MNIST 簡介

MNIST手寫數字辨識資料集

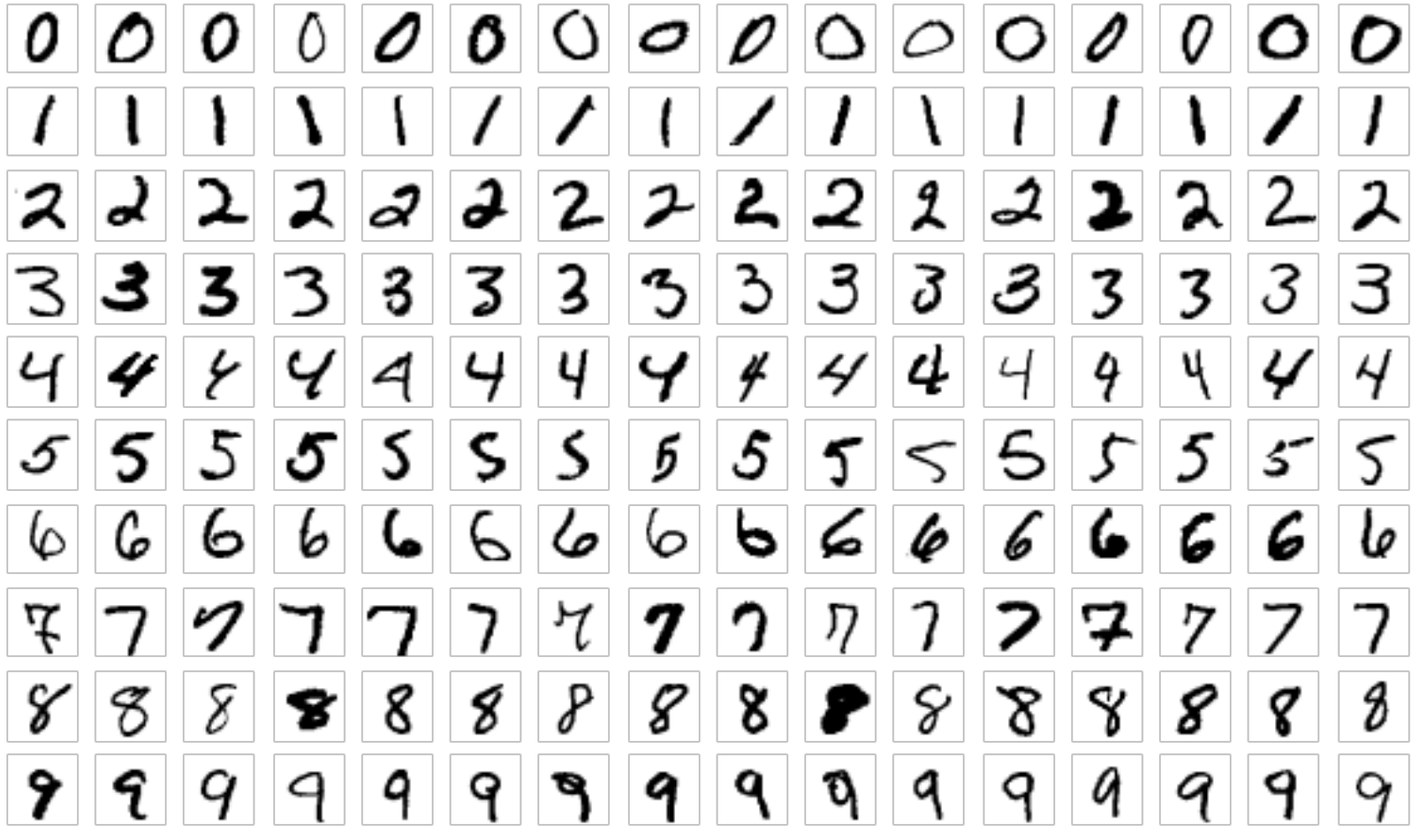
THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

- 資料量不會太多，單色較簡單
- 適於練習建立模型，訓練，預測

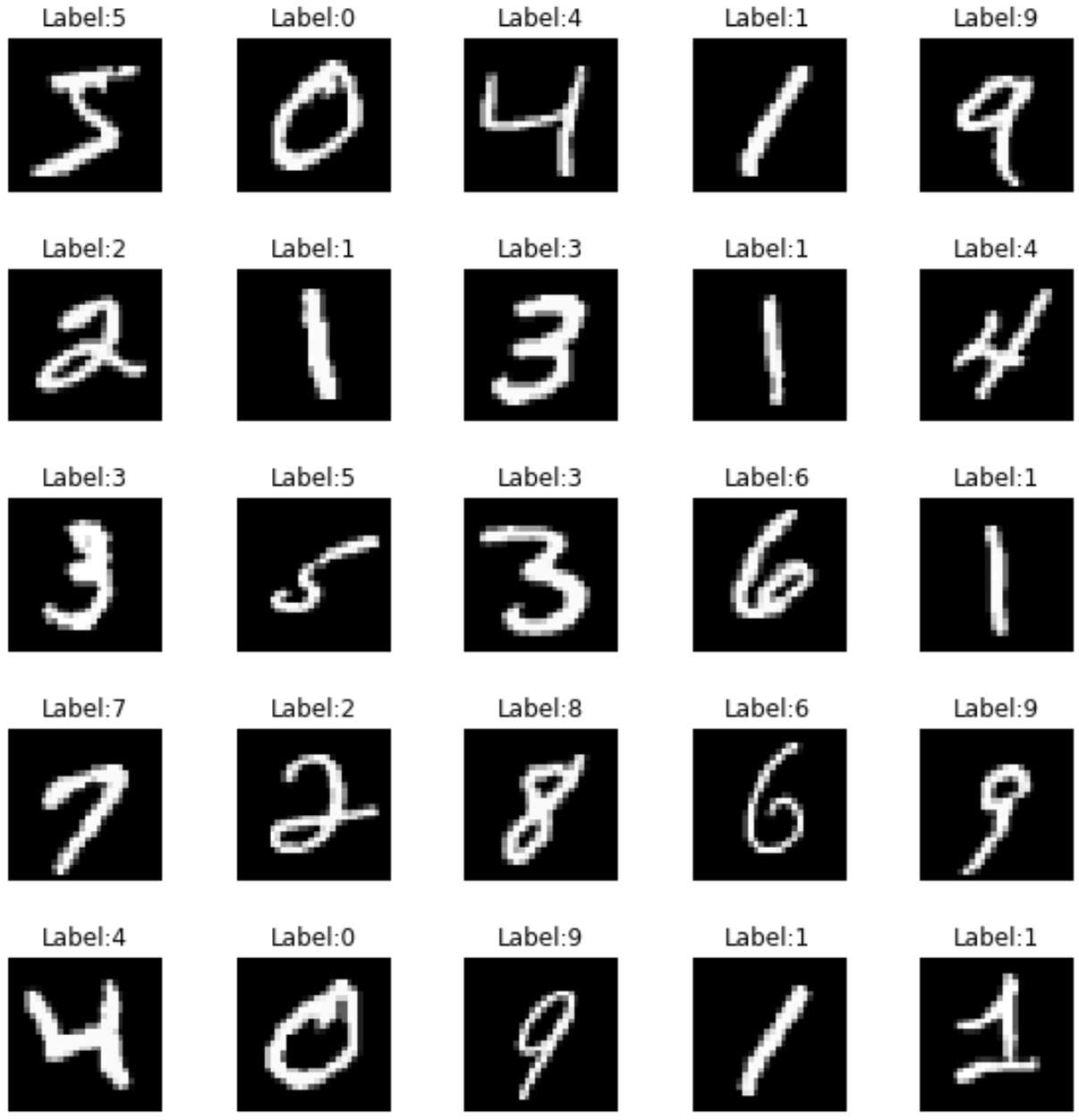
- MNIST資料集是由60,000筆訓練資料、10,000筆測試資料所組成。
- MNIST資料集裡的每一筆資料皆由images(數字的影像)與labels(該圖片的真實數字，其實就是答案)所組成



https://en.wikipedia.org/wiki/MNIST_database

Keras MNIST 手寫數字資料集

- Training data : (60000,28,28)
- Training label: (60000,)
- Test data : (10000, 28,28)
- Test label: (10000,)



資料預處理

- 建立模型必須先將內容進行資料預處理
- 資料預處理分為兩個部分

  ① Features(數字影像特徵值)資料預處理.　28*28數字影像 ➔ 784 , type:float

  ② Labels(數字影像真實值)資料預處理.　數字影像Image的數字影像標準化

```
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
  18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
 253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
 253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253,
 253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253,
 205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,
  90,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
```

```
 0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   3.,
18.,  18.,  18., 126., 136., 175.,  26., 166., 255.,
247., 127.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
  0.,   0.,   0.,   0.,   0.,  30.,  36.,  94., 154.,
170., 253., 253., 253., 253., 253., 225., 172., 253.,
242., 195.,  64.,   0.,   0.,   0.,   0.,   0.,   0.,
  0.,   0.,   0.,   0.,   0.,  49., 238., 253., 253.,
253., 253., 253., 253., 253., 253., 251.,  93.,  82.,
 82.,  56.,  39.,   0.,   0.,   0.,   0.,   0.,   0.,
  0.,   0.,   0.,   0.,   0.,   0.,  18., 219., 253.,
253., 253., 253., 253., 198., 182., 247., 241.,   0.,
  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,  80.,
156., 107., 253., 253., 205.,  11.,   0.,  43., 154.,
  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
  0.,  14.,   1., 154., 253.,  90.,   0.,   0.,   0.,
  0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
```

```
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.01176471, 0.07058824, 0.07058824,
0.07058824, 0.49411765, 0.53333336, 0.68627453, 0.10196079,
0.65098041, 1.        , 0.96862745, 0.49803922, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.11764706, 0.14117648, 0.36862746, 0.60392159,
0.66666669, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
0.99215686, 0.88235295, 0.67450982, 0.99215686, 0.94901961,
0.7647059 , 0.25098041, 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.19215687, 0.93333334,
0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.99215686,
0.99215686, 0.99215686, 0.99215686, 0.98431373, 0.36470589,
0.32156864, 0.32156864, 0.21960784, 0.15294118, 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
0.        , 0.        , 0.        , 0.        , 0.        ,
```

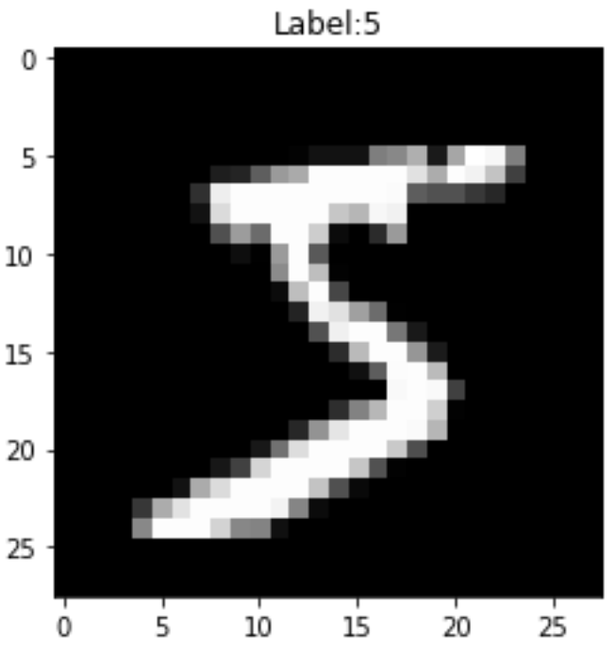
- labels資料預處理

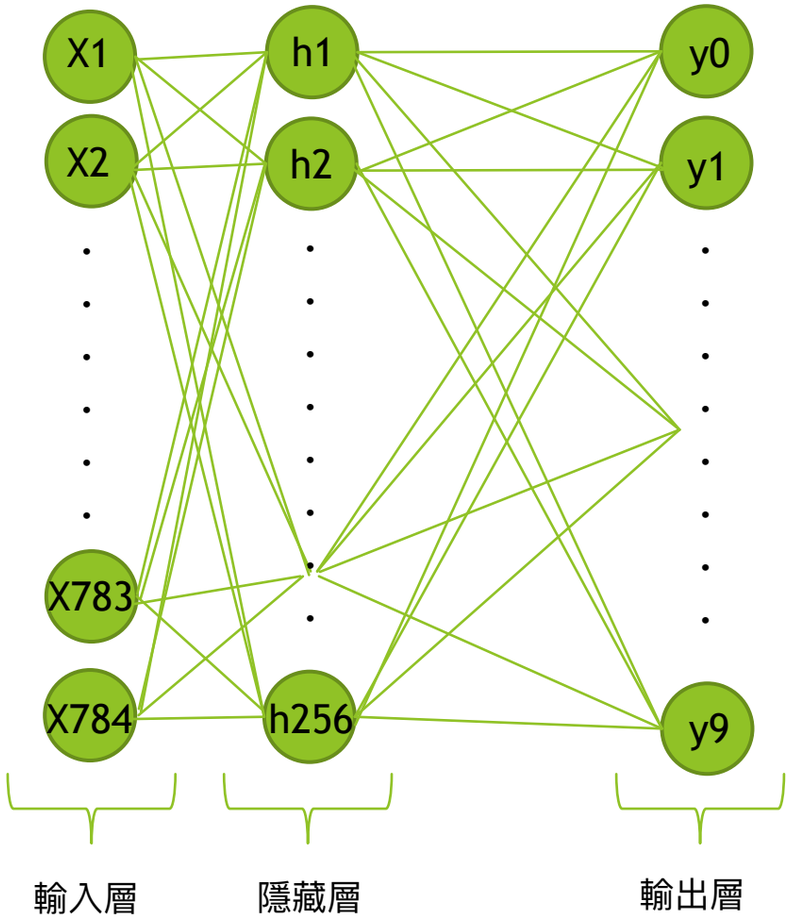
label原本是0~9的數字 → 10個0或1的組合

數字3經過 one-hot encoding ➔ 0001000000

數字7經過 one-hot encoding ➔ 0000000100

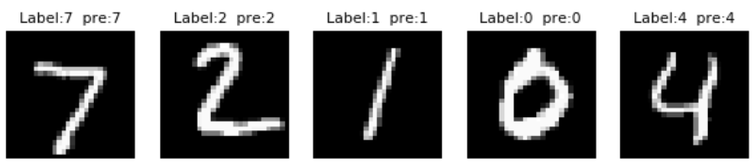
多層感知器模型(MLP)介紹



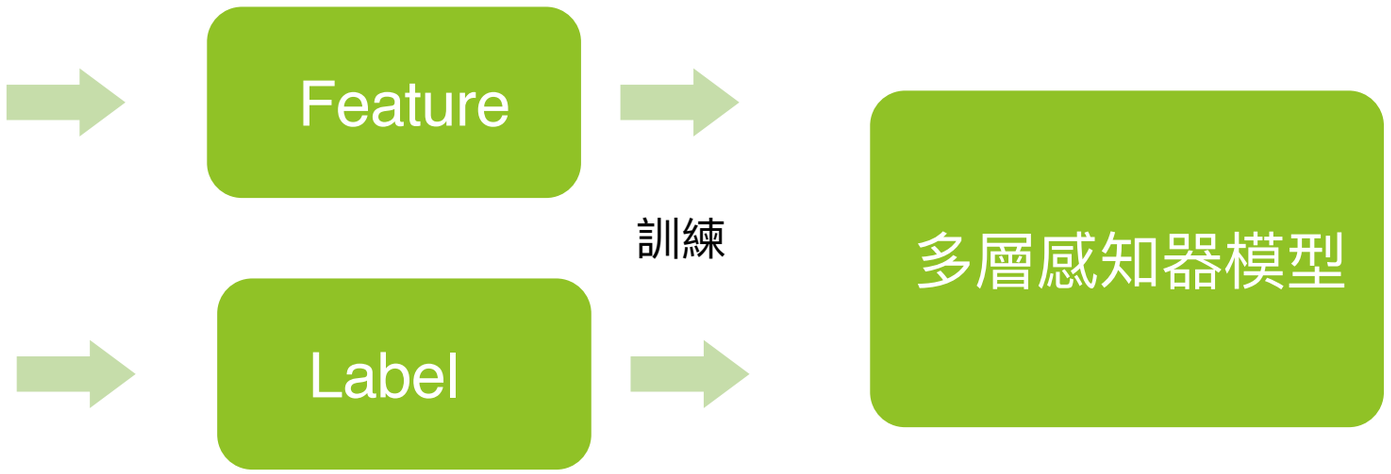


多層感知器(MLP)訓練與預測

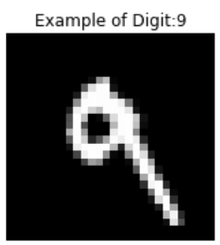
- 訓練 Training



7 2 1 0 4



- 預測 Predict



MLP 建立步驟

1.資料預處理

2.建立模型

3.訓練模型

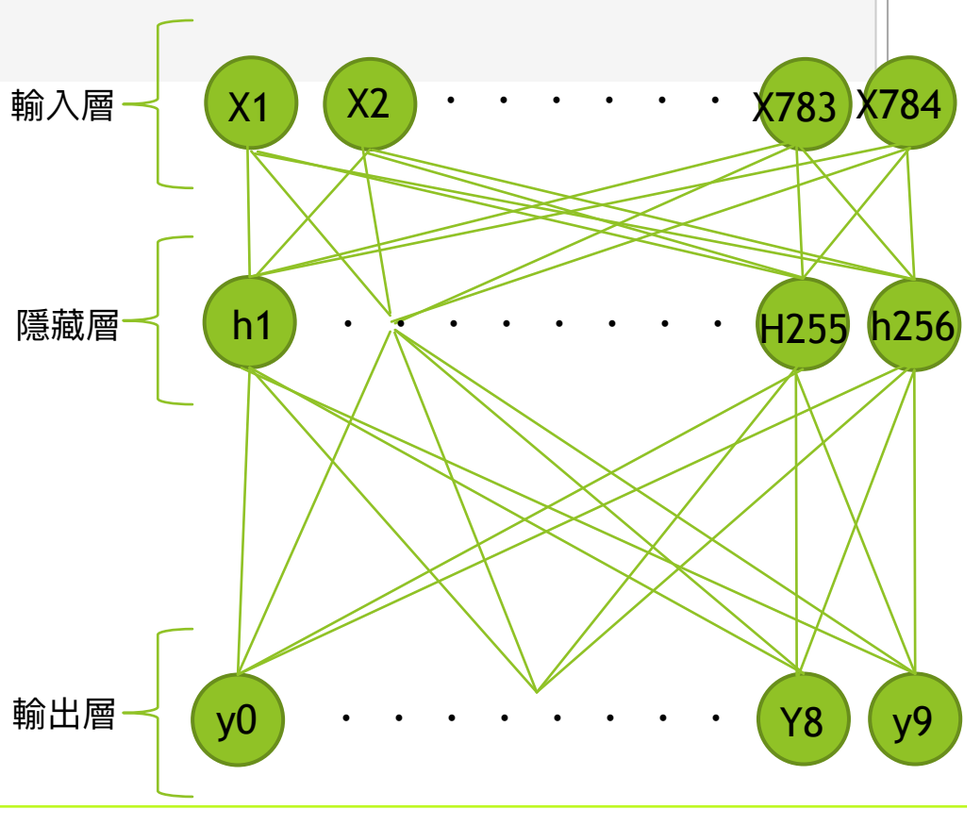
4.評估模型準確率

5.進行預測

建立模型

```
# 建立 Sequential 順序模組
model = Sequential()


# 模型加入【輸入層】與【隱藏層】
model.add( Dense( input_dim = 28*28              # 輸入層有 28*28=784 個神經元
                , units = 256                    # 隱藏層有 256 個神經元    (值越大, 訓練越精準, 相對訓練時間也越久)
                , kernel_initializer = 'normal' # 使用 normal 初始化 weight 權重與 bias 偏差值
                , activation = 'relu'            # 使用 relu 激活函數
                ))

# 模型加入【輸出層】
model.add( Dense( units = 10                     # 輸出層有 10 個神經元 (因為數字只有 0 ~ 9)
                , kernel_initializer = 'normal' # 使用 normal 初始化 weight 權重與 bias 偏差值
                , activation = 'softmax'         # 使用 softmax 激活函數 (softmax 值越高, 代表機率越大)
                ))

# 顯示模型摘要資訊
print (model.summary() )
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               200960
_________________________________________________________________
dense_1 (Dense)              (None, 10)                2570
=================================================================
Total params: 203,530
Trainable params: 203,530
Non-trainable params: 0
_________________________________________________________________
None
```



設定模型的訓練方式

```
## 設定模型的訓練方式 ##

model.compile( loss='categorical_crossentropy' # 設定 Loss 損失函數 為 categorical_crossentropy 使用交叉熵
             , optimizer = 'adam'              # 設定 Optimizer 最佳化方法 為 adam
             , metrics = ['accuracy']          # 設定 Model 評估準確率方法 為 accuracy
             )


# 訓練模型
history = model.fit(                    # 訓練的歷史記錄, 會會回傳到指定變數 history
          x = X_train_normalize         # 設定 圖片 Features 特徵值 (mnist 提供 60000 筆資料)
        , y = Y_train_OneHot            # 設定 圖片 Label    真實值 (mnist 提供 60000 筆資料)
        , validation_split = 0.2        # 設定 有多少筆驗證          (60000*0.2=12000 筆驗證, 60000*0.8=48000 筆訓練)
        , epochs = 10                   # 設定 訓練次數               (值 10 以上,  值越大, 訓練時間越久, 但訓練越精準)
        , batch_size = 128              # 設定 訓練時每批次有多少筆 (值 100 以上, 值越大, 訓練速度越快, 但需記憶體要夠大)
        , verbose = 2                   # 是否 顯示訓練過程          (0: 不顯示, 1: 詳細顯示, 2: 簡易顯示)
)
```

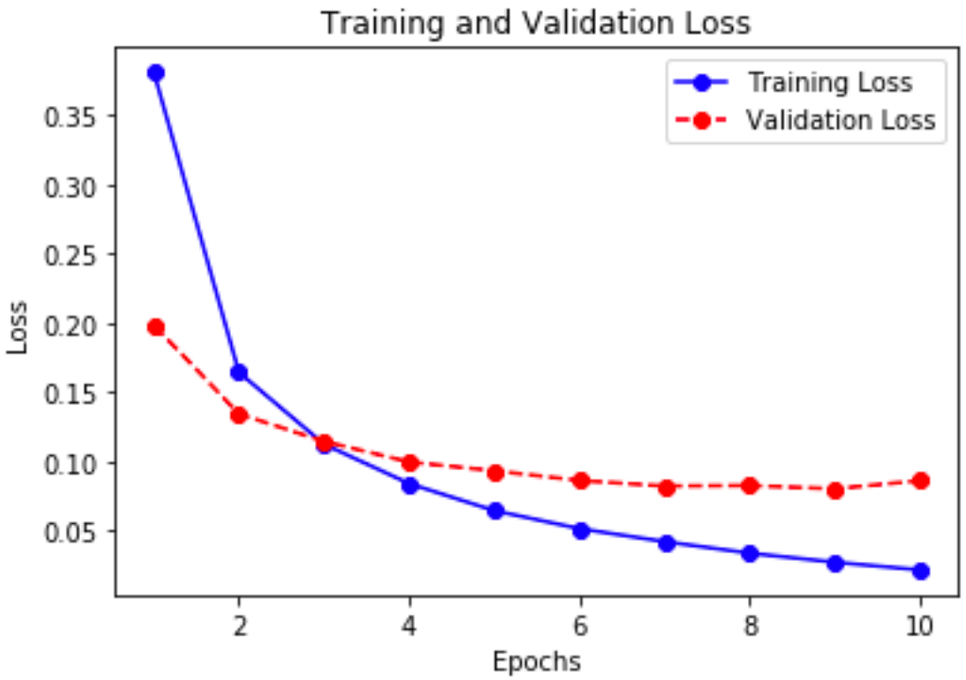
- 設定訓練與驗證比例
- 設定epochs次數與每一批次筆數

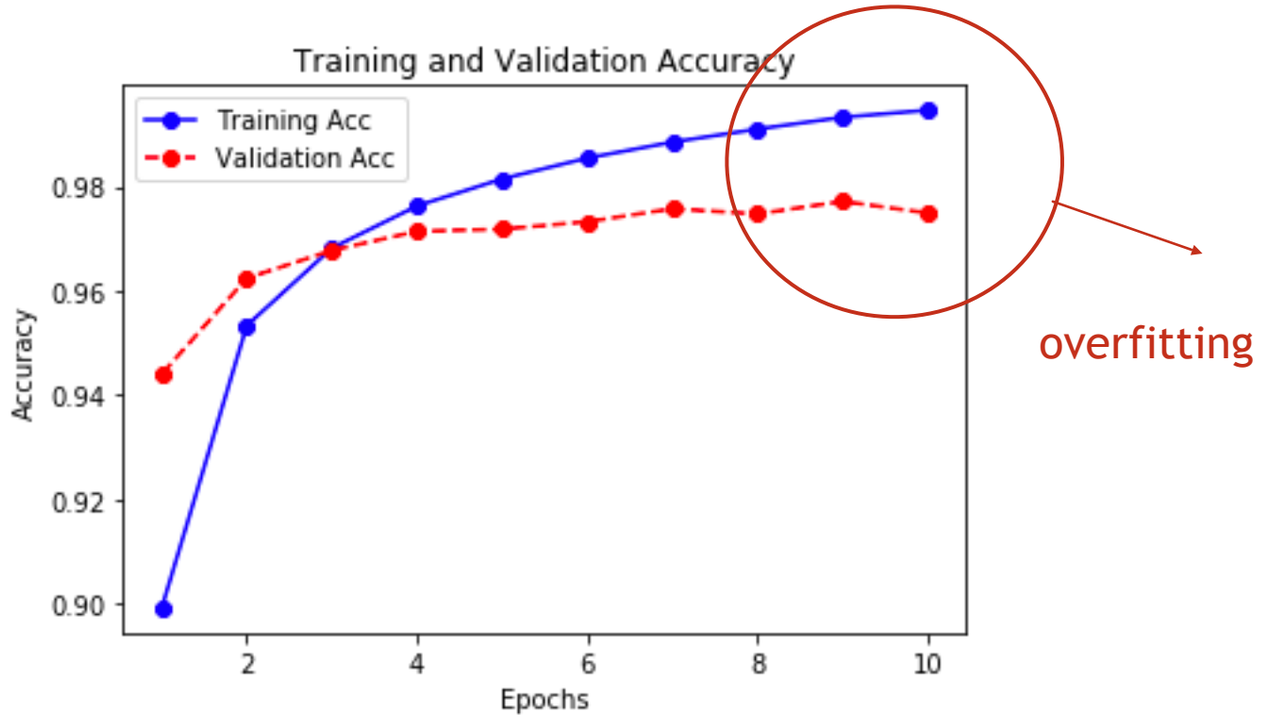
訓練資料

```
# 執行的顯示結果 (這會花一些時間, 然後會逐次顯示訓練結果)
# loss:     使用訓練資料, 得到的損失函數誤差值 (值越小, 代表準確率越高)
# acc:      使用訓練資料, 得到的評估準確率     (值在 0~1, 值越大, 代表準確率越高)
# val_loss: 使用驗證資料, 得到的損失函數誤差值 (值越小, 代表準確率越高)
# val_acc:  使用驗證資料, 得到的評估準確率     (值在 0~1, 值越大, 代表準確率越高)


# 評估模型
print("\nTesting ...")
loss, accuracy = model.evaluate(X_train_normalize , Y_train_OneHot)
print("訓練資料集的準確度 = {:.2f}".format(accuracy))
loss, accuracy = model.evaluate(X_test_normalize , Y_test_OneHot)
print("測試資料集的準確度 = {:.2f}".format(accuracy))

model.save('mnist_mlp.h5')
```

結果

- 多層感知器
- 輸入層:784個神經元,隱藏層:256個神經元,輸出層:10個神經元
- Epoch=10





混淆矩陣 Confusion matrix

- 混淆矩陣(Confusion matrix)是一種對分類模型進行效果評估的方法
- 通過將模型預測的數據與測試數據進行對比，使用準確率，覆蓋率和命中率等指標對模型的分類效果進行度量。

```
import pandas as pd

# 計算分類的預測值
print("\nPredicting ...")
Y_pred = model.predict_classes(X_test_normalize)

# 顯示混淆矩陣
tb = pd.crosstab(Y_test_bk.astype(int), Y_pred.astype(int),
                 rownames=["label"], colnames=["predict"])
print(tb)
```

```
Predicting ...
predict    0     1     2    3    4    5    6     7    8    9
label
0        966     0     1    1    1    3    4     2    1    1
1          0  1124     4    1    0    1    2     0    3    0
2          4     1  1008    1    2    0    2     7    7    0
3          1     1     4  988    0    5    0     5    3    3
4          1     1     2    1  958    0    3     3    0   13
5          2     0     0    7    1  866    6     2    5    3
6          6     2     0    1    4    4  941     0    0    0
7          1     4    10    3    0    0    0  1002    0    8
8          2     0     2    5    6    2    3     3  948    3
9          3     2     0    7    5    1    0     5    0  986
```

Overfitting問題

- Overfitting(過度訓練):
  當可選擇的參數自由度超過資料所包含的資訊內容時，
  這會破壞模型一般化的能力。

- 解決方法:

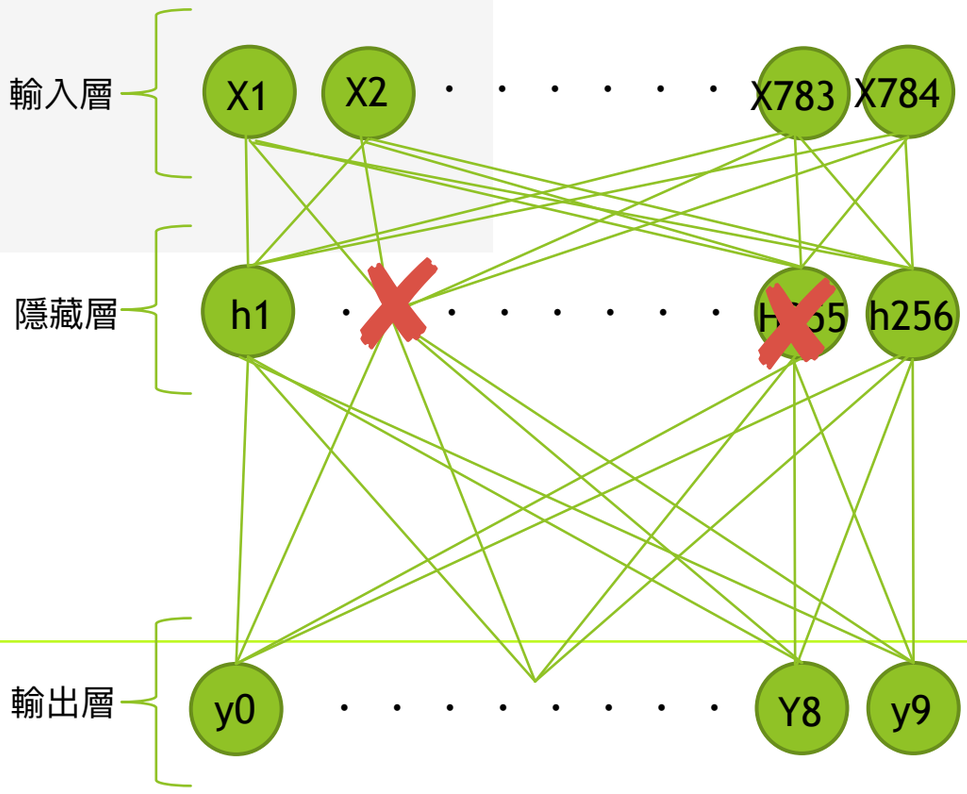
  ① 增加數據量, 大部分過擬合產生的原因是因為數據量太少了。

  ② 加入DropOut功能，在訓練的時候，我們隨機忽略掉一些神經元和神經元的連結，讓每一次預測結果都不會太過依賴於其中某部分特定神經元。

多層感知器加入DropOut功能

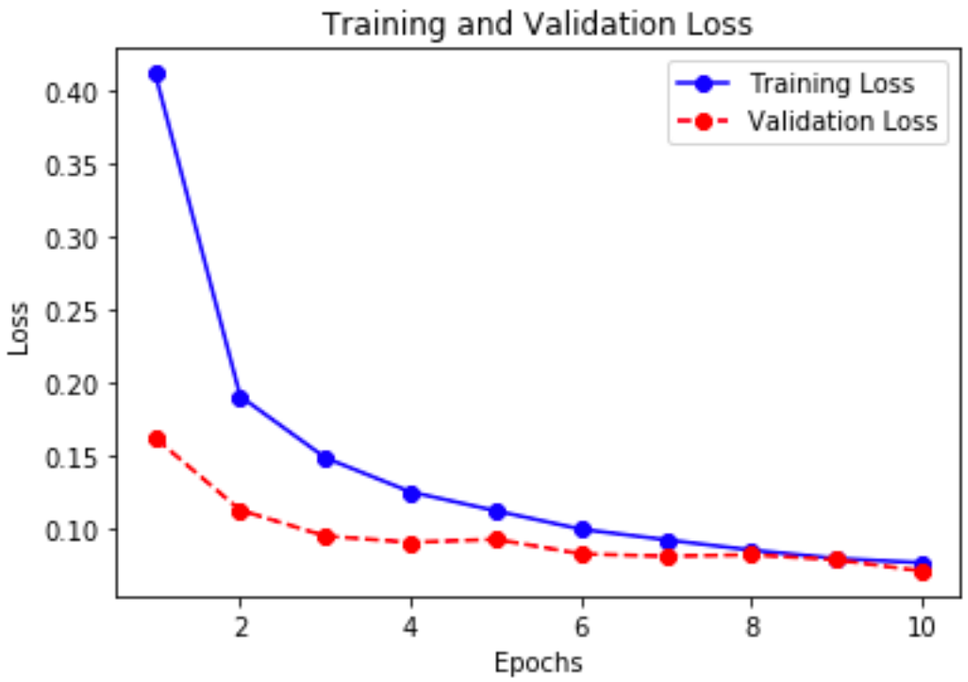
- 增加DropOut功能:為了解決Overfitting的問題
- DropOut的功能:每次訓練迭代時,隨機地在隱藏層中放棄神經元,以避免overfitting
- DropOut(0.5)➔放棄隱藏層中50%的神經元
  DropOut(0.25)➔放棄隱藏層中25%的神經元

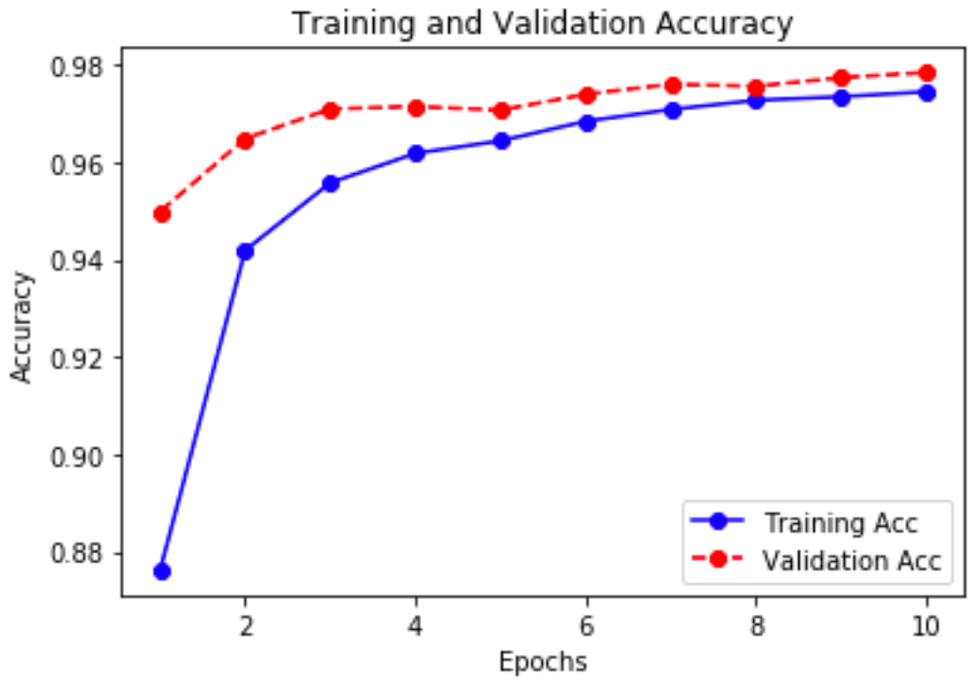
```
# 定義模型
model = Sequential()
model.add(Dense(256, input_dim=28*28, activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(256, activation="relu"))
model.add(Dense(10, activation="softmax"))
model.summary()    # 顯示模型摘要資訊
```



結果

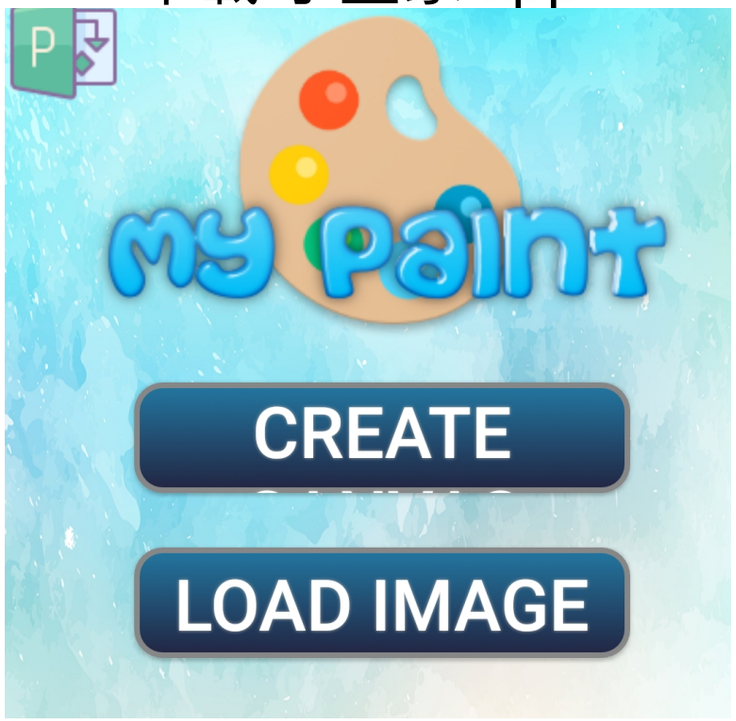
- 多層感知器
- 輸入層:784個神經元,隱藏層:800個神經元,輸出層:10個神經元
- Epoch=10
- 加入DropOut(0.5)

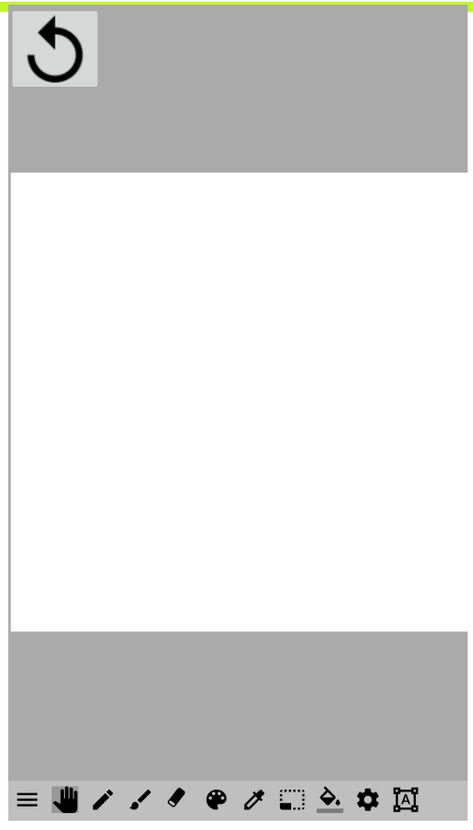


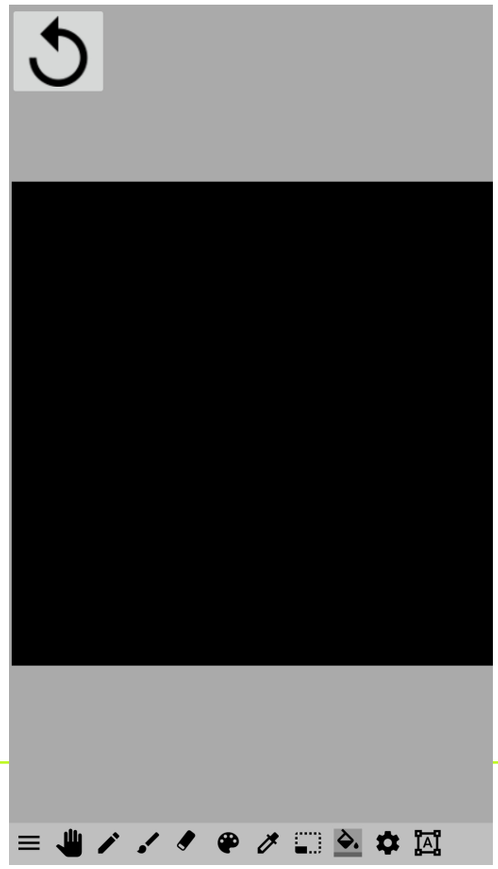


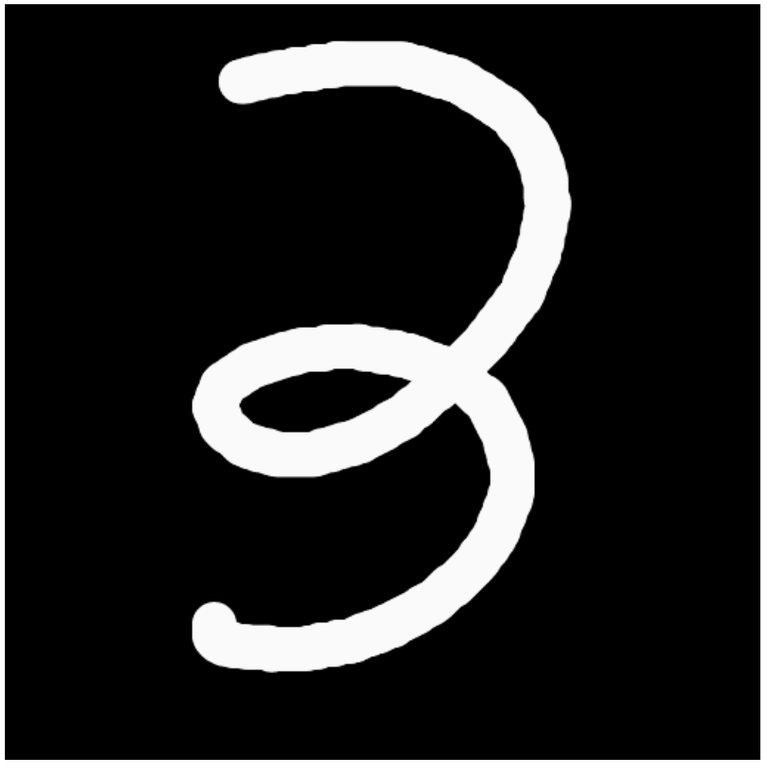
應用： 使用自己的模型 進行預測

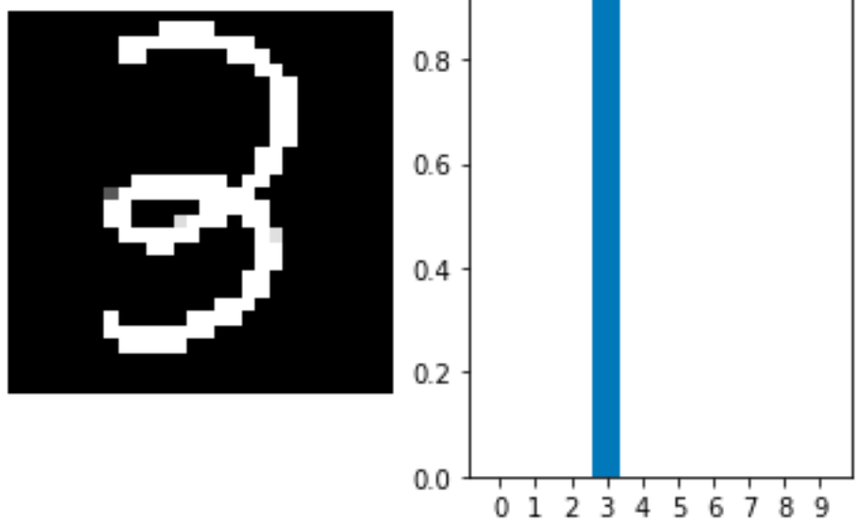
下載 小畫家 app

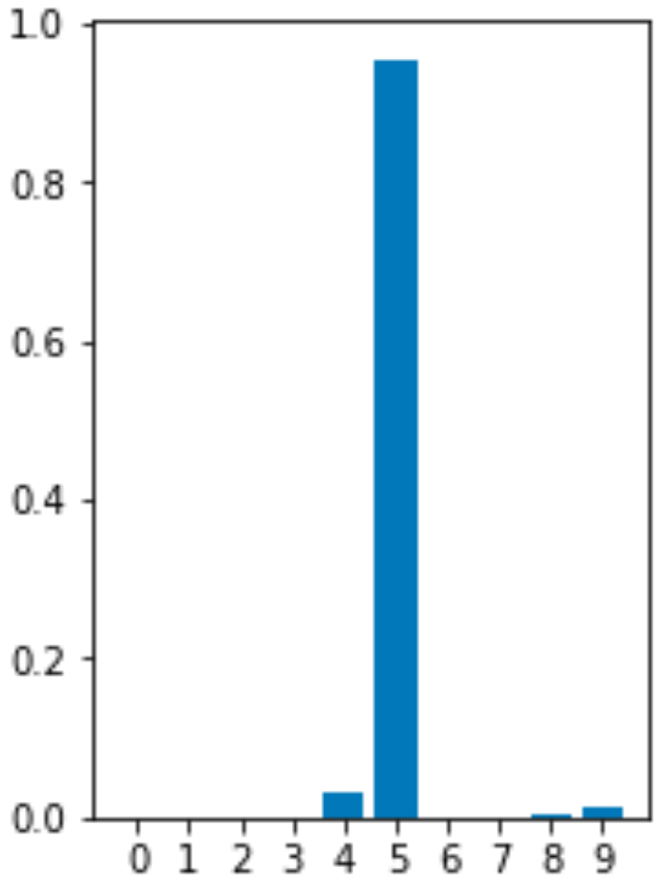









1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9


1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9