# Deep_learning_HW1_Report

林若瑜_0853420

## Problem_1:

In [821]:
```
#12000筆資料
train_data = np.load('train.npz')
#print(train_data['image'][1])
print(len(train_data['label']))

x_train = train_data['image']
y_train = train_data['label']
```

```
12000
```

In [822]:
```
#5768筆資料
test_data = np.load('test.npz')
print(len(test_data['label']))
x_test = test_data['image']
y_test = test_data['label']
```

```
5768
```

把資料 load 進來，training_size = 12000, testing_size = 5768

In [824]:
```
# onehot encoder
y_train = y_train.reshape(-1, 1)
y_test = y_test.reshape(-1, 1)

enc = OneHotEncoder(handle_unknown='ignore')
enc.fit(y_train)
y_train = enc.transform(y_train).toarray()
y_test = enc.transform(y_test).toarray()

x_train = x_train.reshape(x_train.shape[0], -1, 1)
x_test = x_test.reshape(x_test.shape[0], -1, 1)
y_train = y_train.reshape(y_train.shape[0], -1, 1)
y_test = y_test.reshape(y_test.shape[0], -1, 1)
```

In [827]:
```
# normalize
x_train = x_train/255
x_test = x_test/255
```

In [828]:
```
# 配對 data and label
training_data = list(zip(x_train, y_train))
testing_data = list(zip(x_test, y_test))
```

把 y 的資料改成 onehot 形式，並且把值都正規化（除以 255），並把 train, test 的 x, y 值配對。

In [7]:
```
#激活函數
def sigmoid(x):
    return 1 /(1 + np.exp(-x))

def sigmoid_derivate(x):
    return sigmoid(x) * (1-sigmoid(x))


#loss function
def cross_entropy(output, correct):
    return np.sum( np.nan_to_num(-(correct*np.log(output) + (1-correct)*np.log(1-output))))

def cross_entropy_derivative(output, correct):
    return output - correct
```

建立 sigmoid 當 activation function 和 cross entropy 當 loss function。

In [16]:
```
# initial variable = 0
class model_DNN_zero():
    def __init__(self, layers):
        #設定有幾層神經網路
        self.layer_num = len(layers)
        self.neurons = layers
        # create weights and bias
        self.weights = [ np.zeros((j, i)) for i, j in zip(layers[:-1], layers[1:]) ]
        self.biases = [ np.zeros((i, 1)) for i in layers[1:] ]

        self.training_loss = []
        self.training_error_rate = []
        self.testing_error_rate = []
        self.batches_latent = []
        self.latent_count = 0
        self.latent = []


        #np.random.seed(42)
        #weights = np.random.rand(3,1)
        #bias = np.random.rand(1)
        #lr = 0.05 #Learning rate
```

建立 weight initial 是 0 的模型。

In [17]:
```
# initial variable = random
class model_DNN_random():
    def __init__(self, layers):
        #設定有幾層神經網路
        self.layer_num = len(layers)
        self.neurons = layers
        # create weights and bias
        self.weights = [ np.random.randn(j, i) for i, j in zip(layers[:-1], layers[1:]) ]
        self.biases = [ np.random.randn(i, 1) for i in layers[1:] ]

        self.training_loss = []
        self.training_error_rate = []
        self.testing_error_rate = []
        self.batches_latent = []
        self.latent_count = 0
        self.latent = []

        #np.random.seed(42)
        #weights = np.random.rand(3,1)
        #bias = np.random.rand(1)
        #lr = 0.05 #Learning rate
```

建立 weight initial 是隨機的模型。

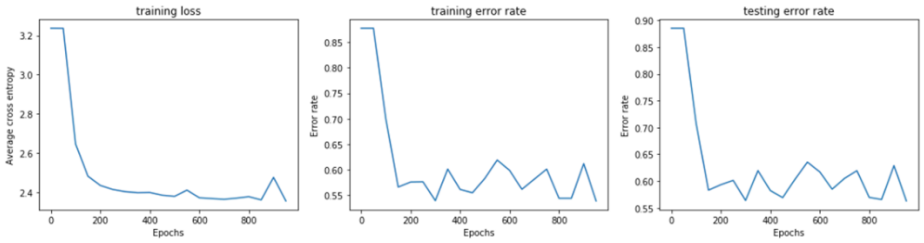
In [*]:
```
module1 = model_DNN_zero([784, 32, 32, 2, 10])
# SGD(self, training_data, testing_data, epochs, batch_size, lr):
module1.SGD(training_data, testing_data, 1000, 100, 0.3)
con_matrix_zero = module1.get_confusion_matrix(testing_data)
```

試 weight initial 是 0 的 model, 設它為 module1，設定輸入為 28*28 個 node，兩層 32 個 node 的 nn，輸出前設為 2 個 node，輸出有 10 個 output(0-9)。設定 Epoch = 1000， batch size = 100, learning rate = 0.3

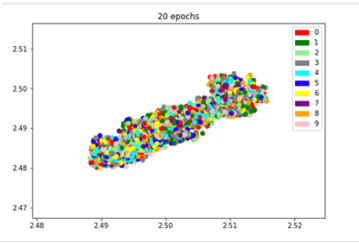
```
==================================
Epoch: 800
    training loss: 2.376703
    training error rate: 6528 <-- 12000 ,rate=(0.544000)
    testing error rate: 3284 <-- 5768 ,rate=(0.569348)
==================================
Epoch: 850
    training loss: 2.360273
    training error rate: 6528 <-- 12000 ,rate=(0.544000)
    testing error rate: 3262 <-- 5768 ,rate=(0.565534)
==================================
Epoch: 900
    training loss: 2.475161
    training error rate: 7342 <-- 12000 ,rate=(0.611833)
    testing error rate: 3627 <-- 5768 ,rate=(0.628814)
==================================
Epoch: 950
    training loss: 2.354856
    training error rate: 6465 <-- 12000 ,rate=(0.538750)
    testing error rate: 3246 <-- 5768 ,rate=(0.562760)
```

結果為上圖。

```
In [44]: new_x_axis = np.arange(0,1000, 50)
         fig, ax = plt.subplots(1, 3)
         fig.set_size_inches(18, 4)
         ax[0].plot(new_x_axis, module1.training_loss)
         ax[0].set_title('training loss')
         ax[0].set_xlabel('Epochs')
         ax[0].set_ylabel('Average cross entropy')


         ax[1].plot(new_x_axis, module1.training_error_rate)
         ax[1].set_title('training error rate')
         ax[1].set_xlabel('Epochs')
         ax[1].set_ylabel('Error rate')

         ax[2].plot(new_x_axis, module1.testing_error_rate)
         ax[2].set_title('testing error rate')
         ax[2].set_xlabel('Epochs')
         ax[2].set_ylabel('Error rate')

Out[44]: Text(0, 0.5, 'Error rate')
```

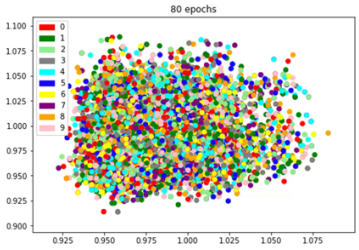



Training loss, training error rate, testing error rate 分別的圖形。



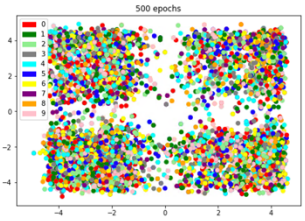

epoch=20

```
In [22]: fig = plt.figure()
         ax = fig.add_axes([0,0,1,1])
         scatter= ax.scatter(latent_feature_80[0], latent_feature_80[1], c = set_color)
         cluster1 = ax.legend(handles = [c0,c1,c2,c3,c4,c5,c6,c7,c8,c9], labels=['0','1','2','3','4','5','6','7','8','9'],  loc='best')
         ax.add_artist(cluster1)
         ax.set_title('80 epochs')
         plt.show()
```



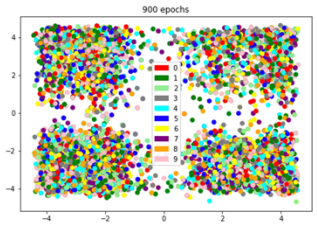

Epoch = 80，仍然沒有分得很好，因此多印了幾個 epoch 的結果。

```
In [24]: latent_feature_500 = np.array(module1.latent[500])
         latent_feature_500 = np.reshape(latent_feature_500, (2,12000))
         true_label = np.array([int(y) for y in train_data['label']])


         fig = plt.figure()
         ax = fig.add_axes([0,0,1,1])
         scatter= ax.scatter(latent_feature_500[0], latent_feature_500[1], c = set_color)
         cluster1 = ax.legend(handles = [c0,c1,c2,c3,c4,c5,c6,c7,c8,c9], labels=['0','1','2','3','4','5','6','7','8','9'],  loc='best')
         ax.add_artist(cluster1)
         ax.set_title('500 epochs')
         plt.show()
```




Epoch = 500，分的比較開了，但還是沒有很明顯分開。

```
In [26]: latent_feature_900 = np.array(module1.latent[900])
         latent_feature_900 = np.reshape(latent_feature_900, (2,12000))
         true_label = np.array([int(y) for y in train_data['label']])


         fig = plt.figure()
         ax = fig.add_axes([0,0,1,1])
         scatter= ax.scatter(latent_feature_900[0], latent_feature_900[1], c = set_color)
         cluster1 = ax.legend(handles = [c0,c1,c2,c3,c4,c5,c6,c7,c8,c9], labels=['0','1','2','3','4','5','6','7','8','9'],  loc='best')
         ax.add_artist(cluster1)
         ax.set_title('900 epochs')
         plt.show()
```




Epoch = 900，分的更開了，但準確率還是沒有很高。

```
In [27]: print(con_matrix_zero)

[[568  79  15   0   0   0   1   0   0   1]
 [368 242  50   0   0   0   0   0   0   1]
 [ 17  85 455  24   0   0   0   0   0   3]
 [ 19  26 321 105   0   0   8   0   0 121]
 [  1   0   0   0   0   0  53   0   0 597]
 [  0   0   0   0   0   0 349   0   0  58]
 [  0   0   0   0   0   0 485   0   0  17]
 [  0   0   0   1   0   0 299   0   0 149]
 [  0   0   0   0   0   0  17   0   0 555]
 [  0   0   4  14   0   0  18   0   0 642]]
```

Confusion matrix 分佈，4,5,7,8 的辨識率很差，9 辨識率最好。

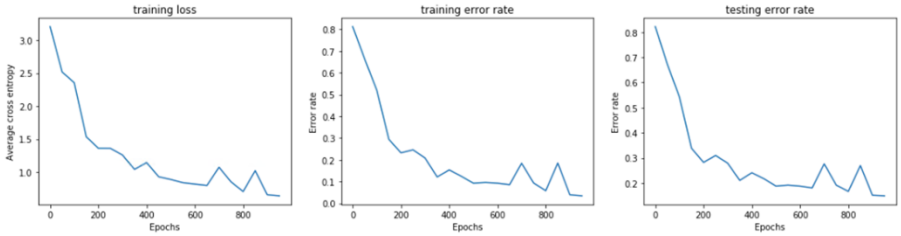
```
In [45]: module2 = model_DNN_random([784, 32, 32, 2, 10])
         module2.SGD(training_data, testing_data, 1000, 100, 0.3)
         con_matrix_random = module2.get_confusion_matrix(testing_data)
```

試 weight initial 是隨機的 model, 設它為 module2，設定輸入為 28*28 個 node，兩層 32 個 node 的 nn，輸出前設為 2 個 node，輸出有 10 個 output(0-9)。設定 Epoch = 1000， batch size = 100, learning rate = 0.3

```
    training loss: 0.852008
==================================
Epoch: 800
    training error rate: 695 <-- 12000 ,rate=(0.057917)
    testing error rate: 964 <-- 5768 ,rate=(0.167129)
    training loss: 0.706923
==================================
Epoch: 850
    training error rate: 2220 <-- 12000 ,rate=(0.185000)
    testing error rate: 1556 <-- 5768 ,rate=(0.269764)
    training loss: 1.022169
==================================
Epoch: 900
    training error rate: 467 <-- 12000 ,rate=(0.038917)
    testing error rate: 875 <-- 5768 ,rate=(0.151699)
    training loss: 0.657034
==================================
Epoch: 950
    training error rate: 407 <-- 12000 ,rate=(0.033917)
    testing error rate: 861 <-- 5768 ,rate=(0.149272)
    training loss: 0.640309
```

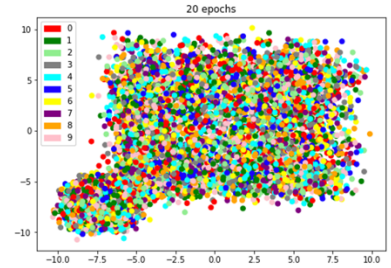
結果為上圖，明顯比 weight initial = 0 的結果好。

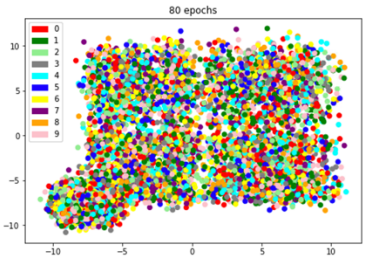
```
In [47]: new_x_axis = np.arange(0,1000, 50)
fig, ax = plt.subplots(1, 3)
fig.set_size_inches(18, 4)
ax[0].plot(new_x_axis, module2.training_loss)
ax[0].set_title('training loss')
ax[0].set_xlabel('Epochs')
ax[0].set_ylabel('Average cross entropy')


ax[1].plot(new_x_axis, module2.training_error_rate)
ax[1].set_title('training error rate')
ax[1].set_xlabel('Epochs')
ax[1].set_ylabel('Error rate')

ax[2].plot(new_x_axis, module2.testing_error_rate)
ax[2].set_title('testing error rate')
ax[2].set_xlabel('Epochs')
ax[2].set_ylabel('Error rate')

Out[47]: Text(0, 0.5, 'Error rate')
```




Training loss, training error rate, testing error rate 分別的圖形。

```
In [49]: fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
scatter= ax.scatter(latent_feature_80[0], latent_feature_80[1], c = set_c
cluster1 = ax.legend(handles = [c0,c1,c2,c3,c4,c5,c6,c7,c8,c9], labels=['
ax.add_artist(cluster1)
ax.set_title('80 epochs')
plt.show()
```

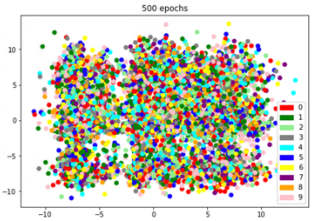


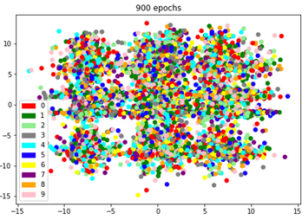

在 epoch = 20 的時候，
就相比 moduel1 分得蠻開的了。

此圖為 Epoch = 80

```
In [50]: latent_feature_500 = np.array(module2.latent[500])
latent_feature_500 = np.reshape(latent_feature_500, (2,12000))
true_label = np.array([int(y) for y in train_data['label']])


fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
scatter= ax.scatter(latent_feature_500[0], latent_feature_500[1], c = s
cluster1 = ax.legend(handles = [c0,c1,c2,c3,c4,c5,c6,c7,c8,c9], labels=
ax.add_artist(cluster1)
ax.set_title('500 epochs')
plt.show()
```




```
In [51]: latent_feature_900 = np.array(module2.latent[900])
latent_feature_900 = np.reshape(latent_feature_900, (2,12000))
true_label = np.array([int(y) for y in train_data['label']])


fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
scatter= ax.scatter(latent_feature_900[0], latent_feature_900[1], c =
cluster1 = ax.legend(handles = [c0,c1,c2,c3,c4,c5,c6,c7,c8,c9], label
ax.add_artist(cluster1)
ax.set_title('900 epochs')
plt.show()
```




此圖為 Epoch = 500

此圖為 Epoch = 900，比較有在分開的趨勢了。

```
In [46]: print(con_matrix_random)
```

```
[[640  17   1   1   0   0   2   1   0   2]
 [ 10 638  11   2   0   0   0   0   0   0]
 [  3  30 488  60   0   0   0   0   2   1]
 [  2   4  69 490   0   2   3  11  18   1]
 [  4   2   0   8 525  33  15  14   6  44]
 [  0   0   0   0   4 371  19  11   1   1]
 [  0   0   0   2   0  12 464  11  13   0]
 [  0   0   1  11  12  15  86 281  37   6]
 [  0   0   0  78   6  12  26  12 438   0]
 [ 38   4   3   3  47   5   1   9   2 566]]
```

Confusion matrix 分佈，大部份都有判斷對，效果明顯比 moduel1 的好很多。

## Problem_2:

```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')

train_filename = train_df['filename']
train_xmin = train_df['xmin']
train_xmax = train_df['xmax']
train_ymin = train_df['ymin']
train_ymax = train_df['ymax']

test_filename = test_df['filename']
test_xmin = test_df['xmin']
test_xmax = test_df['xmax']
test_ymin = test_df['ymin']
test_ymax = test_df['ymax']
```

讀入檔案

```
In [284]: #處理train資料切割(3528)

          #圖片壓縮至同一大小
          IMG_SIZE = 80


          train_x = []
          train_index = []
          #for i in range(len(train_filename)):
          for i in range(3528):
              try:
                  filepath = 'C:\\Users\\user\\Desktop\\images\\'+ train_filename[i]
                  x1 = train_xmin[i]
                  x2 = train_xmax[i]
                  y1 = train_ymin[i]
                  y2 = train_ymax[i]

          #         img = cv2.imread(filepath)
                  img = cv2.imread(filepath,cv2.IMREAD_GRAYSCALE)
                  name = 'img'+ str(i)
                  sub_img = img [x1:x2, y1:y2]

                  resized_sub_img = cv2.resize(sub_img, (IMG_SIZE, IMG_SIZE))
                  train_x.append(resized_sub_img)
                  train_index.append(i)

                  save_path = 'C:\\Users\\user\\Desktop\\train_sub_images\\'+ name + '.jpg'
                  cv2.imwrite(save_path, resized_sub_img)
              except:
                  pass
```

把 train, test 資料依照 excel 檔裡的位置切割，轉為灰階，編號後另存於 train_sub_images, test_sub_images 兩資料夾中，中間設定 try 方法，若切割過程中有問題就 pass。上示意圖為 train 的部分。

In [286]:
```
# train / test data normailzation
x_train = []
x_test = []
for x in train_x:
    x_normalize = x/255
    x_normalize = np.float32(x_normalize)
    x_train.append(x_normalize)
for x in test_x:
    x_normalize = x/255
    x_normalize = np.float32(x_normalize)
    x_test.append(x_normalize)


# encoding label to 0.1.2
y_train = []
y_test = []
train_label = train_df['label']
test_label = test_df['label']

for x in train_index:
    if train_label[x] == 'good':
        y_train.append(0)
    elif train_label[x] == 'bad':
        y_train.append(1)
    else:
        y_train.append(2)
for x in test_index:
    if test_label[x] == 'good':
        y_test.append(0)
    elif test_label[x] == 'bad':
        y_test.append(1)
    else:
        y_test.append(2)
```

把 train, test 正規化，並把 y_train, y_test 轉換為數字形式。

Good = 0, bad = 1, none = 2。

In [287]:
```
# onehot encode
y_train = np.array(y_train).reshape(-1, 1)
y_test = np.array(y_test).reshape(-1, 1)

onehot = OneHotEncoder(handle_unknown='ignore')
onehot.fit(y_train)
y_train = onehot.transform(y_train).toarray()
y_test = onehot.transform(y_test).toarray()
```

In [288]:
```
y_train.shape
```

Out[288]:
```
(2518, 3)
```

把 y_train, y_test 轉成 onehot 形式。

In [293]:
```
# reshape train, test data
x_train = np.array(x_train).reshape(-1,IMG_SIZE, IMG_SIZE, 1)
x_test = np.array(x_test).reshape(-1,IMG_SIZE, IMG_SIZE, 1)
```

把 input 資料 reshape 成可以輸入神經網路的形式。

[sample_size, image_size, image_size, gray=1]。

In [297]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# tf.set_random_seed(1)
# np.random.seed(1)

batch_size = 32
LR = 0.001

def next_batch(train_data, train_target, batch_size):
    index = [ i for i in range(0,len(train_target)) ]
    np.random.shuffle(index);
    batch_data = [];
    batch_target = [];
    for i in range(0,batch_size):
        batch_data.append(train_data[index[i]]);
        batch_target.append(train_target[index[i]])
#     print("batch_data",batch_data[0].shape,"batch_target", batch_target[0].shape )
    return batch_data, batch_target

# # define placeholder for inputs to network
xs = tf.placeholder(tf.float32, [None, IMG_SIZE, IMG_SIZE, 1])/255.
ys = tf.placeholder(tf.float32, [None, 3])
keep_prob = tf.placeholder(tf.float32)
# # print(x_image.shape)  # [n_samples, 100,100,1]
```

用 tensorflow 建立 cnn 模型。用 next_batch 方法切割 batch

Batch size = 32, learning rate = 0.001

```
# CNN1
conv1 = tf.layers.conv2d(inputs=xs, filters=32, kernel_size=5, strides=1, padding='same', activation=tf.nn.relu)
pool1 = tf.layers.max_pooling2d(conv1, pool_size=2, strides=2)           # -> (50,50, 32)
conv2 = tf.layers.conv2d(pool1, 64, 5, 1, 'same', activation=tf.nn.relu)    # -> (50,50, 64)
pool2 = tf.layers.max_pooling2d(conv2, 2, 2)    # -> (25,25, 64)


flat = tf.reshape(pool2, [-1, int(IMG_SIZE/4*IMG_SIZE/4*64)])
dropout = tf.layers.dropout(flat, rate=0.5, noise_shape=None, seed=None, training=False, name=None)
```

測試的第 1 個 CNN 模型→CNN1

Input→32→pooling→64→pooling→flatten→ dropout(0.5)→output

Stride = 1, padding = same, opt = adam, loss = cross entropy

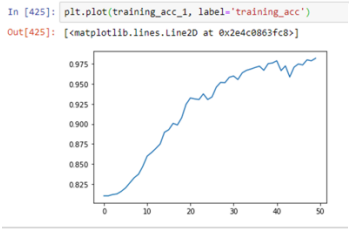
設定每跑 50 step 印一次 train & test accuracy, train loss  總共跑 50 個 epoch。

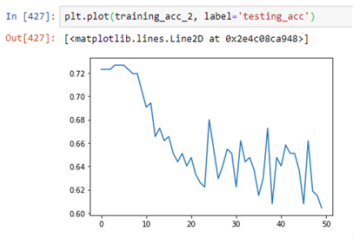
```
epoch: 49
TRAIN
[[2027   13    0]
 [  30  363    0]
 [   2    0   83]]
good: 0.9936274509803922
bad: 0.9236641221374046
none: 0.9764705882352941
TEST
[[159  35   7]
 [ 51   8   0]
 [ 15   2   1]]
good: 0.7910447761194029
bad: 0.13559322033898305
none: 0.05555555555555555
```

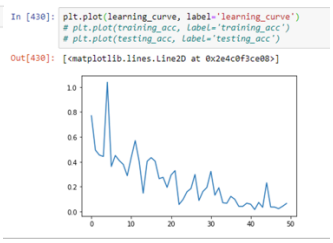
通過 confusion matrix 可得知：

最終 train accuracy = 0.98212867,  test accuracy = 0.60431655

三個分類分別的 accuracy，train 都表現得很好，但 test 表現差很多，其中 bad, none 都比 good 差。可能是有 overfitting 的現象，因此下一個模型會把 dropout 調大一點。




X 軸為%數，y 軸為 epoch 數。

分別為 training_acc, testing_acc, learning_curve

其中 testing 呈現下滑趨勢，可能是因為資料比例過於懸殊，一開始全部都判斷成 good 的準確率都比訓練完再去分類的準確率來得高。

```
# CNN2 dropout
conv1 = tf.layers.conv2d(inputs=xs, filters=32, kernel_size=5, strides=1, padding='same', activation=tf.nn.relu)
pool1 = tf.layers.max_pooling2d(conv1, pool_size=2, strides=2)
conv2 = tf.layers.conv2d(pool1, 64, 5, 1, 'same', activation=tf.nn.relu)
pool2 = tf.layers.max_pooling2d(conv2, 2, 2)


flat = tf.reshape(pool2, [-1, int(IMG_SIZE/4*IMG_SIZE/4*64)])
dropout = tf.layers.dropout(flat, rate=0.7, noise_shape=None, seed=None, training=False, name=None)
```

測試的第 2 個 CNN 模型→CNN2

Input→32→pooling→64 →flatten→dropout(0.7)→output

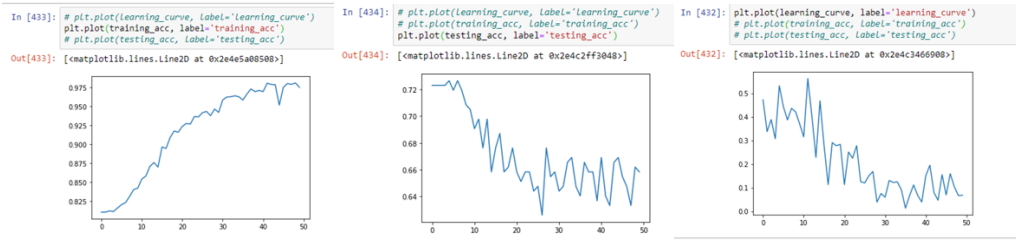
Stride = 1,padding = same, opt = adam, loss = cross entropy

```
*TRAIN*
[[2030   10    0]
 [  34  359    0]
 [   3    1   81]]
 good: 0.9950980392156863
 bad: 0.9134860050890585
 none: 0.9529411764705882
*TEST*
[[175  22   4]
 [ 50   9   0]
 [ 15   3   0]]
 good: 0.8706467661691543
 bad: 0.15254237288135594
 none: 0.0
```

通過 confusion matrix 可得知：

最終 train accuracy = 0.97498014,　test accuracy = 0.65827338

三個分類分別的 accuracy，相比第一個模型，train test 都有表現好一點，且 test 中 bad 和 good 都有更准一點。但可能仍然有 overfitting 的現象。




分別為 training_acc, testing_acc, learning_curve，

其中 testing 呈現下滑趨勢，可能是因為資料比例過於懸殊，一開始全部都判斷成 good 的準確率都比訓練完再去分類的準確率來得高，但這次最終準確度有比第一個模型再提高一點。

```
# CNN3 stride
conv1 = tf.layers.conv2d(inputs=xs, filters=32, kernel_size=5, strides=2, padding='same', activation=tf.nn.relu)
pool1 = tf.layers.max_pooling2d(conv1, pool_size=2, strides=2)
conv2 = tf.layers.conv2d(pool1, 64, 5, 2, 'same', activation=tf.nn.relu)
pool2 = tf.layers.max_pooling2d(conv2, 2, 2)


flat = tf.reshape(pool2, [-1, int(IMG_SIZE/16*IMG_SIZE/16*64)])
dropout = tf.layers.dropout(flat, rate=0.5, noise_shape=None, seed=None, training=False, name=None)
```

測試的第 3 個 CNN 模型→CNN3

Input→32→pooling→64→ pooling→flatten→ dropout(0.5)→output

Stride = 2, padding = same, opt = adam, loss = cross entropy

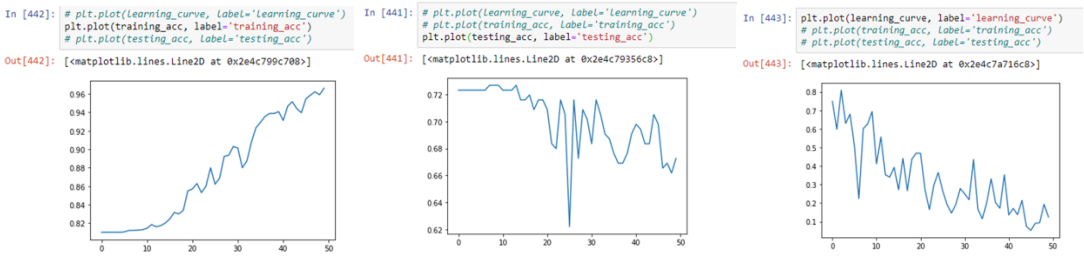
```
*TRAIN*
[[2030    8    2]
 [  62  330    1]
 [  11    1   73]]
 good: 0.9950980392156863
 bad: 0.8396946564885496
 none: 0.8588235294117647
*TEST*
[[180  19   2]
 [ 51   7   1]
 [ 15   3   0]]
 good: 0.8955223880597015
 bad: 0.11864406779661017
 none: 0.0
```

通過 confusion matrix 可得知：

最終 train accuracy = 0.96624305,　test accuracy = 0.67266187

三個分類分別的 accuracy，相比前兩個模型，train 變差了，但 test 有表現更好一點，但 test 中 bad 和 none 都又差了一點，但因為 train 準確度有下降，test

有上升一點，因此 overfitting 的情況應該有改善一點。




分別為 training_acc, testing_acc, learning_curve，

其中 testing 呈現下滑趨勢，可能是因為資料比例過於懸殊，一開始全部都判斷成 good 的準確率都比訓練完再去分類的準確率來得高，但這次最終準確度有比前兩個模型再提高一點。

```
# CNN4 filter
conv1 = tf.layers.conv2d(inputs=xs, filters=64, kernel_size=5, strides=2, padding='same', activation=tf.nn.relu)
pool1 = tf.layers.max_pooling2d(conv1, pool_size=2, strides=2)
conv2 = tf.layers.conv2d(pool1, 64, 5, 2, 'same', activation=tf.nn.relu)
pool2 = tf.layers.max_pooling2d(conv2, 2, 2)
flat = tf.reshape(pool2, [-1, int(IMG_SIZE/16*IMG_SIZE/16*64)])


dropout = tf.layers.dropout(flat, rate=0.5, noise_shape=None, seed=None, training=False, name=None)
```

測試的第 4 個 CNN 模型→CNN4

Input→64→pooling→64→ pooling→flatten→output

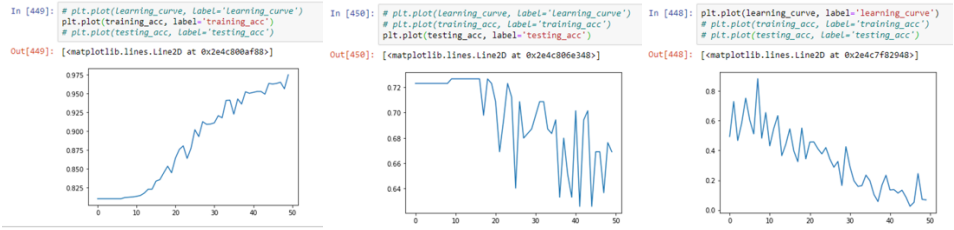
Stride = 1, padding = same, opt = adam, loss = cross entropy

```
*TRAIN*
[[2026   14    0]
 [  37  356    0]
 [   8    4   73]]
  good: 0.9931372549019608
  bad: 0.905852417302799
  none: 0.8588235294117647
*TEST*
[[177  23   1]
 [ 50   8   1]
 [ 14   3   1]]
  good: 0.8805970149253731
  bad: 0.13559322033898305
  none: 0.05555555555555555
```

通過 confusion matrix 算出：

最終 train accuracy = 0.97498014,　test accuracy = 0.66906474。

三個分類分別的 accuracy，相比前兩個模型，train 變差了，但 test 有表現更好一點，但 test 中 none 比之前的模型都高一點，但因為 train 準確度有下降，test 有上升一點，因此 overfitting 的情況應該有改善一點。




分別為 training_acc, testing_acc, learning_curve，

其中 testing 呈現下滑趨勢，可能是因為資料比例過於懸殊，一開始全部都判斷

成 good 的準確率都比訓練完再去分類的準確率來得高，但這次最終準確度有維持，和第三個模型差不多高。