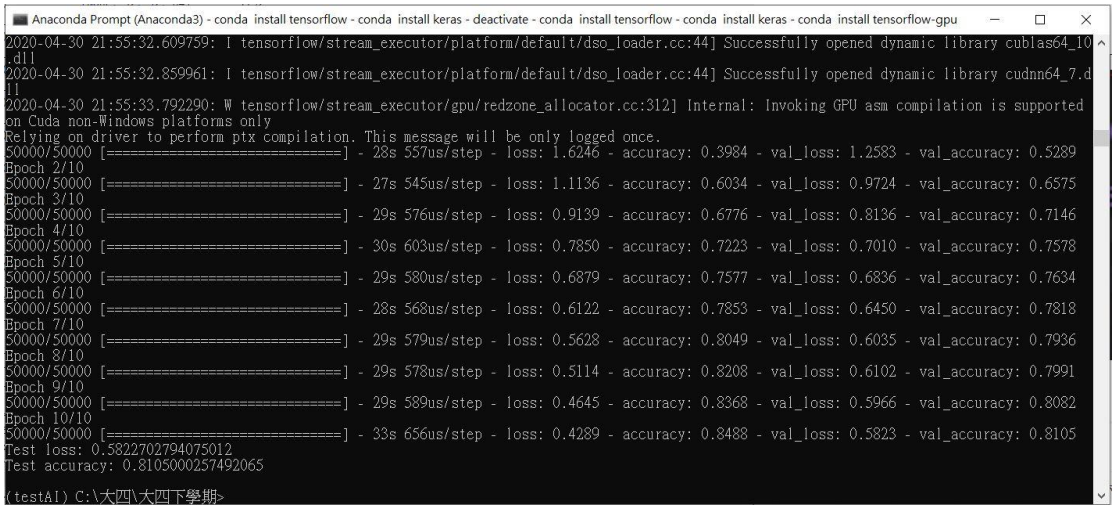
```
#Training
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer='adam',
              metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

在這次作業中最主要的目標就是想辦法提升準確率，所以在建立模型時，一直在試到底需要多少層，然後要如何做參數的設定，才能將準確率提高，一開始我以助教的程式碼來進行訓練，發現準確率大概五成多一些，於是我開始進行一些修改，首先是多加幾層卷積與池化層去跑，發現準確率能提升到七成後，開始對參數進行調整，如將 filter 的數目增加(32、64、128 都有試過)、將 optimizer 改成 adam……等，再試了多種組合與幾次的參數調整之後，下圖是此次作業訓練出來 test accuracy 最高的 model，test accuracy 為 81.05%左右。

模型


Anaconda Prompt (Anaconda3) - conda install tensorflow - conda install keras - deactivate - conda install tensorflow - conda install keras - conda install tensorflow-gpu
Layer (type) Output Shape Param #
conv2d_1 (Conv2D) (None, 32, 32, 64) 1792
conv2d_2 (Conv2D) (None, 32, 32, 64) 36928
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 64) 0
dropout_1 (Dropout) (None, 16, 16, 64) 0
conv2d_3 (Conv2D) (None, 16, 16, 128) 73856
conv2d_4 (Conv2D) (None, 16, 16, 128) 147584
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 128) 0
dropout_2 (Dropout) (None, 8, 8, 128) 0
conv2d_5 (Conv2D) (None, 8, 8, 128) 147584
conv2d_6 (Conv2D) (None, 8, 8, 128) 147584
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128) 0
dropout_3 (Dropout) (None, 4, 4, 128) 0
flatten_1 (Flatten) (None, 2048) 0
dense_1 (Dense) (None, 1024) 2098176
dropout_4 (Dropout) (None, 1024) 0
dense_2 (Dense) (None, 10) 10250
Total params: 2,663,754
Trainable params: 2,663,754
Non-trainable params: 0

準確率截圖


Anaconda Prompt (Anaconda3) - conda install tensorflow - conda install keras - deactivate - conda install tensorflow - conda install keras - conda install tensorflow-gpu
2020-04-30 21:55:32.609759: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cublas64_10.dll
2020-04-30 21:55:32.859961: I tensorflow/stream_executor/platform/default/dso_loader.cc:44] Successfully opened dynamic library cudnn64_7.dll
2020-04-30 21:55:33.792290: W tensorflow/stream_executor/gpu/redzone_allocator.cc:312] Internal: Invoking GPU asm compilation is supported on Cuda non-Windows platforms only
Relying on driver to perform ptx compilation. This message will be only logged once.
50000/50000 [==========] - 28s 557us/step - loss: 1.6246 - accuracy: 0.3984 - val_loss: 1.2583 - val_accuracy: 0.5289
Epoch 2/10
50000/50000 [==========] - 27s 545us/step - loss: 1.1136 - accuracy: 0.6034 - val_loss: 0.9724 - val_accuracy: 0.6575
Epoch 3/10
50000/50000 [==========] - 29s 576us/step - loss: 0.9139 - accuracy: 0.6776 - val_loss: 0.8136 - val_accuracy: 0.7146
Epoch 4/10
50000/50000 [==========] - 30s 603us/step - loss: 0.7850 - accuracy: 0.7223 - val_loss: 0.7010 - val_accuracy: 0.7578
Epoch 5/10
50000/50000 [==========] - 29s 580us/step - loss: 0.6879 - accuracy: 0.7577 - val_loss: 0.6836 - val_accuracy: 0.7634
Epoch 6/10
50000/50000 [==========] - 28s 568us/step - loss: 0.6122 - accuracy: 0.7853 - val_loss: 0.6450 - val_accuracy: 0.7818
Epoch 7/10
50000/50000 [==========] - 29s 579us/step - loss: 0.5628 - accuracy: 0.8049 - val_loss: 0.6035 - val_accuracy: 0.7936
Epoch 8/10
50000/50000 [==========] - 29s 578us/step - loss: 0.5114 - accuracy: 0.8208 - val_loss: 0.6102 - val_accuracy: 0.7991
Epoch 9/10
50000/50000 [==========] - 29s 589us/step - loss: 0.4645 - accuracy: 0.8368 - val_loss: 0.5966 - val_accuracy: 0.8082
Epoch 10/10
50000/50000 [==========] - 33s 656us/step - loss: 0.4289 - accuracy: 0.8488 - val_loss: 0.5823 - val_accuracy: 0.8105
Test loss: 0.5822702794075012
Test accuracy: 0.8105000257492065
(testAI) C:\大四\大四下學期>