A. 準確率：

```
[ ]  1 accuracy = (np.count_nonzero(df["real"]==df["predict"])) / len(data_path)
     2 print(accuracy)
     3 print("accuracy = ", format(accuracy * 100, "0.2f"), sep = "")
```

```
0.9835294117647059
accuracy = 98.35
```

此處我們是將真正的數字，以及要預測的數字給放在一個 cvs 檔中，左右比較看是否正確，而最後得到的結果是 98.35%。

B. Source code 解釋：

MINST_handwrite.py:

1. 載入我們的套件，有包含會快速建構模型的 keras, 以及 cv2…等在，在 preprocessing 和 model 中會使用到的 modules.

```
 1 from keras.models import Model
 2 from keras.layers import Input, Dense, Dropout, Flatten, Conv2D, MaxPooling2D, BatchNormalization
 3 from keras.callbacks import ModelCheckpoint, EarlyStopping, TensorBoard
 4 from PIL import Image, ImageEnhance
 5 import numpy as np
 6 import os
 7 import csv
 8 import cv2
 9 import time
10 import random
11 from matplotlib import pyplot as plt
12
```

2. 此處是我們 model 的架構，利用 Con2D 建構出整個架構，並且設定 filters 數量以及其他參數，讓我們的模型可以更加的提取出照片的特徵。

```
 1 # Create CNN Model
 2 print("Creating CNN model...")
 3 input_data = Input((28, 28, 1))
 4 out = input_data
 5 out = Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu')(out)
 6 out = Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu')(out)
 7 out = Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu')(out)
 8 out = Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu')(out)
 9 out = Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu')(out)
10 out = Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), padding='same', activation='relu')(out)
11 out = BatchNormalization()(out)
12 out = MaxPooling2D(pool_size=(2, 2))(out)
13 out = Dropout(0.2)(out)
14 out = Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu')(out)
15 out = Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu')(out)
16 out = Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu')(out)
17 out = Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu')(out)
18 out = Conv2D(filters=64, kernel_size=(3, 3), activation='relu')(out)
19 out = BatchNormalization()(out)
20 out = MaxPooling2D(pool_size=(2, 2))(out)
21 out = Flatten()(out)
22 out = Dropout(0.25)(out)
23 out = [Dense(10, name='digit', activation='softmax')(out)]
24 model = Model(inputs=input_data, outputs=out)
25 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
26 model.summary()
```

3. 在此處是我們將 原先的 label 是一個 0~9 的數字，改成如：[0,0,0,0,0,0,0,0,0,1]的感覺，我們創建一個 10 個 zero 的 array，然後去 LETTERSTR 中尋找是對應哪一個 Label，並且在那個位置從 0 -> 1.

```
LETTERSTR = '0123456789'
def toonehot(text):
    labellist = []
    for letter in text:
        onehot = [0 for _ in range(10)]
        num = LETTERSTR.find(letter)
        onehot[num] = 1
        labellist.append(onehot)
    return labellist[0]

```

4. 第一行為 train_img 的所在根路徑，下面的兩個 for 迴圈，則是將所有 directories 下的圖片檔案全部給搜尋出來，最後會將每個 directories 的完整路徑存在 train_path 當中，最後因為我們在訓練時應該需要打亂前後順序，以避免偏重某一種資料特性，因此我們用 random.shuffle 打亂。

```
train_root = '/content/drive/MyDrive/Algorithm/Midterm/train_image'
train_path = []
for file in os.listdir(train_root):
  tmp_path = os.path.join(train_root, file)
  for d in os.listdir(tmp_path):
    train_path.append(os.path.join(tmp_path, d))
random.seed(20)
random.shuffle(train_path)
train_path[0:5]
```

5. Data_path 我們利用 os.listdir 迴圈去將所有檔案給撈出來，並把所有 files 的路徑存在 data_path 當中，train_label 則會同時跟著看是在哪一種 label 的檔案下，順便同順序的 label 起來。

```
data_path = []
train_label = []
for i in range(len(train_path)):
  for root, dirs, files in os.walk(train_path[i]):
    for n in range(len(files)):
      data_path.append(os.path.join(root, files[n]))
      train_label.append(toonehot(root[-1]))
print('checkpoint: ', train_label[0:5])
print('data length: ', len(data_path))
print('The shape of labels: ', np.shape(train_label))
```

6. 因為我們 file 名稱中有中文名稱，用 cv2.imread()將會出現錯誤，因此我們自己寫一個 function，是利用 cv2.imdecode()，並且 decode 完畢以後，將圖片轉成灰階，也因此會看到從三維變成了二維的 shape.

```
def cv_imread(filePath):
    cv_img=cv2.imdecode(np.fromfile(filePath,dtype=np.uint8), cv2.IMREAD_COLOR)
    img_gray = cv2.cvtColor(cv_img, cv2.COLOR_RGB2GRAY)
    return img_gray
```

7. 被註解掉的綠色字，是我們將所有資料用 list 儲存起來，並且轉乘 array 形式存取，讓我們能在下一次讀取時直接抓進來，不用重新一次。將存取的東西 load 到 train_matrix 當中，並且因為我們 data 裡面的資料仍然維 RGB 的色碼，因此我們 /255.0 去轉換，並且輸出。

```
#img = [cv_imread(data_path[i]) for i in range(len(data_path))]
#np.save('train_img', img)
```

```
train_matrix = np.load('/content/drive/MyDrive/Algorithm/Midterm/train_img.npy')
train_matrix.shape
```

```
(2450, 28, 28)
```

```
train_data = np.stack([np.array(i)/255.0 for i in train_matrix])
print("The dimension of training data: ", train_data.shape)
```

```
The dimension of training data:  (2450, 28, 28)
```

8. 這裡將剛剛的模型抓回來，開始丟入資料做訓練，設定的 epochs=30, batch_size=32，並且將整個過程利用 history 記錄下來，以方便我們做畫圖使用。

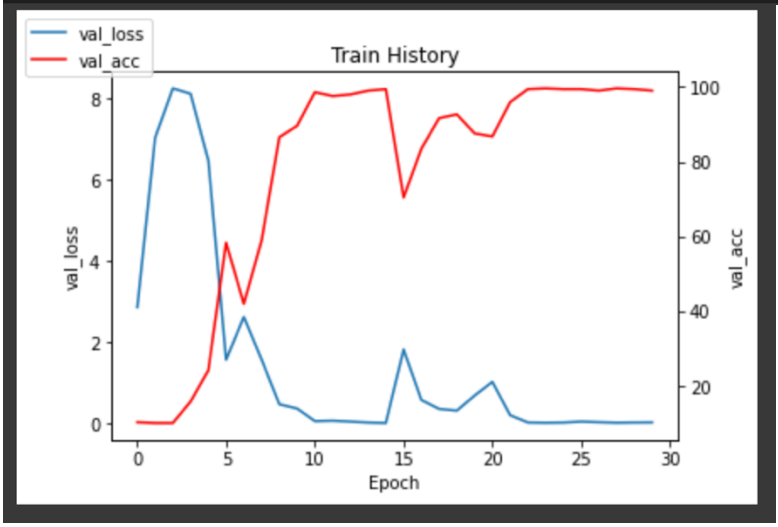
```
my_tensorboard = TensorBoard(log_dir = ".\\log", histogram_freq = 1)
my_callback = my_tensorboard
training_history = model.fit(train_data, train_label, epochs=30, verbose = 1 ,
                             batch_size=32, validation_split=0.2, callbacks=my_callback)
```

9. 將我們整個訓練的過程畫出來，並且標上 x 軸以及 y 軸，可以看到下圖我們的 loss 逐漸下降，並且 validation 的準確度逐漸上升。

```
x = np.arange(0, 30, 1)
y1 = training_history.history['val_loss']
y2 = np.array(training_history.history['val_accuracy']) * 100
fig = plt.figure()
ax1 = fig.add_subplot(111)
ax1.plot(x, y1)
ax1.set_ylabel("val_loss")
ax1.set_xlabel("Epoch")

ax2 = ax1.twinx()
ax2.plot(x, y2, "r")
ax2.set_ylabel("val_acc")

plt.title("Train History")
plt.xlabel("Epoch")
fig.legend(["val_loss", "val_acc"], loc = "upper left")
plt.show()
```



MINST_predict.py:

1. 一樣載入我們會需要用到的 modules。

```
from keras.models import load_model
from keras.models import Model
from keras import backend as K
from PIL import Image
import numpy as np
import os
import csv
import cv2
import time
from matplotlib import pyplot as plt
import pandas as pd
import random
```

2. 先設定我們的 global variable，等等方便做使用，不會太長。LETTERSTR 會是在預測時所尋找最大的機率並取出的那個數值位置，也就成為我們的預測數字。
   Model_path 就是載入我們剛剛訓練模型的位置
   Test_root 也就是我們的 test img 根路徑

```
LETTERSTR = '0123456789'
model_path = '/content/drive/MyDrive/Algorithm/Midterm/Algo_midterm1619459079.h5'
test_root = '/content/drive/MyDrive/Algorithm/Midterm/test_image'
```

3. 先設定一個空的 dataframe，我們是用來儲存預測結果以及實際真實結果的 table，我們可以直接比較，來計算出 accuracy 為多少。

```
df = pd.DataFrame(columns=[None, None, None])
df.columns = ['img_path', 'real', 'predict']
```

4. Test_path 將會儲存所有 directories 的路徑，是利用 os.listdir 來迴圈搜尋，最後存進來裡面，並且一樣不要因為有固定的順序，因此我們用 random.shuffle 打亂資料順序。

```
test_path = []
for file in os.listdir(test_root):
  tmp_path = os.path.join(test_root, file)
  for d in os.listdir(tmp_path):
    test_path.append(os.path.join(tmp_path, d))
random.seed(20)
random.shuffle(test_path)
test_path[0:5]
```

5. 將全部 files 的路徑儲存在 data_path 當中，並且在 loop 的同時，一起標注 test_label 上去，不會有 label 錯誤的情況發生。

```
data_path = []
test_label = []
for i in range(len(test_path)):
  for root, dirs, files in os.walk(test_path[i]):
    for n in range(len(files)):
      data_path.append(os.path.join(root, files[n]))
      test_label.append(root[-1])

print('checkpoint: ', test_label[0:5])
print('length of directories: ', len(test_path))
print('The shape of labels: ', np.shape(test_label))
```

```
checkpoint:  ['9', '9', '9', '9', '9']
length of directories:  340
The shape of labels:  (1700,)
```

6. 將我們 dataframe 中的欄位填上資料，img_path 就是我們的檔案名稱，而剛剛 test_label 則是實際數字數值的 value，存在 real 這個 column 當中。

```
df['img_path'] = data_path
df['real'] = test_label
```

7. 這裏將我們的圖片一樣做處理，定了一個 function 來處理可能有中文名稱路徑的情況，並且轉灰階處理，註解掉的地方將這些 array 給儲存下來成為.npy 檔案方便讀取，最後讀取他們/255.0 來成為只有 0,1 的資料檔案

```
def cv_imread(filePath):
    cv_img=cv2.imdecode(np.fromfile(filePath,dtype=np.uint8), cv2.IMREAD_COLOR)
    img_gray = cv2.cvtColor(cv_img, cv2.COLOR_RGB2GRAY)
    return img_gray
```

```
#img = [cv_imread(data_path[i]) for i in range(len(data_path))]
#np.save('test_img', img)
```

```
test_matrix = np.load('/content/drive/MyDrive/Algorithm/Midterm/test_img.npy')
print(test_matrix.shape)
test_data = np.stack([np.array(i)/255.0 for i in test_matrix])
test_data.shape
```

```
(1700, 28, 28)
(1700, 28, 28)
```

8. K.clear_session()是將我們階段東西清乾淨，以避免有存在的數值或是架構在裡面，導致預測錯誤，接下來將 model 給載入，並且把 test_data 丟給 model 預測。

```
print('--------Predicting---------')
K.clear_session()
model = None
model = load_model(model_path)
prediction = model.predict(test_data)
```

```
--------Predicting---------
```

9. 我們預測的結果會出現在 prediction 當中，但是他出現的會是 n 比資料且 10 個數值，這些會是每個數字的機率，我們利用 np.argmax 拿出最大機率的位置，並且利用 LETTERSTR 去尋找他是哪個數字，並存在 ans 當中，最後統一 append 進入 predict_num。
   我們將 predict_num 載入 csv 中的 predict 欄位，並且將 csv 儲存下來。

```
predict_num = []
for i in range(len(data_path)):
  ans = LETTERSTR[np.argmax(prediction[i])]
  predict_num.append(ans)
df['predict'] = predict_num
df.to_csv(os.getcwd()+'testing.csv')
print('Testing dataframe is saved in: ', os.getcwd())
```

```
Testing dataframe is saved in:  /content
```

10. 最後將 dataframe 當中 real, 以及 predict 欄位去做比較，看是否有不一樣的情況發生，並且去計算 accuracy，最後得到結果 98.35%的辨識準確率。

```
accuracy = (np.count_nonzero(df["real"]==df["predict"])) / len(data_path)
print(accuracy)
print("accuracy = ", format(accuracy * 100, "0.2f"), sep = "")
```

```
0.9835294117647059
accuracy = 98.35
```