鳶尾花分類器

108034544 楊雨澄

清華大學 工業工程與工程管理學系

摘要

平常在路上或是網路圖片想知道一個物種很容易判斷錯誤，本研究希望能透過科技整合做一個分類器來判斷其可能的物種，幫助使用者們以更簡易的方式判斷所希望知道的東西。此次利用著名的鳶尾花（Iris）資料集，內含各樣本的四種特徵屬性及品種，用這個資料集，來建立深度學習模型，判斷及預測其可能的品種。

關鍵字：鳶尾花資料集，深度學習，預測模型，神經網路

一、 前言及文獻探討

平常都會有在路上、花店抑或是在網路上搜尋圖片時，搞錯其代表的物種。分不出非專業的植物學家的我們，其實很難用肉眼就判斷出某些相似度太高的植物甚至是動物的種類，可能上網搜尋相關的可能物種，找他的圖片，或是拿相關書籍做參考判斷，但無法很即時的準確的判斷，這時候就要輔助一些特徵的數值來幫我們做判定的準則。

這個研究希望用一個分類器預測，在具有特定特徵性質的動物或是植物上，可以用這些特徵來判斷該可能的品種，於是希望能透過整理相關特徵數

據，並且用深度學習的方式來建構模型，以得到一個可以判斷我們想知道的動植物的品種。

在這個專案中以一個著名的鳶尾花數聚集作為訓練的資料集，為什麼要挑選這個資料集呢？因為這個資料集已經做好基本的分類整理，並且是一個蠻完整的數據集。他的全名是安德森鳶尾花卉數據集（英文：Anderson's Iris data set），也稱為鳶尾花卉數據集（英文：Iris flower data set）或費雪鳶尾花卉數據集（英文：Fisher's Iris data set），是一類多重變量分析的數據集·安德森從加拿大加斯帕半島上的鳶尾屬花朵中提取的形態學變異數據[1]，後由羅納德·費雪作為判別分析的一個例子[2]，運用到統計學中。

這個數據集包含了 150 個樣本，都屬於鳶尾屬下的三個亞屬，分別是山鳶尾（setosa）、變色鳶尾（verscicolor）和維吉尼亞鳶尾（virginica）。四個特徵被用作樣本的定量分析，它們分別是花萼（sepal）和花瓣（petal）的長度（length）和寬度（width），基於這四個特徵的集合。對於每一個樣本，存在五個資料。其中前四項是花的尺寸的標準測量如上面所述，而第五項是鳶尾花的種類。

所以我的目的就是建立一個分類器，給定兩個花瓣和兩個萼片的測量，就可以預測鳶尾花的種類。

二、　方法

研究的過程將分為：資料前處理、建立模型、訓練模型和測試模型四個階段。在方法的部分主要討論資料前處理、模型的建立與模型的訓練三個部分，測試模型就放在下一個主題探討。

用視覺化的方式呈現資料集（如圖 1），為該 Iris 鳶尾花資料集的資料散佈圖型。可以從散佈圖看出該資料集的特徵分佈都有一定的範圍，且分佈明顯，是適合用來做分類的一個數據集（被稱作是 hello world 等級的數據集）。

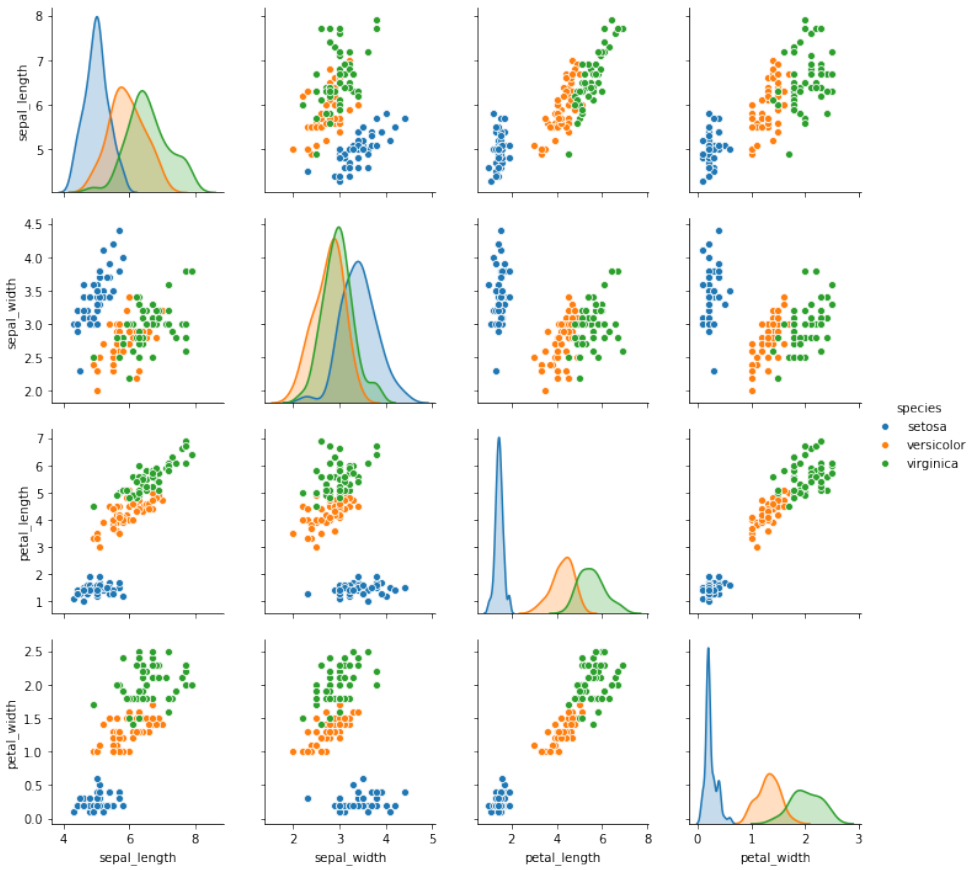

圖 1: Iris 鳶尾花資料集的資料分布圖

（一）資料前處理

首先，在網上下載 iris 的資料集之後，載入 google colab 中，因為這個資料及已經相當完整，不用特別進行太多資料預處理及特徵工程的部分。用 iris.head() 印出首五行觀察此 csv 檔案之形式（如圖 2），可以看出有五個項目第一行是編碼 0-149 個，總共 150 個資料，分為 sepal_length（花萼長），sepal_width（花萼寬），petal_length（花瓣長），petal_width（花瓣寬）和 class（種類）也就是該花的品種。

```
IRIS = pd.read_csv('iris.csv')
IRIS.head()
```

	sepal_length	sepal_width	petal_length	petal_width	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

圖 2: iris.csv 檔案

資料的處理（如圖 3），首先，先設定目標變量，因為要使用前四種特徵來判斷該花屬於哪一種，於是將最後一行 class 設定為目標變量。第二，確定我們要使用的數據概的特徵，也就是除了 class 這個目標變量以外的四個特徵。第三，

用 np.unique()去除目標變量中的元素，並依照元素由大到小排列。第四，增加一列 target 將目標變量進行編碼（以範例為例 0, 1, 2）(如圖 4)。

```
[42] def load_data(path='iris.csv'):
         iris = pd.read_csv('iris.csv')
         target = 'class'
         features = list(iris.columns)
         features.remove(target)
         Class = iris[target].unique()
         Class_dict = dict(zip(Class, range(len(Class))))
         iris['target'] = IRIS[target].apply(lambda x: Class_dict[x])
         lb = LabelBinarizer()
         lb.fit(list(Class_dict.values()))
         labels = lb.transform(iris['target'])
         y_labels = []
         for i in range(labels.shape[1]):
             y_labels.append('y' + str(i))
             iris['y' + str(i)] = labels[:, i]
```

圖 3: 資料處理的過程

```
[73] print (Class_dict)

     {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
```

圖 4: 種類的 dictionary 內容

使用 sklearn.model_selection.train_test_split 隨機劃分訓練集和測試集，

```
train_x, test_x, train_y, test_y = train_test_split(IRIS[features], IRIS[y_bin_labels], train_size=0.7, test_size=0.3, random_state=0)
return train_x, test_x, train_y, test_y, Class_dict
```

用 7、3 分，train_size 為 0.7 和 test_size 為 0.3 的切割（如圖 4）。

圖 4: 訓練及測試及的切割

（二）建立模型

1. 最終模型架構

最終模型架構（如圖 5），所使用之參數顯示在表格 1 之中。最終卷基層數為 3，Activation Function（激勵函數）為 Relu，Loss Function（損失函數）為 Categorical_crossentropy，Optimizer（優化器）為 Adam，輸出層的分類函數為 Softmax。

```
[48] init = K.initializers.glorot_uniform(seed=1)
     adam = K.optimizers.Adam()
     adagrad = K.optimizers.Adagrad()
     model = K.models.Sequential()
     model.add(K.layers.Dense(units=5, input_dim=4, kernel_initializer=init, activation='relu'))
     model.add(K.layers.Dense(units=6, kernel_initializer=init, activation='relu'))
     model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
     model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
```

圖 5: 最終模型架構圖

表 1: 模型使用之參數

CNN parameter	Value
Number of convolution layers	3
Activation function	Relu
Loss function	Categorical_crossentropy
Optimizer	adam

Classification function of the output layer	Softmax

2. 調整參數/模型，以提升模型的表現。

圖 6 顯示為原有的模型設定，正確率表現為 66.67%。

```
[95]  #定義模型
      init = K.initializers.glorot_uniform(seed=1)
      adam = K.optimizers.Adam()
      adagrad = K.optimizers.Adagrad()
      model = K.models.Sequential()
      model.add(K.layers.Dense(units=10, input_dim=4, kernel_initializer=init, activation='sigmoid'))
      model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
      model.compile(loss='categorical_crossentropy', optimizer=adagrad, metrics=['accuracy'])

      #評估模型
      eval = model.evaluate(test_x, test_y, verbose=0)
      print("Evaluation on test data: loss = %0.6f accuracy = %0.2f%% \n" % (eval[0], eval[1] * 100) )

      Evaluation on test data: loss = 0.558617 accuracy = 66.67%
```

圖 6: 原本的模型架構及正確率

(1) 增加模型層數

首先將考慮將原本模型多增加一層，並將 Units 設為 15，來看

```
[75]    #定義模型
        init = K.initializers.glorot_uniform(seed=1)
        adam = K.optimizers.Adam()
        adagrad = K.optimizers.Adagrad()
        model = K.models.Sequential()
        model.add(K.layers.Dense(units=10, input_dim=4, kernel_initializer=init, activation='sigmoid'))
        model.add(K.layers.Dense(units=15, input_dim=4, kernel_initializer=init, activation='sigmoid'))
        model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
        model.compile(loss='categorical_crossentropy', optimizer=adagrad, metrics=['accuracy'])

        #評估模型
        eval = model.evaluate(test_x, test_y, verbose=0)
        print("Evaluation on test data: loss = %0.6f accuracy = %0.2f%% \n" % (eval[0], eval[1] * 100) )

Evaluation on test data: loss = 0.498954 accuracy = 77.78%
```

他的正確率變化。發現其正確率提升至 77.78%（如圖 7）。

圖 7: 增加模型層數

(2) 調整參數 Activation：Sigmoid ➔ Relu

由圖 7 之模型，將 Activation Funcion( )從 Sigmoid 改為 Relu，

```
[101]   #定義模型
        init = K.initializers.glorot_uniform(seed=1)
        adam = K.optimizers.Adam()
        adagrad = K.optimizers.Adagrad()
        model = K.models.Sequential()
        model.add(K.layers.Dense(units=10, input_dim=4, kernel_initializer=init, activation='relu'))
        model.add(K.layers.Dense(units=15, kernel_initializer=init, activation='relu'))
        model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
        model.compile(loss='categorical_crossentropy', optimizer=adagrad, metrics=['accuracy'])

        #評估模型
        eval = model.evaluate(test_x, test_y, verbose=0)
        print("Evaluation on test data: loss = %0.6f accuracy = %0.2f%% \n" % (eval[0], eval[1] * 100) )

Evaluation on test data: loss = 0.174920 accuracy = 93.33%
```

來看他的正確率變化。發現其正確率提升至 93.33%（如圖 8）。

圖 8: 更換 Activation Function

(3) 調整 Units：減少

為了讓成功率提升更多，由圖 8 的模型，改善其卷積層每一層的

```
[83]    #定義模型
        init = K.initializers.glorot_uniform(seed=1)
        adam = K.optimizers.Adam()
        adagrad = K.optimizers.Adagrad()
        model = K.models.Sequential()
        model.add(K.layers.Dense(units=5, input_dim=4, kernel_initializer=init, activation='relu'))
        model.add(K.layers.Dense(units=6, input_dim=4, kernel_initializer=init, activation='relu'))
        model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
        model.compile(loss='categorical_crossentropy', optimizer=adagrad, metrics=['accuracy'])
```

```
        #評估模型
        eval = model.evaluate(test_x, test_y, verbose=0)
        print("Evaluation on test data: loss = %0.6f accuracy = %0.2f%% \n" % (eval[0], eval[1] * 100) )
```

```
Evaluation on test data: loss = 0.164386 accuracy = 95.56%
```

units 數量，第一層從 10 改為 5，第二層由 15 改為 6，觀察其正確率的變化。發現其正確率提升至 95.56%（如圖 9）。

圖 9: 更改卷積層的 units 數量

(4) 調整 Optimizer：Adagrad ➔ Adam

因為離 100%還有改善的空間，於是繼續嘗試改善模型的結構，調整 Optimizer，從 Adagrad 改為 Adam，來觀察正確率的變化。

```
[ ]     #定義模型
        init = K.initializers.glorot_uniform(seed=1)
        adam = K.optimizers.Adam()
        model = K.models.Sequential()
        model.add(K.layers.Dense(units=5, input_dim=4, kernel_initializer=init, activation='relu'))
        model.add(K.layers.Dense(units=6, kernel_initializer=init, activation='relu'))
        model.add(K.layers.Dense(units=3, kernel_initializer=init, activation='softmax'))
        model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
```

```
[26]    #評估模型
        eval = model.evaluate(test_x, test_y, verbose=0)
        print("Evaluation on test data: loss = %0.6f accuracy = %0.2f%% \n" % (eval[0], eval[1] * 100) )
```

```
Evaluation on test data: loss = 0.110360 accuracy = 97.78%
```

發現其正確率提升至 97.78%（如圖 10）。這也是我目前嘗試到最高的正確率。於是也確定了最後的 IRIS 模型架構。

圖 10: 更改優化器

將上述的模型改善步驟整理如下表格 2 之中 (表 2)。經過改善調整之後，整個模型的正確率提升了 31.11 (66.67% ➔ 97.78%)

表 2:改善步驟

改善步驟	Accuracy
原始 model	**66.67%**
原始 model+層數	66.67% ➔ **77.78%** (↑11.11)
改 Activation Function	77.78% ➔ **93.33%** (↑15.55)
改 Units 數	93.33% ➔ **95.56%** (↑2.23)
改 Optimizer	95.56% ➔ **97.78%** (↑2.22)

(三) 訓練模型

以下為模型的訓練過程，相關的參數設定如下，並且得到訓練的時間 (如圖 11、圖 12)。

- Epochs = 100
- batch （批量） = 1

- Training Time （總訓練時間） = 11.573 sec

```
import time
start_time = time.time()
b_size = 1
epochs = 100
print("Starting training")
history = model.fit(train_x, train_y, batch_size=b_size, epochs=epochs, shuffle=True, verbose=1)
print("Training finished \n")
print("--- %s seconds ---" % (time.time()-start_time))
```

```
Starting training
Epoch 1/100
105/105 [==============================] - 0s 4ms/step - loss: 1.0548 - acc: 0.2952
Epoch 2/100
105/105 [==============================] - 0s 1ms/step - loss: 0.9577 - acc: 0.5333
Epoch 3/100
105/105 [==============================] - 0s 1ms/step - loss: 0.8776 - acc: 0.6286
Epoch 4/100
105/105 [==============================] - 0s 1ms/step - loss: 0.8231 - acc: 0.7048
Epoch 5/100
105/105 [==============================] - 0s 1ms/step - loss: 0.7788 - acc: 0.7143
Epoch 6/100
```

圖 11:訓練模型上半段

```
Epoch 96/100
105/105 [==============================] - 0s 1ms/step - loss: 0.0811 - acc: 0.9524
Epoch 97/100
105/105 [==============================] - 0s 1ms/step - loss: 0.0310 - acc: 0.9810
Epoch 98/100
105/105 [==============================] - 0s 1ms/step - loss: 0.0447 - acc: 0.9905
Epoch 99/100
105/105 [==============================] - 0s 1ms/step - loss: 0.0285 - acc: 0.9905
Epoch 100/100
105/105 [==============================] - 0s 1ms/step - loss: 0.0412 - acc: 0.9714
Training finished

--- 11.572912693023682 seconds ---
```

圖 12:訓練模型下半段

三、 測試模型

利用已經建立好的模型去測試新的一筆數據，以測試該模型是不是真的能成功地，在給定兩個花瓣和兩個萼片的資料到模型中使用。這邊測試的資料為兩組，資料的數據跟原本的 Iris.csv 檔案的格式相同（sepal_length, sepal_width, petal_length, petal_width)第一組資料為 :(2.1, 3.1, 4.1, 5.1)，第二組為 :(6.1, 3.1, 5.1, 1.1)，放進模型中去判定為哪一個種類的花（如圖 13)。並將預測的結果放在結果與討論中討論。

```
new = np.array([[2.2, 3.2, 3.1, 5.2]], dtype=np.float32)
predicted = model.predict(new)
print("\n用模型預測四個特徵: ")
print(new)
print("\n預測的softmax向量為: ")
print(predicted)
new_dict = {v:k for k,v in Class_dict.items()}
print("\n預測的種類為: ")
print(new_dict[np.argmax(predicted)])
```

圖 13: 特徵預測程式碼

四、 結果與討論

（一） 訓練模型結果

經過參數的調整以及模型的調整之後，最終訓練好的模型結果（如圖 14)，由圖 14 中可以看出 Iris 模型的正確率為 97.78%，為測試到目前可

```
[26]  #評估模型
      eval = model.evaluate(test_x, test_y, verbose=0)
      print("Evaluation on test data: loss = %0.6f accuracy = %0.2f%% \n" % (eval[0], eval[1] * 100) )

      Evaluation on test data: loss = 0.110360 accuracy = 97.78%
```

以達到的最高值，猜測可能是因為資料量過少（只有 150 筆）導致無法完整的訓練。

圖 14:訓練模型評估

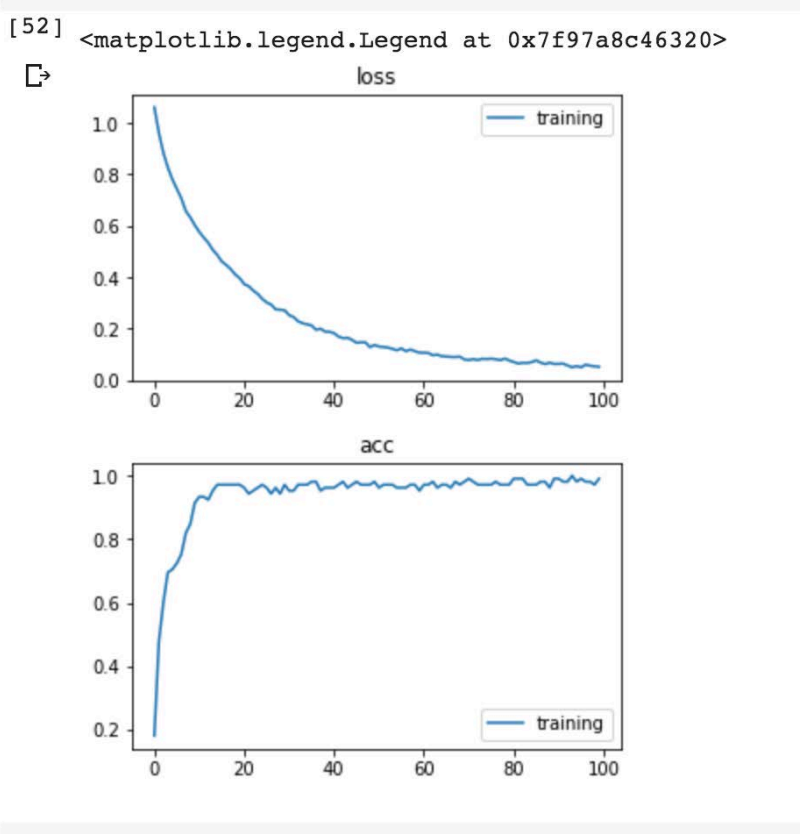

圖 15: 預測模型的 Loss 和 Accuracy 圖

從圖 14 中可以看出 Loss 是有穩定下降的且波動幅度及小，表示是個成功的訓練過程。再來是使用模型測試去探討該模型是否能成功使用特徵來預測其花的種類。

（二） 測試模型的結果

由提供的兩組新數據，導入模型中去做預測，以下三行是從 iris.csv 檔案中，隨意選出的三個品種的數據，可以大致上看出他們的差異。

4.9 3.1 1.5 0.1 Iris-setosa

5.5 2.5 4.0 1.3 Iris-versicolor

6.3 3.4 5.6 2.4 Iris-virginica

```
np.set_printoptions(precision=4)
new = np.array([[6.1, 3.1, 5.1, 1.1]], dtype=np.float32)
predicted = model.predict(new)
print("用模型預測四個特徵: ")
print(new)
print("\n預測的softmax向量為: ")
print(predicted)
new_dict = {v:k for k,v in Class_dict.items()}
print("\n預測的種類為: ")
print(new_dict[np.argmax(predicted)])
```

```
用模型預測四個特徵:
[[6.1 3.1 5.1 1.1]]

預測的softmax向量為:
[[0.0063 0.9676 0.0261]]

預測的種類為:
Iris-versicolor
```

圖 16: 模型預測第一種

```
用模型預測四個特徵:
[[2.2 3.2 3.1 5.2]]

預測的softmax向量為:
[[0.0027 0.0378 0.9595]]

預測的種類為:
Iris-virginica
```

```
new = np.array([[2.2, 3.2, 3.1, 5.2]], dtype=np.float32)
predicted = model.predict(new)
print("\n用模型預測四個特徵: ")
print(new)
print("\n預測的softmax向量為: ")
print(predicted)
new_dict = {v:k for k,v in Class_dict.items()}
print("\n預測的種類為: ")
print(new_dict[np.argmax(predicted)])
```

圖 17: 模型預測第二種

從上面圖 16 及圖 17，可以判斷給定四個特徵的資料去做模型預測並且判斷花的種類是可行的，從 predicted 的 softmax 向量可以判斷其可能的種類（數值最大者），同理能套用在有相同四種類型特徵的不同品種花朵。

五、 未來研究方向

目前的瓶頸及困難是程式語言的能力缺乏，需要再更精進。另外這個專案是利用整理好完整數據的 Iris 資料集去做模型訓練預測測試，可以從訓練的結果數據以及預測新數據，這個過程判斷以該方法建立模型且預測一組新的數據為可行的。之後希望可以朝著整理其他種類特徵的動物或是植物品種，並用相似的方法建立模型，擴展訓練的資料量，來觀察其預測的結果。也希望能發展出一個預測的網頁或是 app，在輸入數據之後，就

能顯現出可能的結果，讓需要使用的人可以更方便的判斷，想知道的動植物可能為何。

參考資料

[1] Edgar Anderson. The irises of the Gaspé Peninsula. *Bulletin of the American Iris Society*. 1935, 59: 2–5.

[2] Fisher, R.A. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*. 1936, 7: 179–188.

[3] Deng, L.; Yu, D. Deep Learning: Methods and Applications. Foundations and Trends in Signal Processing. 2014, 7: 3–4.

[4] 谷歌收購 DNNresearch，下一個帝國呼之欲出. CSDN. 2013-03-13 [2019-12-20].

[5] Bengio, Yoshua; LeCun, Yann; Hinton, Geoffrey. Deep Learning. Nature. 2015, 521: 436–444.

[6] G. Alain and Y. Bengio. What regularized autoencoders learn from the data generating distribution. *In Proceedings of International Conference on Learning Representations (ICLR)*. 2013.

[7] J. Ba and B. Frey. Adaptive dropout for training deep neural networks. In Proceedings of Neural Information Processing Systems (NIPS). 2013.

[8] Keras 教程 - https://zhuanlan.zhihu.com/p/23748037[2019-12-20]

[9] Keras: 基於 Python 的深度學習庫 - https://keras-cn.readthedocs.io/en/latest/[2019-12-20]

[10] Pandas 基礎教學 - https://oranwind.org/python-pandas-ji-chu-jiao-xue/[2019-12-20]

[11] 助教上課內容(W6-CNN)：Keras Documentation - https://keras.io/layers/convolutional/

[12] 助教上課內容(W6-CNN)：Tensorflow - https://www.tensorflow.org/