CNN Lab

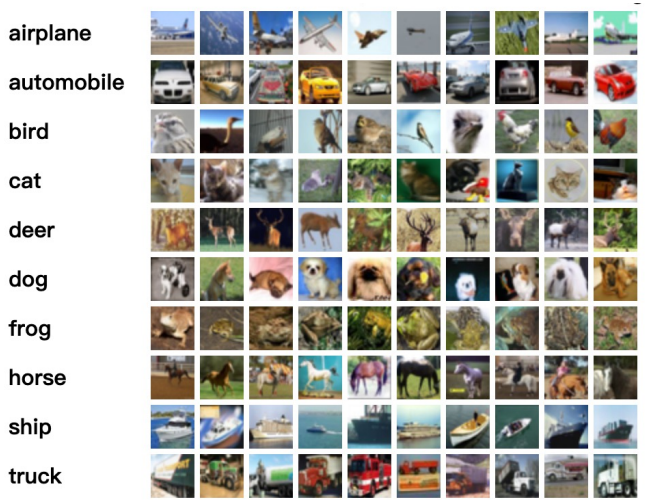
Prof. Chia-Yu Lin

Yuan Ze University

2022 Spring

Dataset

- CIFAR-10
  https://www.cs.toronto.edu/~kriz/cifar.html
- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.



載入Cifar-10資料集

- 執行cifar10.load_data()時，程式會檢查資料夾中是否有cifar10-batches-py.tar檔案
- 如果沒有就會下載並解壓縮，如果之前有執行過，則直接載入資料

```
[ ]:
1 from tensorflow.keras.datasets import cifar10
2
3 (x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 31s 0us/step
```

查看資料的shape

```
1 print('x_train shape:', x_train.shape)
2 print('x_test shape:', x_test.shape)
```

```
x_train shape: (50000, 32, 32, 3)
x_test shape: (10000, 32, 32, 3)
```

(50000, 32, 32, 3)

50000張圖片　32*32　Channel數:3 (RGB圖片)

查看標籤的shape

```
1 print('y_train shape:', y_train.shape)
2 print('y_test shape:', y_test.shape)
```

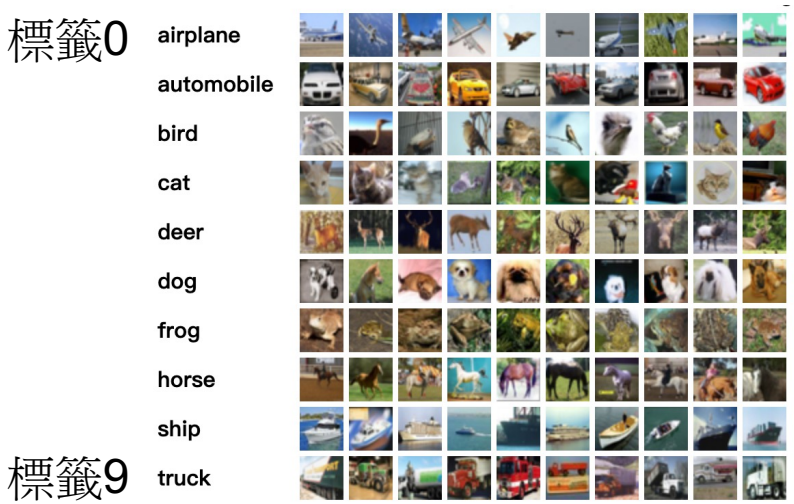
```
y_train shape: (50000, 1)
y_test shape: (10000, 1)
```

查看標籤

- 總共10種類別，標籤數字為0~9

```
1  # 查看前 10 個訓練標籤
2  print(y_train[0: 10])
```

```
[[6]
 [9]
 [9]
 [4]
 [1]
 [1]
 [2]
 [7]
 [8]
 [3]]
```



圖片特徵縮放 (Feature Scaling)

- 圖片送進CNN之前，必須先將圖片的特徵值（像素值）進行特徵縮放 (rescale)
- 特徵縮放的方式有很多種：
  - Min-Max Normalization
  - Mean Normalization
  - Standardization (Z-score Normalization)
  - Scaling to unit length
- 這邊採用Min-Max Normalization

原特徵值　原特徵值的最小值(以圖片像素來說是0)

$$X' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

原特徵值的最大值
(以圖片像素來說是255)

圖片特徵縮放 (Feature Scaling)

- 進行特徵縮放除了可以增加準確率，也可以讓模型訓練時更快收斂

```
x_train_norm = x_train.astype('float32') / 255   # 每個像素值除以 255
x_test_norm = x_test.astype('float32') / 255     # 每個像素值除以 255
```

```
# min-max normalization 後 #
print(x_train_norm[0][0][0])
```

```
[0.23137255 0.24313726 0.24705882]
```

數字標籤轉為One-Hot Encoding

- 目前圖片的標籤是0~9，要轉為One-Hot Encoding
- 使用Keras內建的函式:
  tensorflow.keras.utils.to_categorical()

```
1  from tensorflow.keras import utils
2
3  # 轉換前
4  print(y_train[0])
```

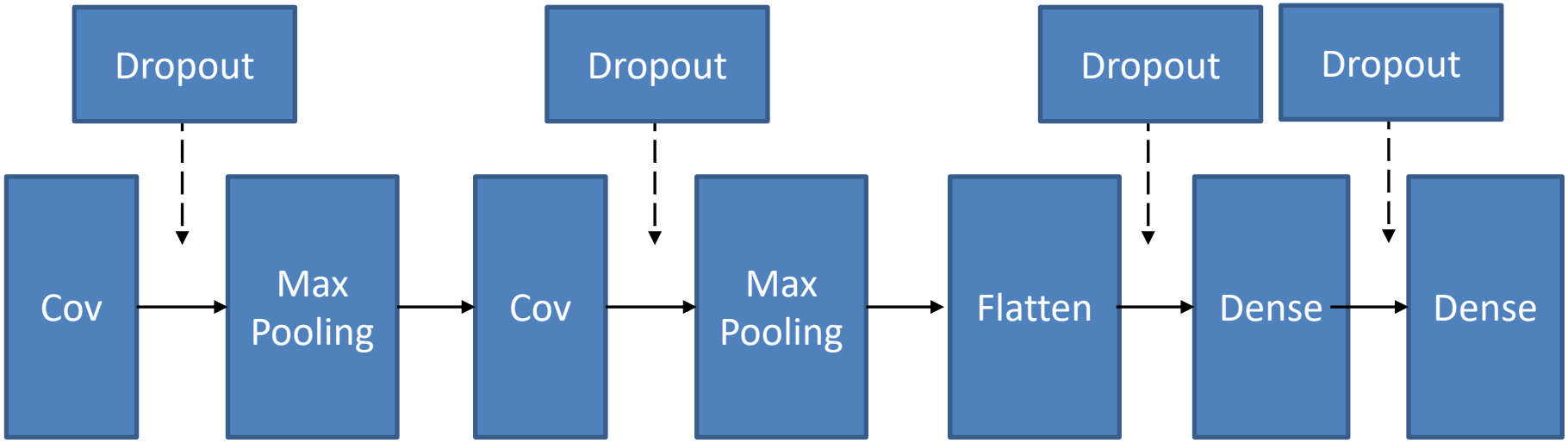
```
[6]
```

```
1  # 進行 One-hot 編碼轉換...
2  y_train_onehot = utils.to_categorical(y_train, 10) # 將訓練標籤進行 One-hot 編碼
3  y_test_onehot = utils.to_categorical(y_test, 10) # 將測試標籤進行 One-hot 編碼
4
5  # 轉換後
6  print(y_train_onehot[0])
```

```
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```

第六個位置為1

建立CNN (1/2)


Dropout
Dropout
Dropout
Dropout
Cov
Max Pooling
Cov
Max Pooling
Flatten
Dense
Dense

建立CNN (2/2)

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

```
cnn = Sequential()
cnn.add(Conv2D(32, (3, 3), activation='relu', padding='same',input_shape=(32, 32, 3)))  # 卷積層 (輸入)
cnn.add(Dropout(0.25))                                                                  # Dropout
cnn.add(MaxPooling2D((2, 2)))                                                           # 池化層

cnn.add(Conv2D(64, (3, 3), padding='same', activation='relu'))                          # 卷積層
cnn.add(Dropout(0.25))                                                                  # Dropout 層
cnn.add(MaxPooling2D((2, 2)))                                                           # 池化層

cnn.add(Flatten())                                                                      # 展平層
cnn.add(Dropout(0.25))                                                                  # Dropout
cnn.add(Dense(1024, activation='relu'))                                                 # 密集層
cnn.add(Dropout(0.25))                                                                  # Dropout
cnn.add(Dense(10, activation='softmax'))                                                # 密集層 (輸出分類)
```

設定CNN的訓練配置

- Loss function
- Optimizer
- Metrics

```
# 神經網路的訓練配置 #
cnn.compile(loss='categorical_crossentropy',     # 損失函數
            optimizer='adam',                    # adam優化器
            metrics=['acc'])                     # 以準確度做為訓練指標
```

進行訓練 (1/2)

```
# -- 進行訓練 -- #
history = cnn.fit(x=x_train_norm,          # 訓練資料
                  y=y_train_onehot,        # 訓練標籤
                  batch_size=128,          # 每個批次用 128 筆資料進行訓練
                  epochs=20,               # 20 個訓練週期 (次數)
                  validation_split = 0.1, # 拿出訓練資料的 10% 做為驗證資料
                  )
```

```
Epoch 1/20
352/352 [==============================] - 33s 93ms/step - loss: 1.8595 - acc: 0.3418 - val_loss: 1.3447 - val_acc:
0.5606
Epoch 2/20
352/352 [==============================] - 52s 149ms/step - loss: 1.2336 - acc: 0.5608 - val_loss: 1.1721 - val_acc:
0.6362
Epoch 3/20
352/352 [==============================] - 42s 118ms/step - loss: 1.0633 - acc: 0.6275 - val_loss: 1.0426 - val_acc:
0.6658
Epoch 4/20
352/352 [==============================] - 40s 113ms/step - loss: 0.9569 - acc: 0.6624 - val_loss: 0.9437 - val_acc:
0.6920
```

⋮

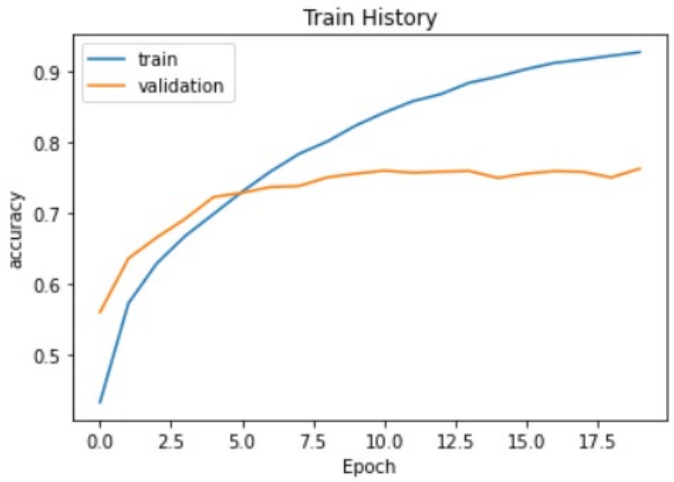
```
Epoch 17/20
352/352 [==============================] - 43s 123ms/step - loss: 0.2372 - acc: 0.9169 - val_loss: 0.7270 - val_acc:
0.7592
Epoch 18/20
352/352 [==============================] - 43s 123ms/step - loss: 0.2273 - acc: 0.9224 - val_loss: 0.7516 - val_acc:
0.7582
Epoch 19/20
352/352 [==============================] - 41s 117ms/step - loss: 0.2146 - acc: 0.9253 - val_loss: 0.7611 - val_acc:
0.7502
Epoch 20/20
352/352 [==============================] - 44s 125ms/step - loss: 0.1983 - acc: 0.9291 - val_loss: 0.7337 - val_acc:
0.7626
```

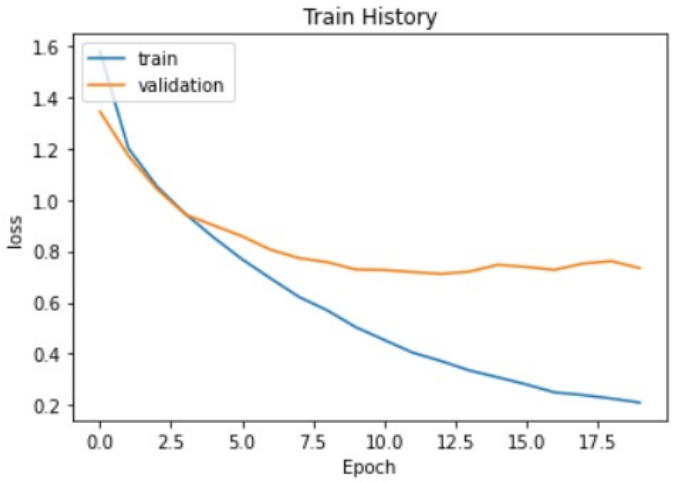
進行訓練 (2/2)

- 訓練準確率(acc)達92.9%
- 驗證準確率(val_acc)達76.2%

繪製損失函數與準確率圖表

- 如何畫？ 回想DNN作業





大概10個epoch
出現Overfitting
情況！

儲存與載入模型

```
# --  儲存模型 -- #
cnn.save('CNN_Model.h5')
```

```
# -- 儲存模型權重-- #
cnn.save_weights('CNN_weights.h5')
```

```
# -- 載入模型 -- #
from tensorflow.keras.models import load_model

old_cnn = load_model('CNN_Model.h5')
print('載入模型成功')
```

載入模型成功

使用測試資料評估模型

- 使用Keras的 evaluate(資料,標籤) 方法，將10000筆的測試資料與測試標籤餵給神經網路

```
1 test_loss, test_val = cnn.evaluate(x_test_norm, y_test_onehot)
2 print('測試資料損失值:', test_loss)
3 print('測試資料準確度:', test_val)
```

```
313/313 [==============================] - 6s 18ms/step - loss: 0.7798 - acc: 0.7514
測試資料損失值: 0.7797853946685791
測試資料準確度: 0.7513999938964844
```

- 跟訓練與驗證準確度差不多，合理

查看神經網路預測結果

- 想看訓練好的神經網路對類別的預測機率，則可使用 predict(資料)

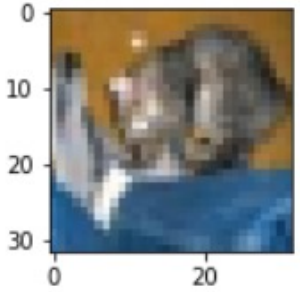
```
1  predict_prop = cnn.predict(x_test_norm)
2  print('第一筆測試資料的預測機率', predict_prop[0])
```

- 第一筆測試資料的預測機率:
- [3.3953902e-03
- 1.0135617e-03
- 3.8660592e-03
- 7.0831507e-01
- 1.8781211e-03
- 2.5143954e-01
- 1.8282762e-02
- 1.2978372e-03
- 9.8135006e-03 ← 預測第8個類別(船)的機率最高0.98
- 6.9811579e-04]

查看圖片

```python
# -- 查看測試資料的第 1 張圖片 -- #
import matplotlib.pyplot as plt

fig = plt.gcf()
fig.set_size_inches(2, 2)
plt.imshow(x_test[0])
plt.show()
```




```python
# -- 查看測試資料的第 2 張圖片 -- #
fig = plt.gcf()
fig.set_size_inches(2, 2)
plt.imshow(x_test[1])
plt.show()
```