HW3_P3_Jha_Vibhav

April 27, 2021

0.1 HW3 Problem 3

0.2 Name: Vibhav Jha

0.2.1 Imports

[149]:
```
import struct
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
import sklearn
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
import random
```

0.2.2 1. Loading the data

[3]:
```
# training images
with open('train-images-idx3-ubyte','rb') as f:
    magic, size = struct.unpack(">II", f.read(8))
    nrows, ncols = struct.unpack(">II", f.read(8))
    train_data = np.fromfile(f, dtype=np.dtype(np.uint8).newbyteorder('>'))
    train_data = train_data.reshape((size, nrows, ncols))
# training labels
with open('train-labels-idx1-ubyte','rb') as f:
    magic, size = struct.unpack(">II", f.read(8))
    train_labels = np.fromfile(f, dtype=np.dtype(np.uint8).newbyteorder('>'))

# test images
with open('t10k-images-idx3-ubyte','rb') as f:
    magic, size = struct.unpack(">II", f.read(8))
    nrows, ncols = struct.unpack(">II", f.read(8))
    test_data = np.fromfile(f, dtype=np.dtype(np.uint8).newbyteorder('>'))
    test_data = test_data.reshape((size, nrows, ncols))

# test labels
with open('t10k-labels-idx1-ubyte','rb') as f:
    magic, size = struct.unpack(">II", f.read(8))
    test_labels = np.fromfile(f, dtype=np.dtype(np.uint8).newbyteorder('>'))
```

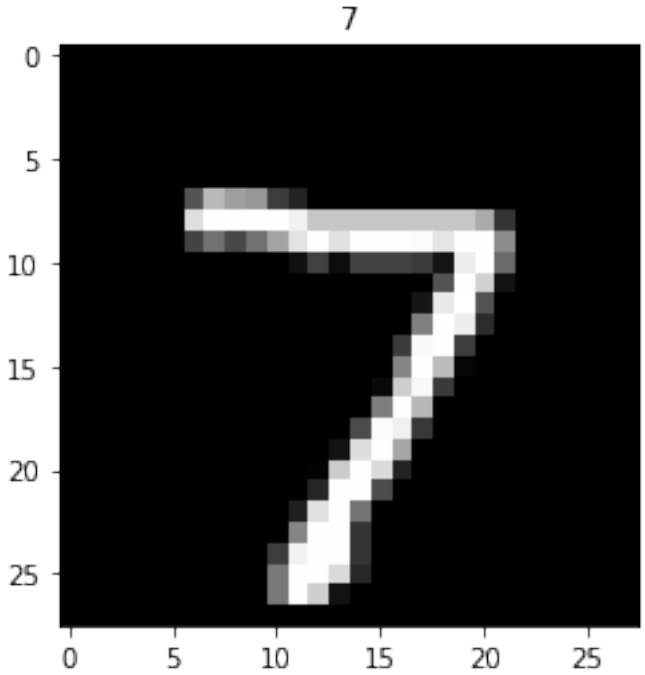
[5]:
```
print('Train Data Shape', np.shape(train_data))

print('Train Label Shape', np.shape(train_labels))

print('Test Data Shape', np.shape(test_data))

print('Test Label Shape', np.shape(test_labels))
```

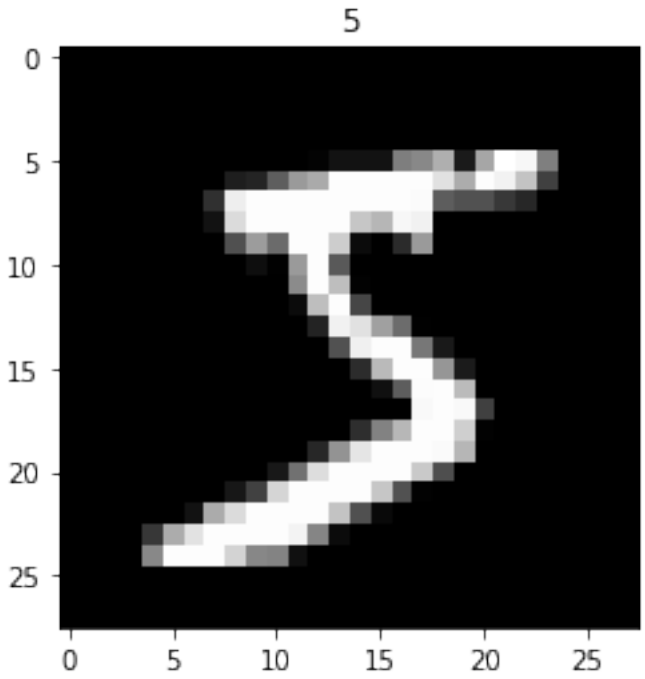
```
Train Data Shape (60000, 28, 28)
Train Label Shape (60000,)
Test Data Shape (10000, 28, 28)
Test Label Shape (10000,)
```

c. Image plots

[10]:
```
plt.figure(1)
plt.imshow(test_data[0], cmap='gray')
plt.title(test_labels[0])

plt.figure(2)
plt.imshow(train_data[0], cmap='gray')
plt.title(train_labels[0])

print('The labels and images match.')
```

```
The labels and images match.
```

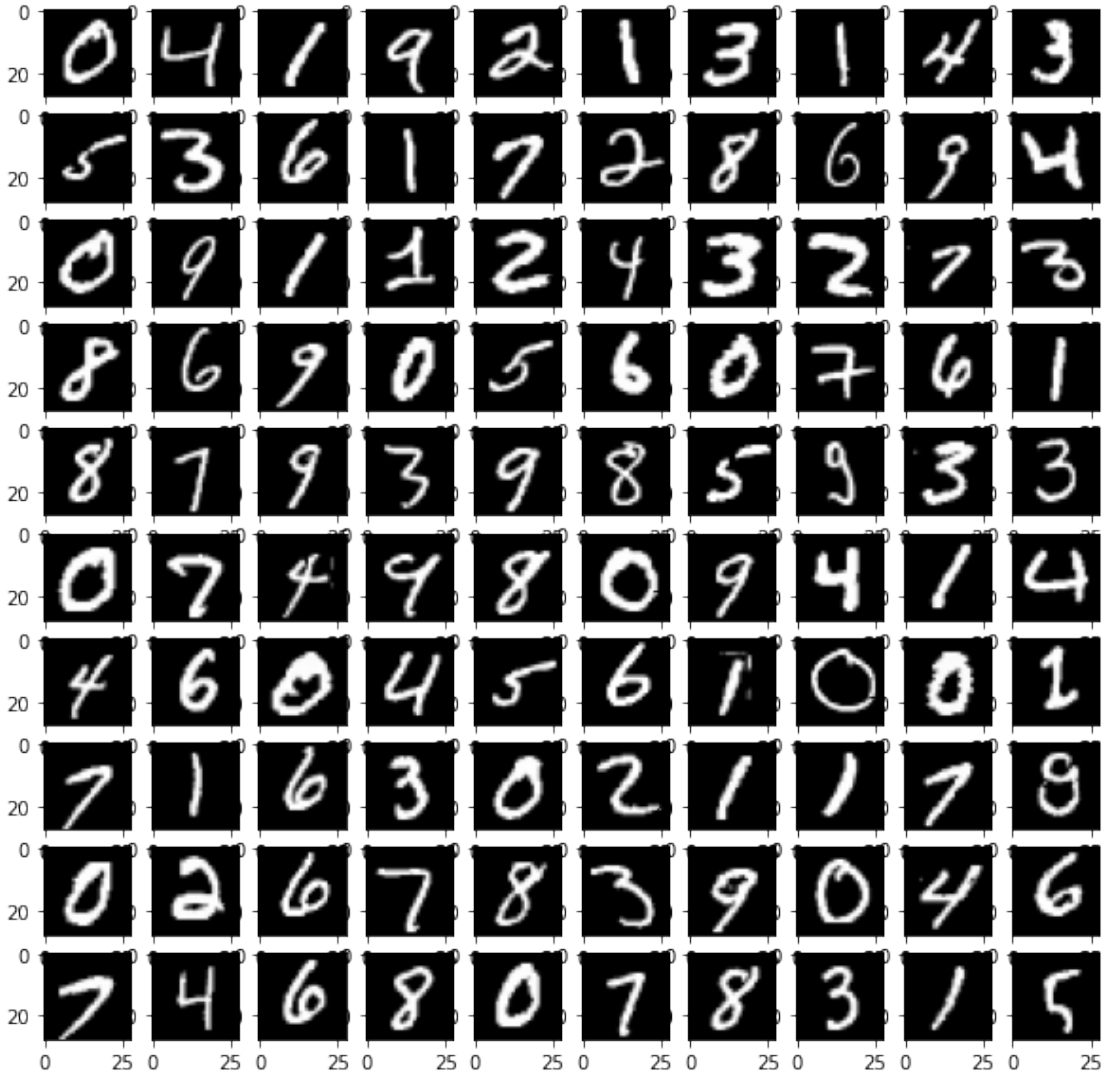


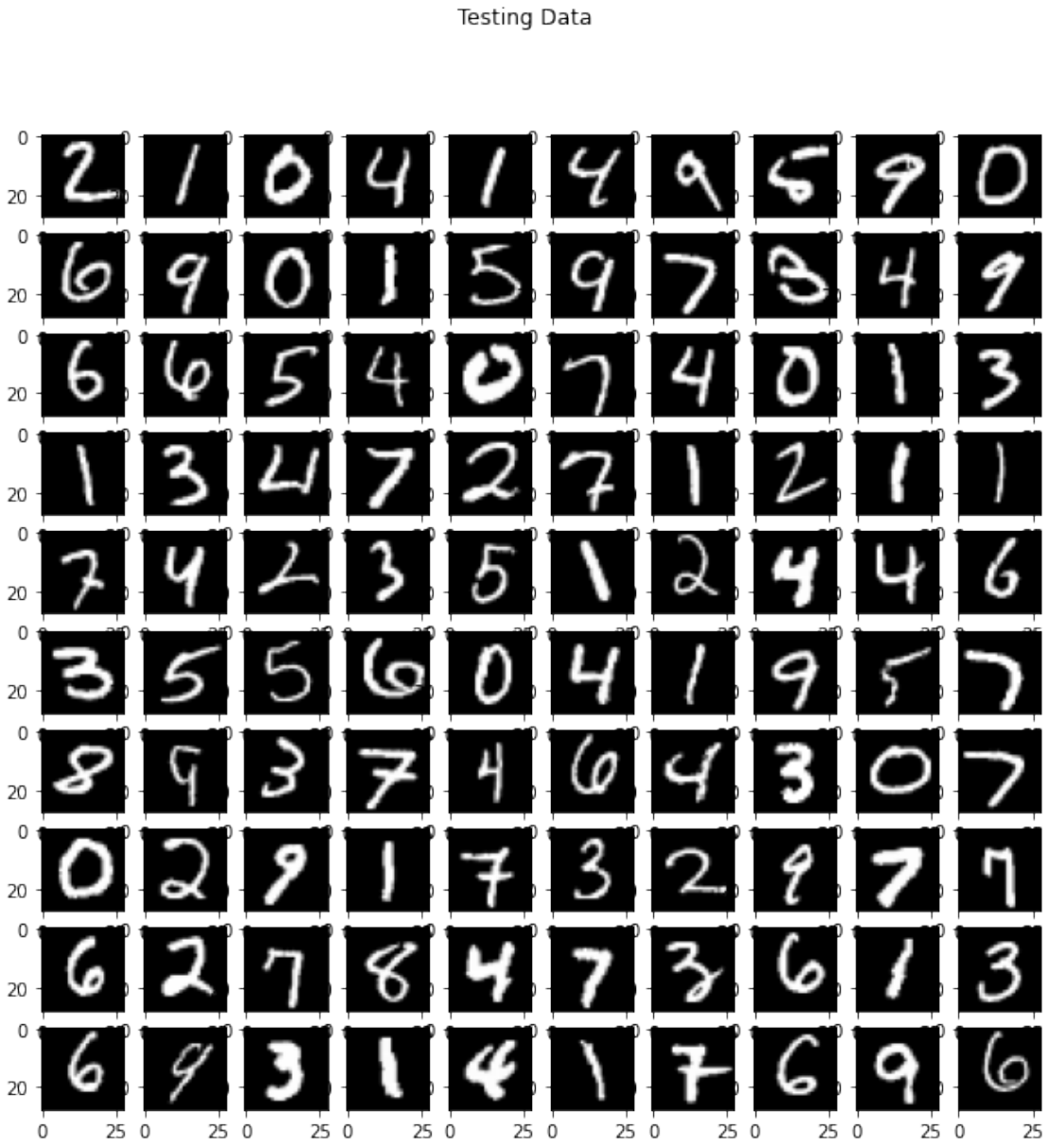


d. Image plot (10x10 grid)

[15]:
```
plt.subplots(figsize=(10,10))
for c in range(10):
    for r in range(10):
        a = r*10 + c + 1
        plt.subplot(10,10,a)
        plt.imshow(train_data[a], cmap = 'gray')
plt.suptitle('Training Data')

plt.subplots(figsize=(10,10))
for c in range(10):
    for r in range(10):
        a = r*10 + c + 1
        plt.subplot(10,10,a)
        plt.imshow(test_data[a], cmap = 'gray')
plt.suptitle('Testing Data')
```

[15]:
```
Text(0.5, 0.98, 'Testing Data')
```


Training Data
0
20
0
25



e. Digit frequency

[29]:
```
trainlab100 = train_labels[:100]
testlab100 = test_labels[:100]
#a = np.where(trainlab100==0)
#print(a)
#np.size(a)

print('Occurences of 0 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==0)))
```

```
print('Occurences of 1 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==1)))
print('Occurences of 2 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==2)))
print('Occurences of 3 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==3)))
print('Occurences of 4 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==4)))
print('Occurences of 5 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==5)))
print('Occurences of 6 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==6)))
print('Occurences of 7 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==7)))
print('Occurences of 8 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==8)))
print('Occurences of 9 in the first 100 train labels: ', np.size(np.
 ↪where(trainlab100==9)))
print('')
print('Occurences of 0 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==0)))
print('Occurences of 1 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==1)))
print('Occurences of 2 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==2)))
print('Occurences of 3 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==3)))
print('Occurences of 4 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==4)))
print('Occurences of 5 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==5)))
print('Occurences of 6 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==6)))
print('Occurences of 7 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==7)))
print('Occurences of 8 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==8)))
print('Occurences of 9 in the first 100 test labels: ', np.size(np.
 ↪where(testlab100==9)))
```

```
Occurences of 0 in the first 100 train labels:  13
Occurences of 1 in the first 100 train labels:  14
Occurences of 2 in the first 100 train labels:  6
Occurences of 3 in the first 100 train labels:  11
Occurences of 4 in the first 100 train labels:  11
Occurences of 5 in the first 100 train labels:  5
```

```
Occurences of 6 in the first 100 train labels:  11
Occurences of 7 in the first 100 train labels:  10
Occurences of 8 in the first 100 train labels:  8
Occurences of 9 in the first 100 train labels:  11

Occurences of 0 in the first 100 test labels:  8
Occurences of 1 in the first 100 test labels:  14
Occurences of 2 in the first 100 test labels:  8
Occurences of 3 in the first 100 test labels:  11
Occurences of 4 in the first 100 test labels:  14
Occurences of 5 in the first 100 test labels:  7
Occurences of 6 in the first 100 test labels:  10
Occurences of 7 in the first 100 test labels:  15
Occurences of 8 in the first 100 test labels:  2
Occurences of 9 in the first 100 test labels:  11
```

0.2.3 2. Data prepartion

Normalization and reshaping

[296]:
```
train_labels6 = train_labels[:6000]
train_data6 = train_data[:6000]
print('Occurences of 0 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==0)))
print('Occurences of 1 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==1)))
print('Occurences of 2 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==2)))
print('Occurences of 3 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==3)))
print('Occurences of 4 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==4)))
print('Occurences of 5 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==5)))
print('Occurences of 6 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==6)))
print('Occurences of 7 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==7)))
print('Occurences of 8 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==8)))
print('Occurences of 9 in the first 6000 train labels: ', np.size(np.
 ↪where(train_labels6==9)))

#scaler = sklearn.preprocessing.MinMaxScaler()
#train_norm = sklearn.preprocessing.MinMaxScaler(train_data6)
#train_norm = scaler.train_data6_
#np.shape(train_norm) this didn't work for some reason, so doing it manually␣
 ↪using a for loop
```

```
#same story for preprocessing.normalize()

train_data_norm = np.zeros([0,784])
for i in train_data6: #general normalization formula
    train_data_norm = np.vstack((train_data_norm,((i - np.min(i)) / (np.max(i) -␣
 ↪np.min((i)))).flatten()))

train_data_norm = train_data_norm.T #transpose to get ndims x nsamples
print('New Train Data Shape: ',np.shape(train_data_norm))

test_data_norm = np.zeros([0,784])
for ii in test_data:
    test_data_norm = np.vstack((test_data_norm,((ii - np.min(i)) / (np.max(ii) -␣
 ↪np.min((ii)))).flatten()))

test_data_norm = test_data_norm.T
print('New Test Data Shape: ',np.shape(test_data_norm))
```

```
Occurences of 0 in the first 6000 train labels:  592
Occurences of 1 in the first 6000 train labels:  671
Occurences of 2 in the first 6000 train labels:  581
Occurences of 3 in the first 6000 train labels:  608
Occurences of 4 in the first 6000 train labels:  623
Occurences of 5 in the first 6000 train labels:  514
Occurences of 6 in the first 6000 train labels:  608
Occurences of 7 in the first 6000 train labels:  651
Occurences of 8 in the first 6000 train labels:  551
Occurences of 9 in the first 6000 train labels:  601
New Train Data Shape:  (784, 6000)
New Test Data Shape:  (784, 10000)
```

One-hot encoding of labels

[297]:
```
train_int = np.zeros([0,10])

for jj in range(6000):
    train_int = np.vstack((train_int, (np.arange(10) == train_labels[jj].
 ↪astype(int))))
train_int = train_int.T

print('1-hot encoded Train Labels: ',np.shape(train_int))
print('Train', train_labels[0], ' ', train_int[:,0])

test_int = np.zeros([0,10])
for jj in range(10000):
    test_int = np.vstack((test_int, (np.arange(10) == test_labels[jj].
 ↪astype(int))))
test_int = test_int.T
```

```
print('1-hot encoded Test Labels: ',np.shape(test_int))
print('Test ', test_labels[0], ' ', test_int[:,0])
```

```
1-hot encoded Train Labels:  (10, 6000)
Train 5   [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
1-hot encoded Test Labels:  (10, 10000)
Test  7   [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
```

0.2.4 3. Neural Network

Computational graph

[62]:
```
tf.reset_default_graph() #heavily adapted from Michelucci p.110-111 and recitatio
#784 as it is 28*28
learning_rate = tf.placeholder(tf.float64, shape=())
X = tf.placeholder(tf.float64, [784, None])
Y = tf.placeholder(tf.float64, [10, None])

#want 10 neurons, so 10 weights
weights = tf.Variable(tf.random_normal(shape = [10, 784], dtype=tf.float64,␣
 ↪seed=12345))

bias = tf.Variable(tf.zeros([10,1], tf.float64))

out = tf.sigmoid(tf.matmul(weights, X) + bias)

#from Michelucci p108 with the inclusion of the no nan from p77
cost = - tf.reduce_mean(tf.math.multiply_no_nan(Y,tf.log(out)) + tf.math.
 ↪multiply_no_nan((1-Y),tf.log(1-out)), axis=1 )

optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

init = tf.global_variables_initializer()

saver = tf.train.Saver()
```

Training function

[114]:
```
def mnist_trainer(epochs, trainx, trainlabels, lrate, costf, optimizerf):
    sess = tf.Session()
    sess.run(init)

    cost_history = np.empty(shape=[10], dtype = float)
    for ee in range(epochs):
        [_, cost_] = sess.run([optimizer, cost], feed_dict = {X: trainx, Y:
 ↪trainlabels, learning_rate:lrate})

        cost_history = np.append(cost_history, np.mean(cost_))
```

```
        if (ee%500 == 0):
            print('Cost = ', np.mean(cost_), 'at epoch ', ee)
            save_mod = saver.save(sess,'trained_model_' + str(lrate) + '_' +␣
 ↪str(epochs) + '.ckpt')
    return sess, cost_history, save_mod


s1, chist1, saved1 = mnist_trainer(epochs = 10001,
                                   trainx = train_data_norm,
                                   trainlabels = train_int,
                                   lrate = 0.05,
                                   costf = cost,
                                   optimizerf = optimizer)
s1.close()

s2, chist2, saved2 = mnist_trainer(epochs = 50001,
                                   trainx = train_data_norm,
                                   trainlabels = train_int,
                                   lrate = 0.05,
                                   costf = cost,
                                   optimizerf = optimizer)
s2.close()

s3, chist3, saved3 = mnist_trainer(epochs = 50001,
                                   trainx = train_data_norm,
                                   trainlabels = train_int,
                                   lrate = 0.01,
                                   costf = cost,
                                   optimizerf = optimizer)
s3.close()
```

```
Cost =  2.1182936721476553 at epoch  0
Cost =  0.27998292388249524 at epoch  500
Cost =  0.20307103433352286 at epoch  1000
Cost =  0.17104236086479957 at epoch  1500
Cost =  0.1523633175459273 at epoch  2000
Cost =  0.1397022359518514 at epoch  2500
Cost =  0.13033701063250494 at epoch  3000
Cost =  0.1230082450100857 at epoch  3500
Cost =  0.11704592620593783 at epoch  4000
Cost =  0.11205470319914505 at epoch  4500
Cost =  0.10778404062490321 at epoch  5000
Cost =  0.10406686941551509 at epoch  5500
Cost =  0.10078706993627766 at epoch  6000
Cost =  0.09786095486433724 at epoch  6500
Cost =  0.09522634659131059 at epoch  7000
Cost =  0.09283589400966233 at epoch  7500
Cost =  0.09065277318198824 at epoch  8000
```

```
Cost =  0.08864780065633096 at epoch  8500
Cost =  0.08679744633296001 at epoch  9000
Cost =  0.08508244598020351 at epoch  9500
Cost =  0.08348681929148014 at epoch  10000
Cost =  2.1182936721476553 at epoch  0
Cost =  0.27998292388249524 at epoch  500
Cost =  0.20307103433352286 at epoch  1000
Cost =  0.17104236086479957 at epoch  1500
Cost =  0.1523633175459273 at epoch  2000
Cost =  0.1397022359518514 at epoch  2500
Cost =  0.13033701063250494 at epoch  3000
Cost =  0.1230082450100857 at epoch  3500
Cost =  0.11704592620593783 at epoch  4000
Cost =  0.11205470319914505 at epoch  4500
Cost =  0.10778404062490321 at epoch  5000
Cost =  0.10406686941551509 at epoch  5500
Cost =  0.10078706993627766 at epoch  6000
Cost =  0.09786095486433724 at epoch  6500
Cost =  0.09522634659131059 at epoch  7000
Cost =  0.09283589400966233 at epoch  7500
Cost =  0.09065277318198824 at epoch  8000
Cost =  0.08864780065633096 at epoch  8500
Cost =  0.08679744633296001 at epoch  9000
Cost =  0.08508244598020351 at epoch  9500
Cost =  0.08348681929148014 at epoch  10000
Cost =  0.08199716189054078 at epoch  10500
Cost =  0.08060212276116276 at epoch  11000
Cost =  0.07929200940329773 at epoch  11500
Cost =  0.07805848371783727 at epoch  12000
Cost =  0.07689432438626831 at epoch  12500
Cost =  0.07579323908763191 at epoch  13000
Cost =  0.07474971456184852 at epoch  13500
Cost =  0.07375889565799124 at epoch  14000
Cost =  0.07281648676737945 at epoch  14500
Cost =  0.07191867072993789 at epoch  15000
Cost =  0.0710620415604673 at epoch  15500
Cost =  0.07024354826138748 at epoch  16000
Cost =  0.06946044764749522 at epoch  16500
Cost =  0.06871026457507187 at epoch  17000
Cost =  0.06799075830004488 at epoch  17500
Cost =  0.06729989393229985 at epoch  18000
Cost =  0.06663581813682132 at epoch  18500
Cost =  0.06599683837725842 at epoch  19000
Cost =  0.06538140511582786 at epoch  19500
Cost =  0.06478809648212798 at epoch  20000
Cost =  0.06421560500643417 at epoch  20500
Cost =  0.0636627260828897 at epoch  21000
Cost =  0.06312834788654638 at epoch  21500
```

```
Cost =  0.06261144251696187 at epoch  22000
Cost =  0.06211105818142225 at epoch  22500
Cost =  0.06162631226407052 at epoch  23000
Cost =  0.061156385154413365 at epoch  23500
Cost =  0.060700514730863554 at epoch  24000
Cost =  0.06025799141300995 at epoch  24500
Cost =  0.059828153710953694 at epoch  25000
Cost =  0.05941038421192426 at epoch  25500
Cost =  0.0590041059540067 at epoch  26000
Cost =  0.05860877914459871 at epoch  26500
Cost =  0.05822389818747583 at epoch  27000
Cost =  0.0578489889873679 at epoch  27500
Cost =  0.05748360650492249 at epoch  28000
Cost =  0.05712733253805903 at epoch  28500
Cost =  0.05677977370816147 at epoch  29000
Cost =  0.05644055963147156 at epoch  29500
Cost =  0.05610934125760294 at epoch  30000
Cost =  0.05578578935841618 at epoch  30500
Cost =  0.05546959315170966 at epoch  31000
Cost =  0.0551604590453675 at epoch  31500
Cost =  0.054858109488814685 at epoch  32000
Cost =  0.05456228191987569 at epoch  32500
Cost =  0.054272727796402555 at epoch  33000
Cost =  0.05398921170329844 at epoch  33500
Cost =  0.053711510526779846 at epoch  34000
Cost =  0.05343941268884652 at epoch  34500
Cost =  0.053172717435953 at epoch  35000
Cost =  0.05291123417676127 at epoch  35500
Cost =  0.052654781864613545 at epoch  36000
Cost =  0.05240318842099132 at epoch  36500
Cost =  0.05215629019674043 at epoch  37000
Cost =  0.05191393146824835 at epoch  37500
Cost =  0.0516759639660939 at epoch  38000
Cost =  0.05144224643394315 at epoch  38500
Cost =  0.05121264421568078 at epoch  39000
Cost =  0.05098702886893545 at epoch  39500
Cost =  0.050765277803300216 at epoch  40000
Cost =  0.050547273941673486 at epoch  40500
Cost =  0.050332905403250736 at epoch  41000
Cost =  0.050122065206799 at epoch  41500
Cost =  0.04991465099292798 at epoch  42000
Cost =  0.04971056476415907 at epoch  42500
Cost =  0.04950971264166353 at epoch  43000
Cost =  0.049312004637616216 at epoch  43500
Cost =  0.049117354442171454 at epoch  44000
Cost =  0.04892567922413048 at epoch  44500
Cost =  0.04873689944442455 at epoch  45000
Cost =  0.04855093868158693 at epoch  45500
```

```
Cost =  0.04836772346843875 at epoch  46000
Cost =  0.04818718313925023 at epoch  46500
Cost =  0.04800924968668656 at epoch  47000
Cost =  0.04783385762788032 at epoch  47500
Cost =  0.04766094387901065 at epoch  48000
Cost =  0.047490447637803294 at epoch  48500
Cost =  0.04732310273397776 at epoch  49000
Cost =  0.04715647522306051 at epoch  49500
Cost =  0.046992887895254555 at epoch  50000
Cost =  2.1182936721476553 at epoch  0
Cost =  0.6425182246590546 at epoch  500
Cost =  0.45780883553476787 at epoch  1000
Cost =  0.36604690533082984 at epoch  1500
Cost =  0.31374149726180545 at epoch  2000
Cost =  0.2800565789265448 at epoch  2500
Cost =  0.25629042694719756 at epoch  3000
Cost =  0.23838421930333062 at epoch  3500
Cost =  0.22425527033156967 at epoch  4000
Cost =  0.21272793082639768 at epoch  4500
Cost =  0.20308520646571906 at epoch  5000
Cost =  0.19486334613296252 at epoch  5500
Cost =  0.18774634309607824 at epoch  6000
Cost =  0.18150864343165932 at epoch  6500
Cost =  0.17598343456684834 at epoch  7000
Cost =  0.17104432195090086 at epoch  7500
Cost =  0.16659378014092457 at epoch  8000
Cost =  0.16255537259172503 at epoch  8500
Cost =  0.15886831611193955 at epoch  9000
Cost =  0.1554835999313468 at epoch  9500
Cost =  0.15236120382329682 at epoch  10000
Cost =  0.14946809207953493 at epoch  10500
Cost =  0.14677672892021656 at epoch  11000
Cost =  0.14426395517835408 at epoch  11500
Cost =  0.14191013689488133 at epoch  12000
Cost =  0.13969851811094794 at epoch  12500
Cost =  0.13761471801394515 at epoch  13000
Cost =  0.13564632861065215 at epoch  13500
Cost =  0.13378258677373023 at epoch  14000
Cost =  0.1320141056052944 at epoch  14500
Cost =  0.13033265475533948 at epoch  15000
Cost =  0.12873098104956815 at epoch  15500
Cost =  0.12720266183475287 at epoch  16000
Cost =  0.12574198457621596 at epoch  16500
Cost =  0.12434384743102653 at epoch  17000
Cost =  0.1230036766165854 at epoch  17500
Cost =  0.12171735730585889 at epoch  18000
Cost =  0.12048117549374653 at epoch  18500
Cost =  0.11929176882201169 at epoch  19000
```

```
Cost =  0.11814608476220453 at epoch  19500
Cost =  0.1170413448721461 at epoch  20000
Cost =  0.11597501408813278 at epoch  20500
Cost =  0.11494477421016265 at epoch  21000
Cost =  0.11394850089358974 at epoch  21500
Cost =  0.11298424358629469 at epoch  22000
Cost =  0.11205020795215673 at epoch  22500
Cost =  0.11114474040421111 at epoch  23000
Cost =  0.11026631443818313 at epoch  23500
Cost =  0.10941351851199269 at epoch  24000
Cost =  0.10858504526156557 at epoch  24500
Cost =  0.10777968187959339 at epoch  25000
Cost =  0.1069963015131153 at epoch  25500
Cost =  0.1062338555591285 at epoch  26000
Cost =  0.10549136675591038 at epoch  26500
Cost =  0.10476792298223123 at epoch  27000
Cost =  0.1040626716880412 at epoch  27500
Cost =  0.10337481488923414 at epoch  28000
Cost =  0.10270360466636763 at epoch  28500
Cost =  0.10204833911327867 at epoch  29000
Cost =  0.10140835868677107 at epoch  29500
Cost =  0.10078304291325184 at epoch  30000
Cost =  0.10017180741252871 at epoch  30500
Cost =  0.09957410120307203 at epoch  31000
Cost =  0.09898940425689177 at epoch  31500
Cost =  0.09841722527581259 at epoch  32000
Cost =  0.09785709966429947 at epoch  32500
Cost =  0.09730858767708281 at epoch  33000
Cost =  0.0967712727226265 at epoch  33500
Cost =  0.09624475980595963 at epoch  34000
Cost =  0.09572867409657199 at epoch  34500
Cost =  0.09522265960895707 at epoch  35000
Cost =  0.09472637798499325 at epoch  35500
Cost =  0.0942395073687324 at epoch  36000
Cost =  0.09376174136531926 at epoch  36500
Cost =  0.09329278807674635 at epoch  37000
Cost =  0.09283236920797565 at epoch  37500
Cost =  0.09238021923765997 at epoch  38000
Cost =  0.09193608464828511 at epoch  38500
Cost =  0.09149972321106839 at epoch  39000
Cost =  0.09107090332138318 at epoch  39500
Cost =  0.09064940338085321 at epoch  40000
Cost =  0.09023501122259872 at epoch  40500
Cost =  0.08982752357639552 at epoch  41000
Cost =  0.08942674557077243 at epoch  41500
Cost =  0.0890324902692976 at epoch  42000
Cost =  0.08864457823850709 at epoch  42500
Cost =  0.08826283714511937 at epoch  43000
```

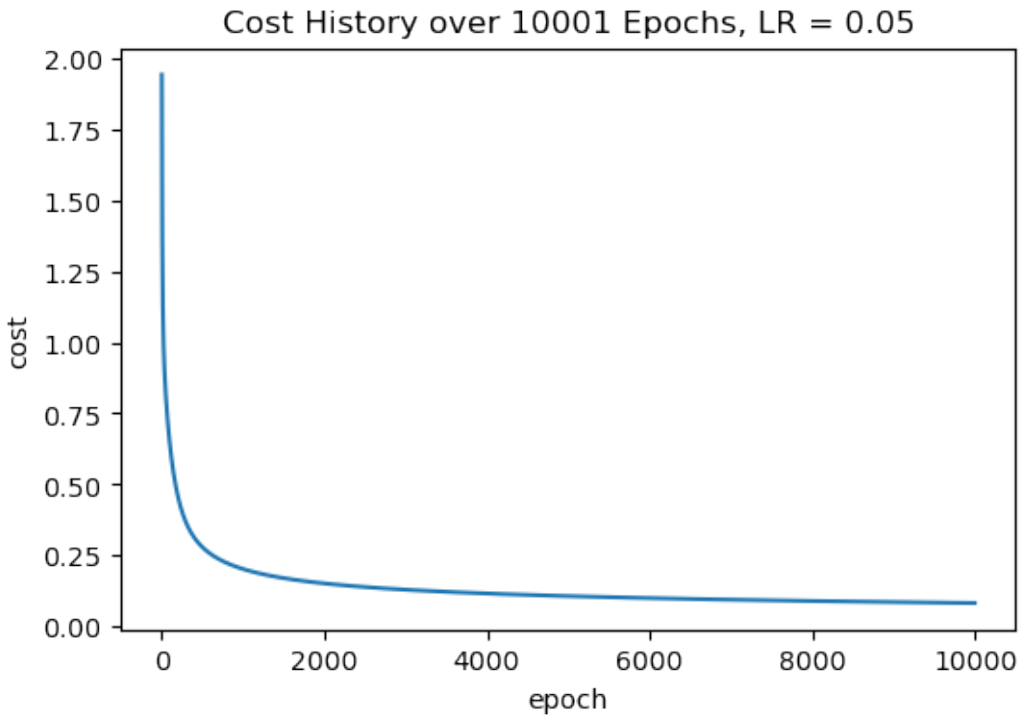
```
Cost =  0.08788710138034621 at epoch  43500
Cost =  0.08751721170926555 at epoch  44000
Cost =  0.08715301494336802 at epoch  44500
Cost =  0.08679436363452313 at epoch  45000
Cost =  0.08644111578873956 at epoch  45500
Cost =  0.08609313459820822 at epoch  46000
Cost =  0.0857502881902357 at epoch  46500
Cost =  0.08541244939177844 at epoch  47000
Cost =  0.08507949550838974 at epoch  47500
Cost =  0.08475130811648801 at epoch  48000
Cost =  0.0844277728679423 at epoch  48500
Cost =  0.08410877930605673 at epoch  49000
Cost =  0.0837942206921122 at epoch  49500
Cost =  0.08348399384169977 at epoch  50000
```

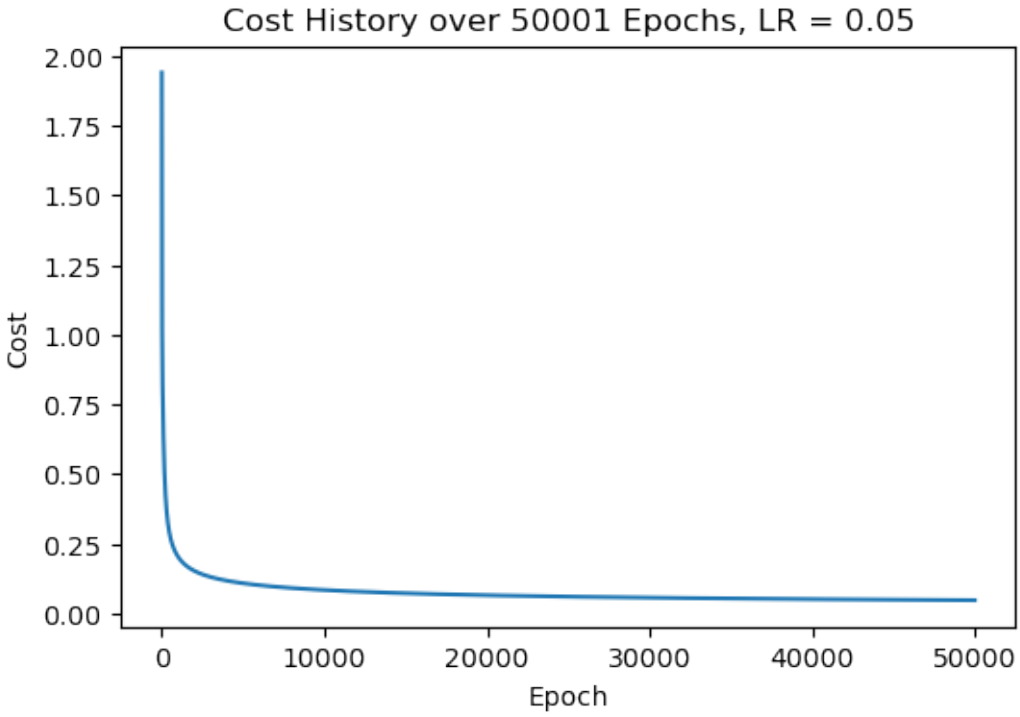
0.2.5 4. Training and testing

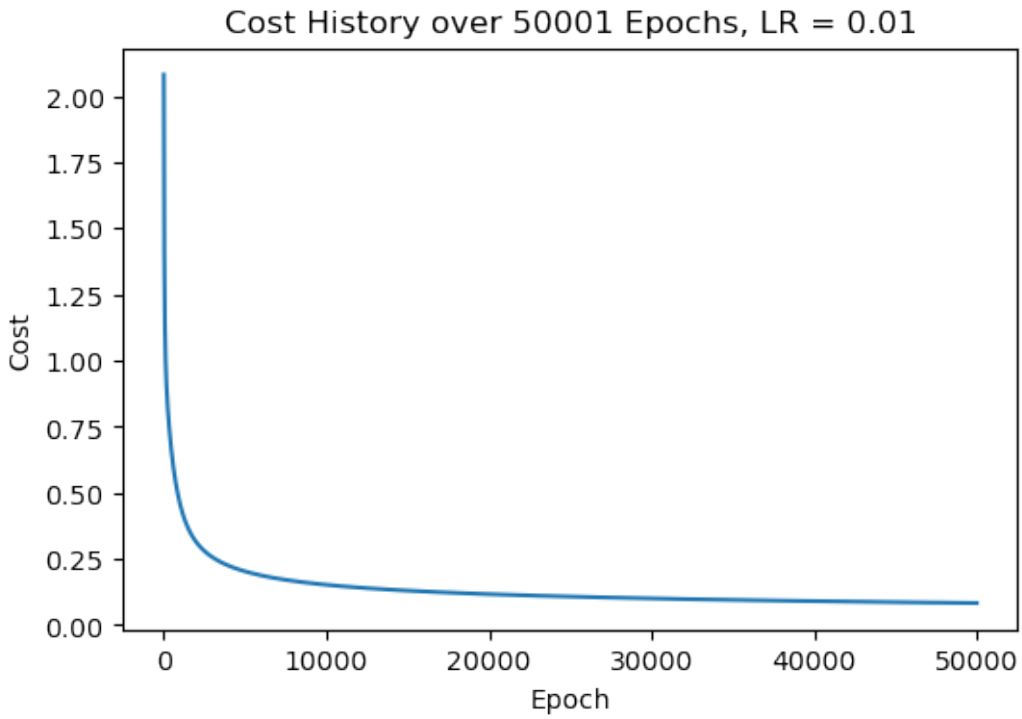
a. Cost history

[295]:
```
plt.figure()

#print(np.shape(chist1))
plt.plot(chist1[11:])
plt.title('Cost History over 10001 Epochs, LR = 0.05')
plt.xlabel('epoch')
plt.ylabel('cost')


plt.figure()
plt.plot(chist2[11:])
plt.title('Cost History over 50001 Epochs, LR = 0.05')
plt.xlabel('Epoch')
plt.ylabel('Cost')

plt.figure()
plt.plot(chist3[11:])
plt.title('Cost History over 50001 Epochs, LR = 0.01')
plt.xlabel('Epoch')
plt.ylabel('Cost')
```

[295]:
```
Text(0, 0.5, 'Cost')
```


Cost History over 10001 Epochs, LR = 0.05
cost
epoch
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2000
4000
6000
8000
10000


Cost History over 50001 Epochs, LR = 0.05
Cost
Epoch
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
10000
20000
30000
40000
50000



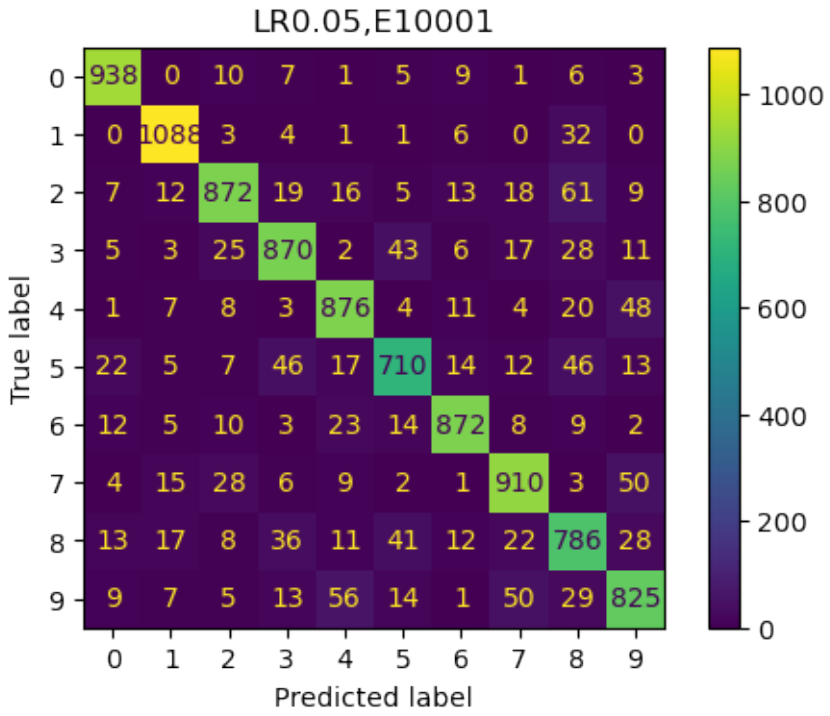
b. Confusion matrix

[142]:
```
plt.rcParams['figure.dpi'] = 100
sess = tf.Session()
saver.restore(sess, saved1)
ytestout = sess.run(out, {X:test_data_norm})
ytestout = np.argmax(ytestout, axis =0)
cm1 = confusion_matrix(np.argmax(test_int, axis=0), ytestout)
plt.figure()

ConfusionMatrixDisplay(confusion_matrix = cm1).plot()
plt.title('LR0.05,E10001')


sess.close()


sess = tf.Session()
saver.restore(sess, saved2)
ytestout = sess.run(out, {X:test_data_norm})
ytestout = np.argmax(ytestout, axis =0)
cm2 = confusion_matrix(np.argmax(test_int, axis=0), ytestout)
plt.figure()
```

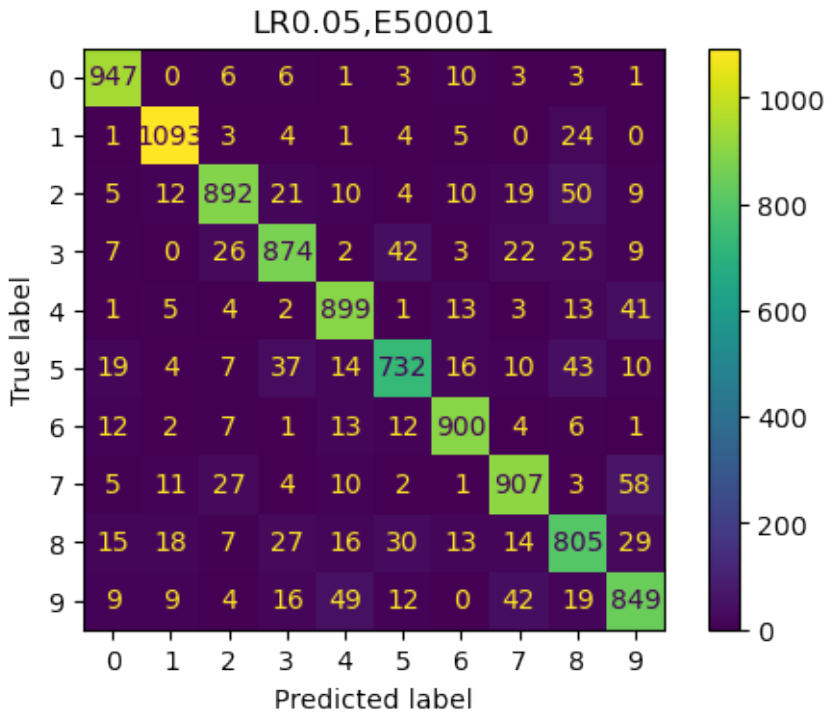
```
ConfusionMatrixDisplay(confusion_matrix = cm2).plot()
plt.title('LR0.05,E50001')
sess.close()

sess = tf.Session()
saver.restore(sess, saved3)
ytestout = sess.run(out, {X:test_data_norm})
ytestout = np.argmax(ytestout, axis =0)
cm3 = confusion_matrix(np.argmax(test_int, axis=0), ytestout)
plt.figure()

ConfusionMatrixDisplay(confusion_matrix = cm3).plot()
plt.title('LR0.01')
sess.close()
```

```
INFO:tensorflow:Restoring parameters from trained_model_0.05_10001.ckpt
INFO:tensorflow:Restoring parameters from trained_model_0.05_50001.ckpt
INFO:tensorflow:Restoring parameters from trained_model_0.01_50001.ckpt
```

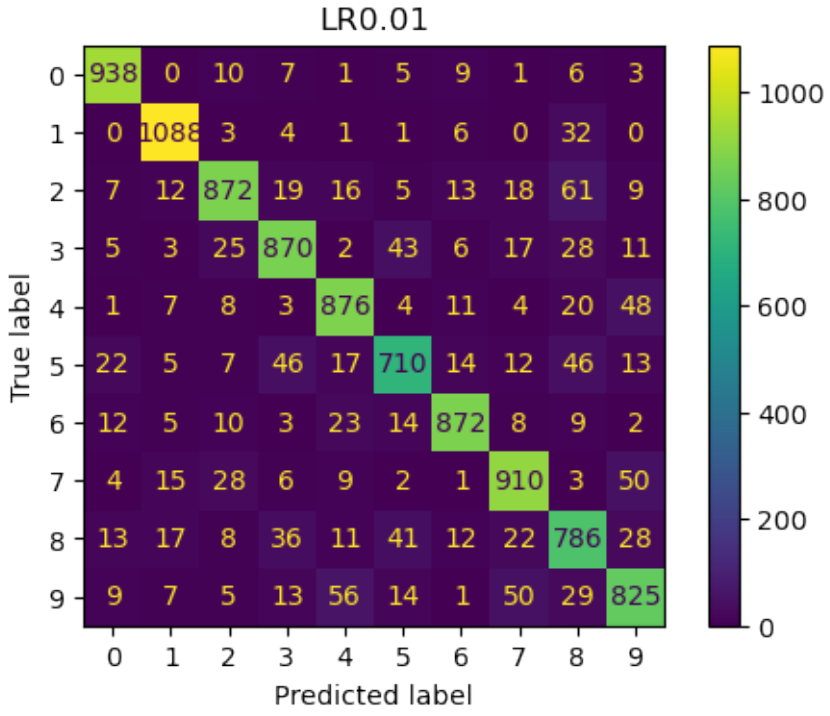
```
<Figure size 600x400 with 0 Axes>
```




```
<Figure size 600x400 with 0 Axes>
```

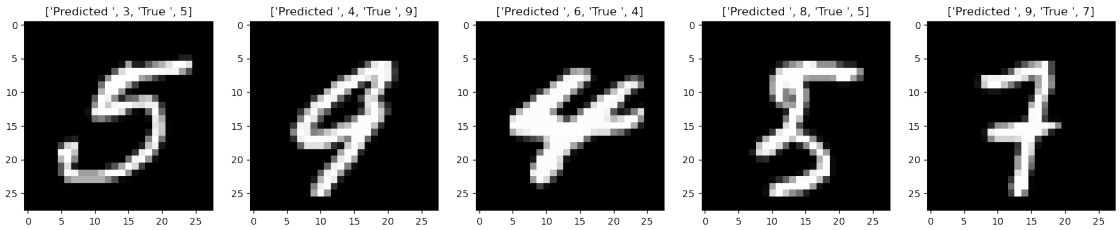


<Figure size 600x400 with 0 Axes>



c. Common misclassifications

[293]:
```
cases = [219, 7580 , 6759, 3776, 307]
plt.figure(figsize = (20,20))
for i in range(5):
    plt.subplot(1,5,i+1)
    plt.imshow(test_data[cases[i]], cmap = 'gray')
    plt.title(['Predicted ', ytestout[cases[i]], 'True ', test_labels[cases[i]]])
```



[294]:
```
print('9 and 4 can be easily misclassfied as the 9 is not always connected at␣
 ↪the top, thus looking like a 4.')
print('5 and 3 are written similarly, however if the upper stem on the five is␣
 ↪close to the rest of the number, it can begin to look a little like a␣
 ↪compacted 3.')
print('6 and 4 are most likely for a similar reason as to nine, however if we␣
 ↪trace a 6 the bottom part, if done crudely, can look like the line going␣
 ↪through a 4.')
print('8 and 5: a very compacted 5 nearly can resemble two circles on top of␣
 ↪each other, thus becoming an 8.')
print('9 and 7, if 7 is written with a horizontal line through it and possible␣
 ↪and extra vertical line from the top, it resembles a 9.')
print('Ultimately, handwriting styles are unique to a person, and depending on␣
 ↪how one chooses to write many numbers can have very similar features to␣
 ↪others, in this case we see')
print('5 and 3, 9 and 4, 8 and 5, 9 and 7')
```

```
9 and 4 can be easily misclassfied as the 9 is not always connected at the top,
thus looking like a 4.
5 and 3 are written similarly, however if the upper stem on the five is close to
the rest of the number, it can begin to look a little like a compacted 3.
6 and 4 are most likely for a similar reason as to nine, however if we trace a 6
the bottom part, if done crudely, can look like the line going through a 4.
8 and 5: a very compacted 5 nearly can resemble two circles on top of each
other, thus becoming an 8.
9 and 7, if 7 is written with a horizontal line through it and possible and
```

extra vertical line from the top, it resembles a 9.
Ultimately, handwriting styles are unique to a person, and depending on how one chooses to write many numbers can have very similar features to others, in this case we see
5 and 3, 9 and 4, 8 and 5, 9 and 7

[ ]:

[ ]: