MACHINE LEARNING

Usps&Mnist Domain adaption

使用 CNN 建模

組員：**H24041105** 潘縉緯

H24044014 廖毅桓

H24041147 廖怡華

H24041244 周秋全

目錄

目標

如今我們有兩筆手寫數字資料，分別為 MNIST Handwritten Digits 以及 USPS Handwritten Digits，資料集皆為數字 0 到 9，而更詳盡的資料介紹將會在稍後為大家說明。我們想要透過一個分類器以達到完成辨識擁有相同目標的兩個資料集—也就是分辨出兩筆資料中 0 到 9 的影像。我們大可一筆資料創建一個分類器，各自進行辨識，相信效果一定會比只有一個分類器同時使用於兩組資料集好上太多，畢竟兩組資料集來自不同的公司，影像所擁有的特徵一定會有些許的差異，如此一來辨識的準確率或許效果就不會太好。縱使如此，我們仍舊想嘗試只用一個分類器，完成這看似不可能的任務，也期望可以擁有一定程度的準確率。

資料簡介

【MNIST Handwritten Digits】

MNIST 是一個非常有名的手寫數字資料集，他的重要性就如同學習如何打程式時入門有 Hello World，而在學習機器學習入門有 MNIST。這筆資料是由卷積神經網路之父 Yann LeCun 所蒐集，他目前是臉書 AI 研究院院長。

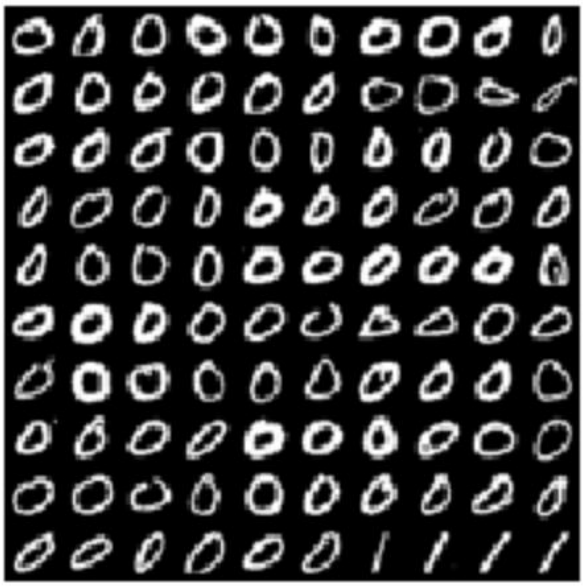
在 MNIST 原始資料中有 60000 個訓練樣本集和 10000 個測試樣本集，在 MNIST 資料集中的每一張圖片都代表了 0 到 9 中的一個數字。圖片的大小皆為 28×28，且數字都會出現在圖片的正中間。

而 MNIST 資料集是由美國國家標準與技術研究院（NIST）的兩個資料集—

Special Database 1 和 Special Database 3 結合得到。Special Database 1 是美國高中生的手寫數字，而 Special Database 3 是美國人口調查局的員工所寫的手寫數字。

這次期末報告中，我們使用的 MNIST Handwritten Digits 只有 2000 筆，每筆大小皆為 16×16，視覺化後可以看到以下結果：

```
img=data['X_src'][0:256,100:200]
montage_img=np.zeros([100,16,16])
for i in range(100):
    montage_img[i]=img[0:256,i].reshape(16,16)
plt.imshow(montage2d(montage_img),cmap='gray')
plt.axis('off')
plt.show()
```



【USPS Handwritten Digits】

USPS 資料集總共有 1600 筆資料，是從美國郵政署掃描信封上的數字所得到的手寫數字，一開始是用二進位表示，每張大小、方向也都不同。然而對他們的大小做標準化，全部變為大小同為 16x16 的影像，一樣為灰階

```
img=test['X_src'][0:256,100:200]
montage_img=np.zeros([100,16,16])
for i in range(100):
    montage_img[i]=img[0:256,i].reshape(16,16)
plt.imshow(montage2d(montage_img),cmap='gray')
plt.axis('off')
plt.show()
```

【兩筆資料綜合比較】

相同

兩筆資料經過標準化後都呈現大小為 16x16 的影像，且都呈灰階之樣貌，相信對之後創建分類器進行辨識是一大好事。

相異

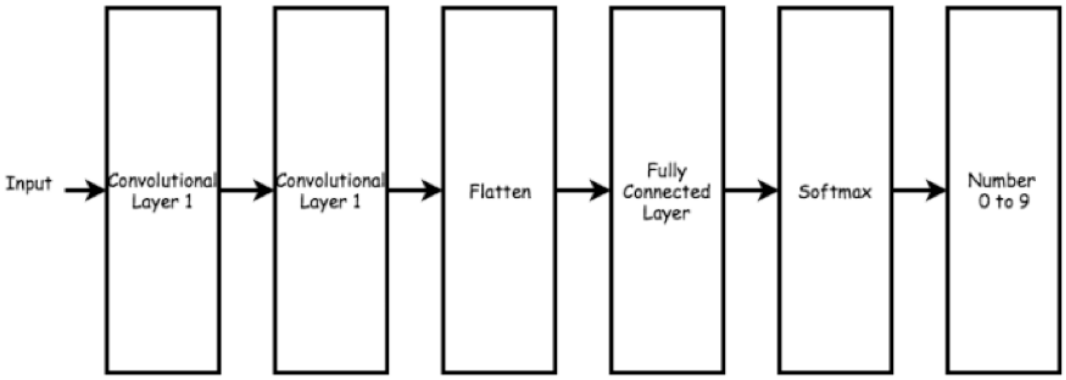
由視覺化後的展現不難發現 MNIST Handwritten Digits 大多集中在中央，不像 USPS Handwritten Digits 會延展到圖片的四邊。簡單來說，MNIST 剩餘的黑色區域相對於 USPS 而言多上許多，如此一來在兩邊取得的特徵就會有所差異。

CNN 介紹

當 CNN 分辨一張新圖片時，在不知道上述特徵　在哪的情況下，CNN 會比對圖片中的任何地方。為了計算整張圖片裡有多少相符的特徵，我們在這裡創造了

一套篩選機制。這套機制背後的數學原理被稱為卷積(convolution)，也就是 CNN 的名稱由來。

CNN 流程



圖片輸入會經過兩個卷積層（convolutional layer）然後把它扳平（Flatten）之後進入全連結層（Fully Connected Layer）最後就是進入 Softmax 分類成 10 個數字。

卷積層 Convolutional Layer

卷積層中有三個部分，我們會邊用程式邊解說:

- Convolution
- ReLU
- Max Pooling

第一層卷積為例

以下是建立 `CONVOLUTION` 的五行程式碼:

```
# first convolutinal layer
W_conv1 = weight_variable([8, 8, 1, 32])
b_conv1 = bias_variable([32])
x_image = tf.reshape(x, [-1, 16, 16, 1])
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
```

我們先從第三行開始看，它的用意是把圖片像素輸入變成一個 1x16x16x1 的四維矩陣。
x_image:

- 第一個參數為 -1 的意思是不管輸入有幾個，Tensorflow 會自動調整的意思，假設輸入 50 個圖片第一個維度就是 50。
- 第二個第三個就是指數字圖片為 (16*16)矩陣 的意思。
- 最後一個則是指色階，這裡又稱 CHANNEL，這裡因為我們的輸入都是黑白灰階，因此值為 1，如果是彩色圖有 RGB 三原色，這裡的數值就是 3。

第一行則是建立起過濾器 (filter)，過濾器我覺得也可以稱作特徵篩選器，可以想像說是負責來辨認圖片中的某些特徵，像是直線或是橫線或是轉彎處。
W_conv1:

- 一二三維的數字 [8, 8, 1] 代表著這個過濾器是一個 8x8x1 的矩陣，這跟上面的圖片一樣 8x8 代表著 (8*8)矩陣，而 1 則代表著灰階。
- 最後一個數字 32 代表著我們建立了 32 個過濾器來篩選特徵。

定義卷積運算函數

Code:

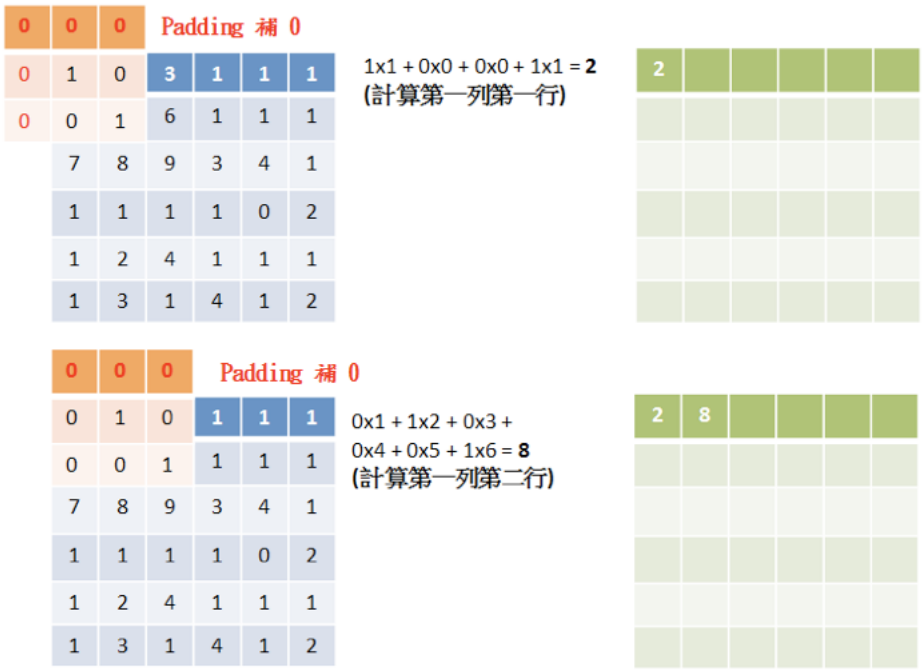
```
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding = 'SAME')
```

stride 的意義:

- 第一個為 1 代表所有樣本都會使用到。
- 中間兩個是 x 軸和 Y 軸的移動距離，邊界使用補零（zero padded）的方式處理。所以輸入與輸出的資料大小是相同的。
- 最後一個 1 也是色階。

padding 為 SAME 的意思是就是要讓輸入和輸出的大小是一樣的。

以下為運算模式:



經過卷積運算後得到的數值越大就越接近過濾器的特徵。

定義 RELU 激活函數

將卷積層的輸出結果通過一個函數如下:

x < 0 的時候全部為 0，而 x > 0 時則為 x。

激活函數的用意是增快電腦的運算速度。

定義池化函數

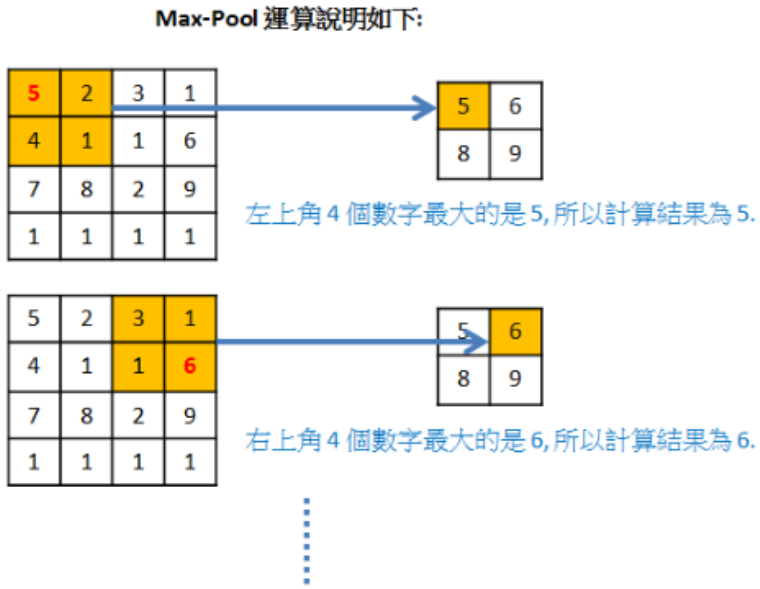
Code:

```
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding = 'SAME')
```

K size 的意義:

池化窗口的大小，取一个四维向量，一般是[1, height, width, 1]，因为我们不想在 batch(資料數量)和 channels(色階)上做池化，所以这两个维度都設為 1。所以 pooling 是 2x2 max pooling。

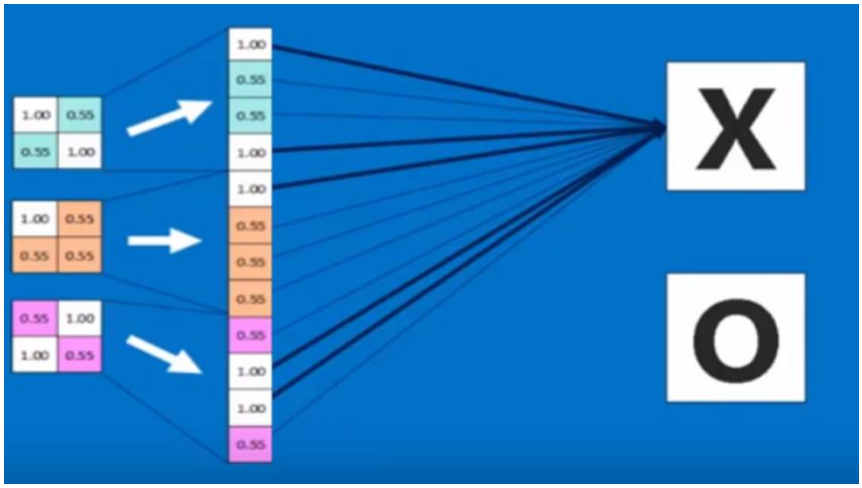
以下為運算模式:



池化函數的用意是把卷積層中重要的特徵保留下來之外，又可以縮減維度，再次增快電腦的速度。

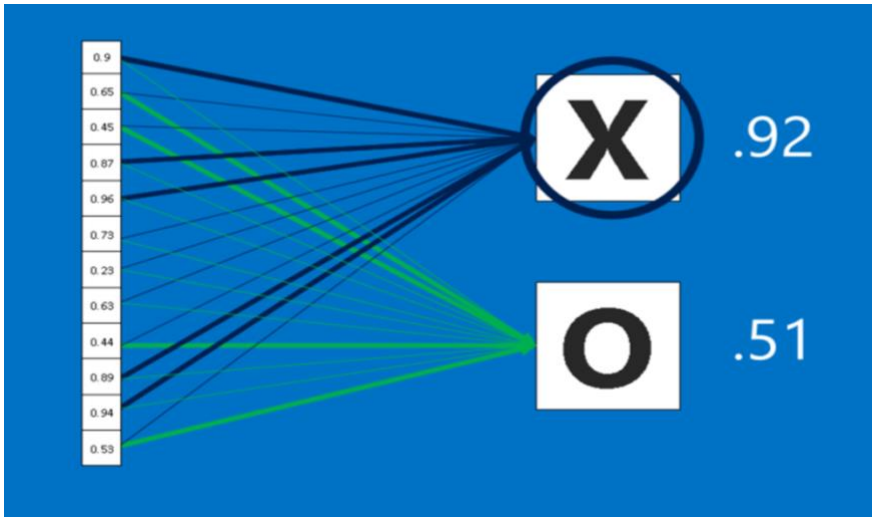
第三層 Fully-Connected Layer

全連結層（fully connected layers），在這層之前的作用主要是提取特徵，而全理解層的作用就是完成最後分類的工作。



首先以簡單的辨別叉與圈來作為例子，經過前面的卷積層後，圖片進入這層時，它會將所有像素的值轉成一個一維向量，有別於前面的多維矩陣形式。

向量裡的每個值都可以決定圖片中的符號是圈還是叉，不過某些值可以更好地判別叉，有些則更適合用來判斷圈，所以每個值得判斷並不一致(如上圖中的線粗細程度)。為了解決所有值對不同選項的影響程度不一的問題，將會以**權重**（weight）或**連結強度**（connection strength）來解決此一問題。



最後可以如上圖，根據乘上權重後的值來決定圖片。

code:

```
# densely connected layer
W_fc1 = weight_variable([4 * 4 * 64, 256])
b_fc1 = bias_variable([256])
h_pool2_flat = tf.reshape(h_pool2, [-1, 4 * 4 * 64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

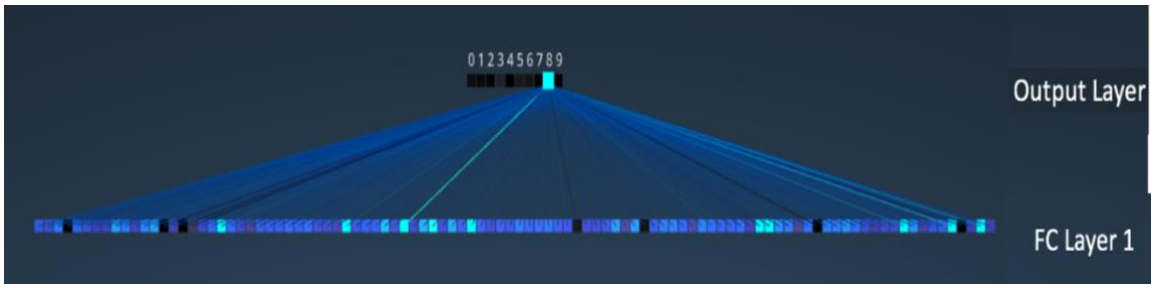
在進入這層時，經過卷積與池化，資料輸入進來為 4*4 的圖片，共 64 個特徵，

在 h_pool2_flat 的部分主要作用為將這些 1024 個特徵拉成一個一維的向量，接下來 W_fc1 為根據平均為零，標準差為 0.1 的常態分配所造出的權重矩陣，最後將一維向量與權重相乘，得到 256 個神經元。

```
# softmax layer
W_fc2 = weight_variable([256, 10])
b_fc2 = bias_variable([10])
y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
```

多個全連結層組合在一起，整個神經網路就可以學習更複雜的特徵組合，所做出的判斷也會更準確。這層的 code 可以視為第二次的全連接層，將前面得到的

256 個神經元和新的權重相乘，最後會轉為 10 個輸出，對應於 10 個數字 0~9，故也稱為輸出層。



（原始資料圖片輸入為數字「8」）

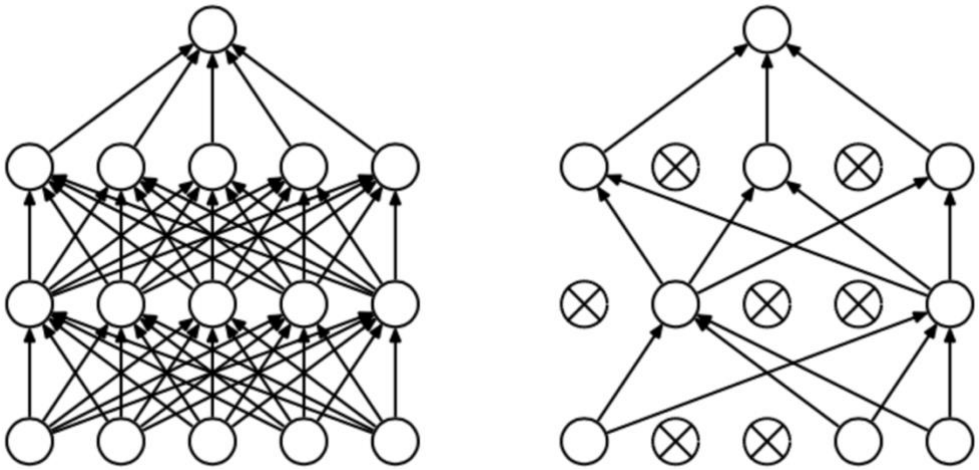
如上圖所示，第一個 FC 層中有 256 個神經元，第二層有 10 個神經元，輸出層的 10 個節點每一個都連接到第一個完全連接層中的全部 256 個節點（因此稱為完全連接）。另外，輸出層中唯一明亮的節點是「8」，這意味著神經網絡對我們的手寫數字進行了正確分類（節點亮度越高表示它的輸出更高，即 8 在所有數字中具有最高的概率）。

Drop out

code:

```
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

dropout 是一種防止 overfitting 的措施，當一筆 training data 進來之後，我們透過設定一個機率選取所有的神經元，讓 training data 只跑我們選到的神經元去訓練參數，當然，未選到的神經元並不是就這樣刪除了，而是下一筆 training data 進來時還是有機會會被選到，而在 testing data 進來時，我們通常就將 keep_prob 設為 1，也就是所有的神經元都使用。



左圖為未使用 dropout，右圖則是使用 dropout 後。

訓練模型(Train model)

code:

```
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
```

1. cross_entropy:

$$H_{y'}(y) = -\sum_i y'_i log(y_i)$$

y_i代表電腦透過模型計算出此圖片屬於 0~9 個別的機率。

y'_i代表圖片正確的 label，ex:圖片真實的數字如果是 0，則 y'_i就會是 [1,0,0,0,0,0,0,0,0,0]。

預測的 y_i越準確的話，$H_{y'}(y)$的值就會越小，因為前面有個負號的關係。

2. tf.train.AdamOptimizer(1e-4):
   使用 Adam 演算法當作我們的優化器幫助我們找尋最小值，1e-4 則是讓我們演算法當中的除法不會除以 0 導致計算死去所加上的一個非常小的常數。

抽取(Random Select)

code:

```
def next_batch(num, data, labels):
    idx = np.arange(0 , len(data))
    np.random.shuffle(idx)
    idx = idx[:num]
    data_shuffle = [data[ i] for i in idx]
    labels_shuffle = [labels[ i] for i in idx]

    return np.asarray(data_shuffle), np.asarray(labels_shuffle)
```

在我們 train model 時，為了加快運行速度，我們每次只丟些許的 data，所以我們定義了 next_batch 的 function，透過這個 function 能 return 給我們指定的數量(num)的 data 與 labels。

結果(Result)

1.

code:

```
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  for i in range(20000):
    Xtr, Ytr = next_batch(50, test['X_tar'].T, test_usps)
    train_step.run(feed_dict={x: Xtr, y_: Ytr, keep_prob: 0.4})
    if i % 100 == 0:
      train_accuracy = accuracy.eval(feed_dict={
          x: Xtr, y_: Ytr, keep_prob: 1.0})
      print('step %d, training accuracy %g' % (i, train_accuracy))
  print('test accuracy %g' % accuracy.eval(feed_dict={
    x:test['X_src'].T, y_: data_usps, keep_prob: 1.0}))
```

```
step 19800, training accuracy 1
step 19900, training accuracy 1
test accuracy 0.637778
```

train data -> Mnist

test data -> Usps

我們重複 20000 次，每次抽取 50 個 train data 中的 data 來 train model，最後將 test data 進行 test，準確率只有 63.78%

2.

code:

```
combined_matrix1 = np.vstack( [test['X_tar'].T,test['X_src'].T[:50]])
combined_label1 = np.vstack([test_usps,data_usps[:50]])
test_matrix1 = test['X_src'].T[50:]
test_label1 = data_usps[50:]
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  for i in range(3000):
    Xtr, Ytr = next_batch(50,combined_matrix1,combined_label1)
    train_step.run(feed_dict={x: Xtr, y_: Ytr, keep_prob: 0.4})
    if i % 100 == 0:
      train_accuracy = accuracy.eval(feed_dict={
          x: Xtr, y_: Ytr, keep_prob: 1.0})
      print('step %d, training accuracy %g' % (i, train_accuracy))
  print('test accuracy %g' % accuracy.eval(feed_dict={
    x:test_matrix1, y_:test_label1, keep_prob: 1.0}))
```

```
step 2800, training accuracy 1
step 2900, training accuracy 1
test accuracy 0.818857
```

train data -> Mnist+Usps(其中 50 個)

test data -> Usps(剩下的 usps)

我們加入 50 筆 usps data 進入 train data 我們重複 3000 次，每次抽取 50 個 train data 中的 data 來 train model，最後將 test data 進行 test，準確率來到 81.89%

3.

code:

```
combined_matrix1 = np.vstack( [test['X_tar'].T,test['X_src'].T[:100]])
combined_label1 = np.vstack([test_usps,data_usps[:100]])
test_matrix1 = test['X_src'].T[100:]
test_label1 = data_usps[100:]
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  for i in range(3000):
    Xtr, Ytr = next_batch(50,combined_matrix1,combined_label1)
    train_step.run(feed_dict={x: Xtr, y_: Ytr, keep_prob: 0.4})
    if i % 100 == 0:
      train_accuracy = accuracy.eval(feed_dict={
          x: Xtr, y_: Ytr, keep_prob: 1.0})
      print('step %d, training accuracy %g' % (i, train_accuracy))
  print('test accuracy %g' % accuracy.eval(feed_dict={
    x:test_matrix1, y_:test_label1, keep_prob: 1.0}))
```

```
step 2800, training accuracy 1
step 2900, training accuracy 0.98
test accuracy 0.862353
```

我們加入 100 筆 usps data 進入 train data 我們重複 3000 次，每次抽取 50 個 train data 中的 data 來 train model，最後將 test data 進行 test，準確率來到 86.24%

參考網址

1.https://blog.csdn.net/stdcoutzyx/article/details/49022443 ____ (dropout 圖片)

2. https://blog.csdn.net/xierhacker/article/details/53174558 ____(Adam 演算法)

3.https://blog.csdn.net/mao_xiao_feng/article/details/53382790____(cross_entropy 數學式子介紹)

4. https://ithelp.ithome.com.tw/articles/10187282____CNN 流程與程式)

5. https://puremonkey2010.blogspot.com/2017/07/toolkit-keras-mnist-cnn.html____(CNN 運算圖解)

6. https://blog.csdn.net/xiaodongxiexie/article/details/74012239____(程式參數說明)

7. https://www.youtube.com/watch?v=2-Ol7ZB0MmU&list=LLEUJhKPny1b4M0HU_LdASWg

8. https://brohrer.github.io/how_convolutional_neural_networks_work.html (introduction of Fully connected layers)