```
import os
import numpy as np
import random
import matplotlib.pyplot as plt
import image_resources as ir
from decimal import *

# Functions that might be useful (please read the documentation)
# x.flatten() (take a N-dimensional numpy array and make it one-
dimensional)
# numpy.random.choice -- choose from the list of images
# numpy.dot -- compute the dot product
# numpy.random.normal -- set up random initial weights

DIM = (28,28) #these are the dimensions of the image

def load_image_files(n, path="images/"):
    # helper file to help load the images
    # returns a list of numpy vectors
    images = []
    for f in os.listdir(os.path.join(path,str(n))): # read files in
the path
        p = os.path.join(path,str(n),f)
        if os.path.isfile(p):
            i = np.loadtxt(p)
            assert i.shape == DIM # just check the dimensions here
            # i is loaded as a matrix, but we are going to flatten it
into a single vector
            images.append(i.flatten())
    return images

## Your code here:
def saveImages(files, show=False):
    """Saves digit image matrices for faster access during training"""

    for j in range(len(files)):
        np.save(files[j], load_image_files(j))
        if show:
            print(files[j] + '.npy saved')

def loadImages(files, show=False, unseen=False):
    """Returns a dictionary of image files with digits as keys"""
    trainingDict = dict()
    unseenDict = dict()

    for f in range(len(files)):
        trainingDict[f] = np.load(files[f] + '.npy')
        if show:
            print(files[f] + '.npy loaded')
```

```
    if unseen:
        print('Creating a dictionary of test images, 500 for each 
digit...')
        for x in range(len(trainingDict)):
            unseenDict[x] = np.empty((500, 784))

            for i in range(500):
                randomIndex = random.randint(0, len(trainingDict[x]) - 
1)
                unseenDict[x][i] = trainingDict[x][randomIndex]
                trainingDict[x] = np.delete(trainingDict[x], 
randomIndex, 0)
            print('...')
    print('Test dictionary complete')


    return trainingDict, unseenDict

fileNames = ['Zero', 'One', 'Two', 'Three', 'Four', 'Five', 'Six', 
'Seven', 'Eight', 'Nine']

trainingImages, unseenImages = loadImages(fileNames, unseen=True)
N = len(trainingImages[0][0])
assert N == DIM[0]*DIM[1] # just check our sizes to be sure

"""Problem 1: Write an implementation of the perceptron learning 
algorithm that first
    loads images for the digit “0” and then for the digit “1”. Start 
with random weights
    from a normal distribution. Compute the average accuracy on blocks 
of 25 items and plot
    this accuracy until you think it won’t get better.

    ANSWER: The answer to how many images it takes to train my model 
varies depending
        on the random weights I start with but it usually takes around 
400 blocks to
        achieve achieve .99 accuracy. This is about 10,000 images 
between the 0 and 1
        training sets. I attempted to get .9999 accuracy but it takes 
10x longer
        and doesn't seem to produce significantly better results. """

class Perceptron(object):

    def __init__(self, dimensions, data, digit0, digit1):
        self.weights = np.random.normal(0, 1, size=dimensions)
        self.data_set = {0: data[digit0], 1: data[digit1]}
        self.digits = [digit0, digit1]
        self.bias = np.zeros(dimensions)
```

```
        self.overall_accuracy = 0

    def get_weights(self):
        return self.weights

    def dotProd(self, w, x):

        return np.dot(w, x)

    def predict(self, w, x, b):
        dot = self.dotProd(w, x)

        if dot >= 0:
            return 1

        else: return 0

    def classify(self, w, x):
        dot = self.dotProd(w, x)

        if dot >= 0:
            return 1
        else: return 0

    def random_label(self):
        """Returns a random dataset label"""
        labels = list(self.data_set.keys())
        return random.choice(labels)

    def plot_accuracy(self, x, y, save=False, name=None, color='red'):

        plt.xlabel('Blocks of 25')
        plt.ylabel('Accuracy')
        plt.title('P1: Training Accuracy for Digits {0} and
{1}'.format(self.digits[0], self.digits[1]))
        plt.legend(loc=0, title='Converged to {0} Accuracy in {1}
Blocks'.format(round(y[-1], 5), x[-1]))
        plt.plot(x, y, c=color, label='')

        if save:
            plt.savefig(name + '.pdf')

        plt.show()
        plt.close()

    def train(self, threshold, precision, blocks, plot=False):
        """Data_set is a dictionary of lists containing (784,)
arrays"""
        accuracy = .5
        all_blocks = blocks
```

```
        correct = 0
        accuracy_trace = []
        xAxis = []
        acc_delta = 1
        getcontext().prec = precision

        while (all_blocks // 25) <= 500:

            xAxis += [all_blocks // 25]
            acc_delta = accuracy

            for i in range(blocks):
                label = self.random_label()
                data = self.data_set[label]
                sub_data = random.choice(data)
                y = self.predict(self.weights, sub_data, self.bias)

                if label == 0 and y == 1:
                    self.bias -= sub_data
                    self.weights -= sub_data - self.bias
                elif label == 1 and y == 0:
                    self.bias += sub_data
                    self.weights += sub_data + self.bias
                else:
                    correct += 1

            accuracy = correct / all_blocks
            acc_delta = abs(round(Decimal(accuracy), precision) - 
round(Decimal(acc_delta), precision))
            all_blocks += blocks
            accuracy_trace.append(accuracy)

        self.overall_accuracy = accuracy

        if plot:
            self.plot_accuracy(xAxis, accuracy_trace)

# zeroOnePercept = Perceptron(N, trainingImages, 0, 1)
# zeroOnePercept.train(.97, 5, 25, plot=True)
#
# np.save('zero_one_weights', zeroOnePercept.get_weights())

"""Problem 2: Does your solution in Q1 converge on 100% accuracy or 
not? What does this
    mean in terms of the linear separability of “0” and “1” on this 
feature space?

    ANSWER: My solution for Q1 approaches 100% (I used the method of 
total correct
    /total images) which reaches .9999 for digits 0 and 1. This tells
```

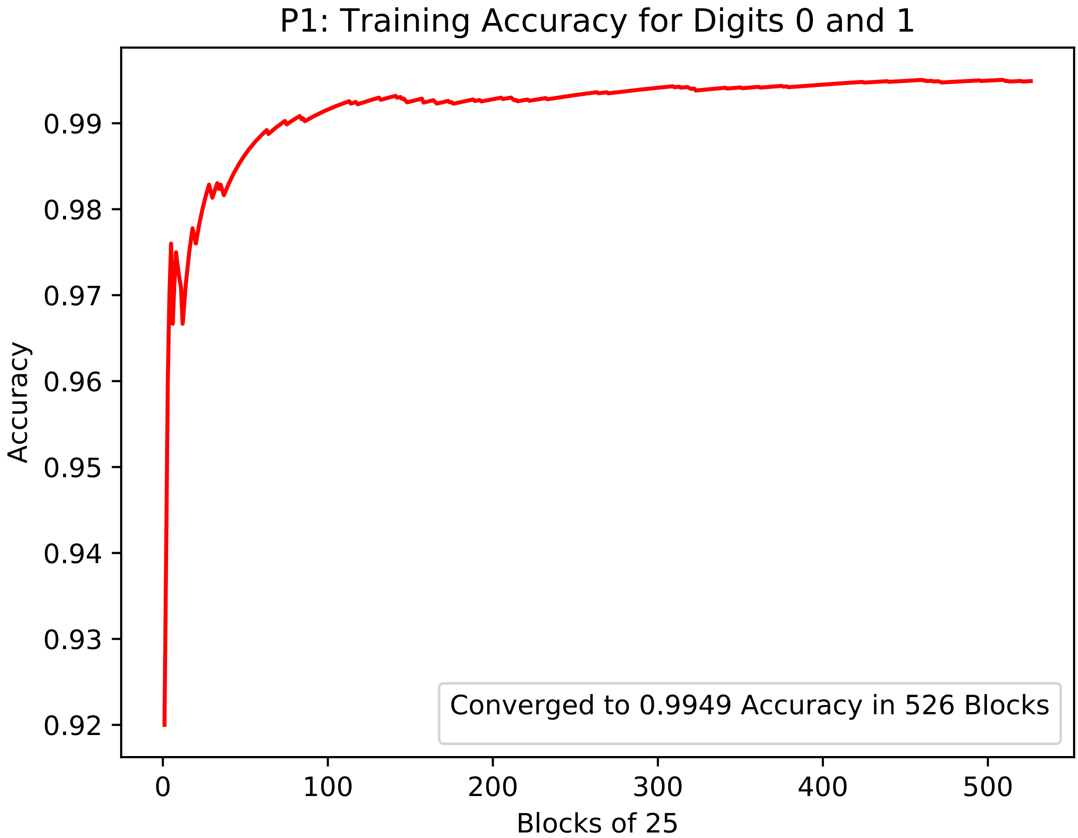
```
me that 0 and 1
    are completely linearly separable. The pair have unique features
that allow for
    distinguishing them completely. """

"""Problem 3: What do large negative and large positive values mean,
intuitively? What
    do numbers near zero mean? Why does this matrix look the way that
it does, in terms
    of where large positive and negative terms are located?

    ANSWER: The large positive and negative values seem to occur where
the digits were
    in the training images. For my perceptron, 1 took very large
values while 0 took
    very negative values. In my matrix, the general shape of a 0 can
be seen in black
    which is the color for large negative values and bright yellow can
be seen in the center
    of the 0 where a 1 would be if the digits overlapped. There is
also yellow outside the
    black 0 shape and I take this to represent some of the 1 images
that were written in
    a 'forward slash' shape. Numbers near 0 represent the 0's (as
opposed to the 1's) from
    the training images where the weights were updated but the feature
values in that area
    was 0 so the weights stayed small."""

def weightMatrix(weights, dims, save=False,
fileName='FILENAME',method=None, bounds=[0, 1]):
    # weights= weights[10]
    weights = weights.reshape(dims, dims)
    fig, im = plt.subplots(figsize=(10, 10)) #
    wm = plt.imshow(weights) #, vmin=bounds[0], vmax=bounds[1]
    #cmap=plt.get_cmap(ir.color_maps[ir.color_maps.index('inferno')]),
interpolation=ir.interpol_methods[method],
    wm.axes.get_xaxis().set_visible(False)
    wm.axes.get_yaxis().set_visible(False)
    plt.title('Heat Map of Trained Weight Matrix For Digits 0 and 1',
fontsize=15)
    fig.colorbar(wm, orientation='horizontal', fraction=.0415)
    if save:
        plt.savefig(fileName + '.pdf')

    plt.show()

# boundList = [0, 1]
# weightMatrix(np.load('zero_one_weights.npy'), 28, save=True,
fileName='a7p3_norm_bias', method=0, bounds=boundList)
```

```
"""Problem 4: What should you expect to happen if you set the elements
of the weight vector
    which are close to zero to be actually zero? Do this for the 10,
20, 30, ... 780 weight
    values closest to zero (in absolute value) and plot the resulting
accuracies on 1000 random
    classifications of “0” vs “1”. What does this tell you about the
proportion of the image
    which is diagnostic about “0” vs “1”?

    ANSWER: Setting the weights 'close' to 0 should negate the effect
of those elements; they
    shouldn't contribute to the classification of the digits. My plot
showed near 100% classification
    accuracy up to about 120 elements set to 0 and then the accuracy
falls to and hovers around 50%.
    When the accuracy falls to 50% the perceptron is essentially
guessing as it doesn't have enough
    feature data to know which digit is which.

    Seeing that about 121 elements of the weight array can essentially
be ignored tells me that
    a low proportion of the image is diagnostic about 0 vs 1
especially when you consider
    that a large portion of the weights were not effected by training
in the first place.
    I created a matrix of the weight array using the last version with
~98% accuracy and
    it contained only 4 or 5 elements in the area of the digits but it
was enough for linear
    separation.
    """

def testClassification(numIters, digit0, digit1, perceptron,
unseenDict, weights):
    numCorrect = 0

    for i in range(numIters):
        label = random.choice([digit0, digit1])
        result = perceptron.classify(weights, unseenDict[label][i %
(numIters // 2)])

        if label == digit0 and result == 0:
            numCorrect += 1
            label = digit1
        elif label == digit1 and result == 1:
            numCorrect += 1
            label = digit0
```

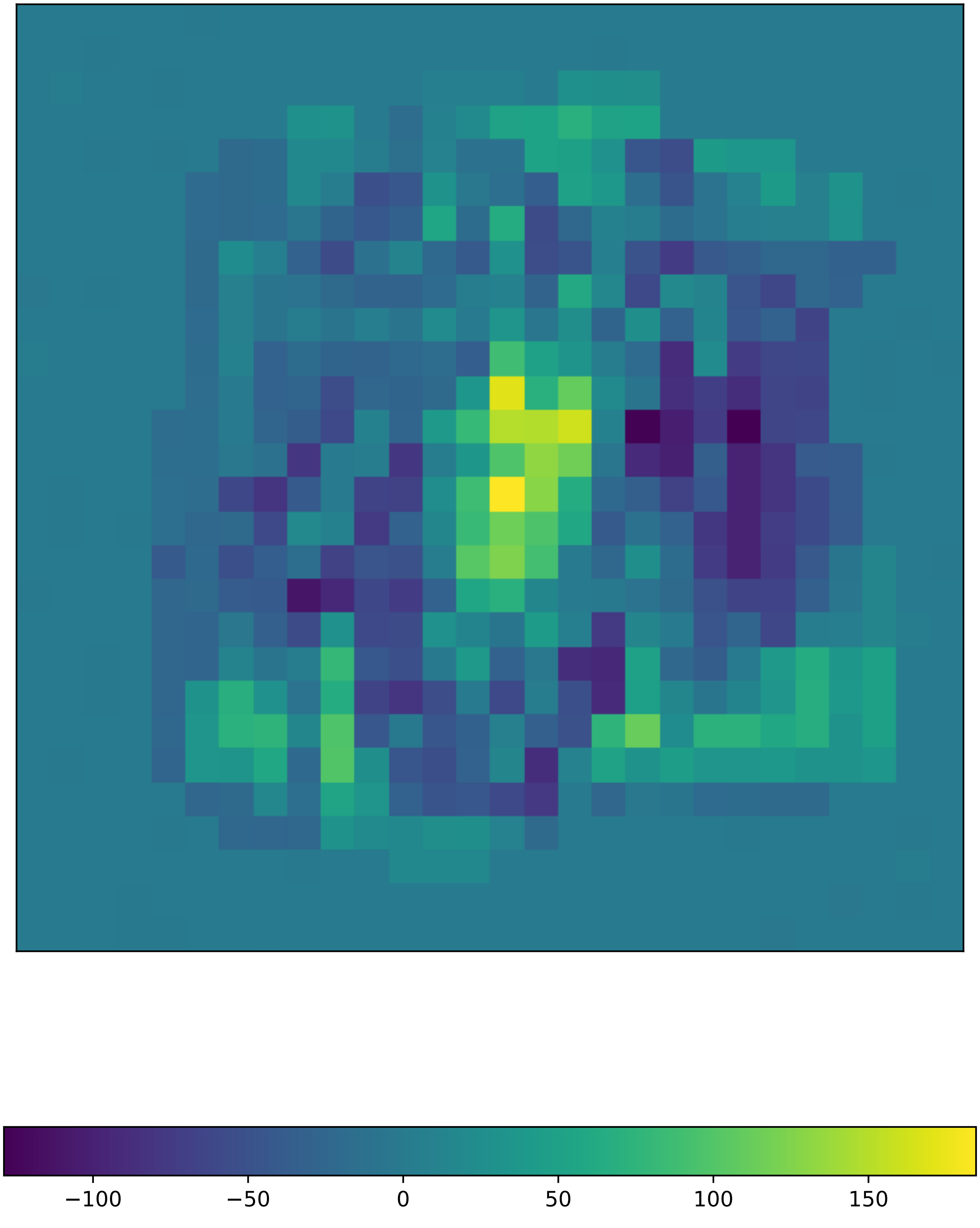
```
    return numCorrect

def setToZero(trainDict, unseenDict, N, digit0, digit1):
    accuracyList = []
    weightsList = []
    xList = [10 * i for i in range(1, 79)]
    perceptron01 = Perceptron(N, trainDict, digit0, digit1)
    perceptron01.train(.97, 5, 25)
    modWeights = np.copy(perceptron01.get_weights())
    modWeights = np.absolute(modWeights) #convert all to positive
values
    modWeights[modWeights == 0] = 1000 #make all 0's into 1000, if any

    for x in range(1, 79):
        k = 0

        while k < (len(range(x * 10))):
            index = np.argmin(modWeights)
            perceptron01.weights[index] = 0
            modWeights[index] = 1000
            k += 1

        result = testClassification(1000, digit0, digit1,
perceptron01, unseenDict, perceptron01.weights)
        accuracyList += [result / 1000]
        weightsList += [np.copy(perceptron01.weights.reshape(28, 28))]

    return accuracyList, xList, weightsList

# accList, xList, moddedWeights = setToZero(trainingImages,
unseenImages, N, 0, 1)
# plt.plot(xList, accList)
# plt.title('Modified Weight Values at Intervals 10, 20, 30, ...,
780')
# plt.ylabel('Average Accuracy')
# plt.xlabel('Weight Values Changed to 0 (smallest first)')
# plt.savefig('a7p4.pdf')
# plt.show()
# plt.close()
#
# plt.imshow(moddedWeights[10])
# plt.savefig('mod_at_11.pdf')
# plt.close()
# plt.imshow(moddedWeights[13])
# plt.savefig('mod_at_13.pdf')
# plt.close()


"""Problem 5: Next show a matrix of the classification accuracy of
each pair of digits
```

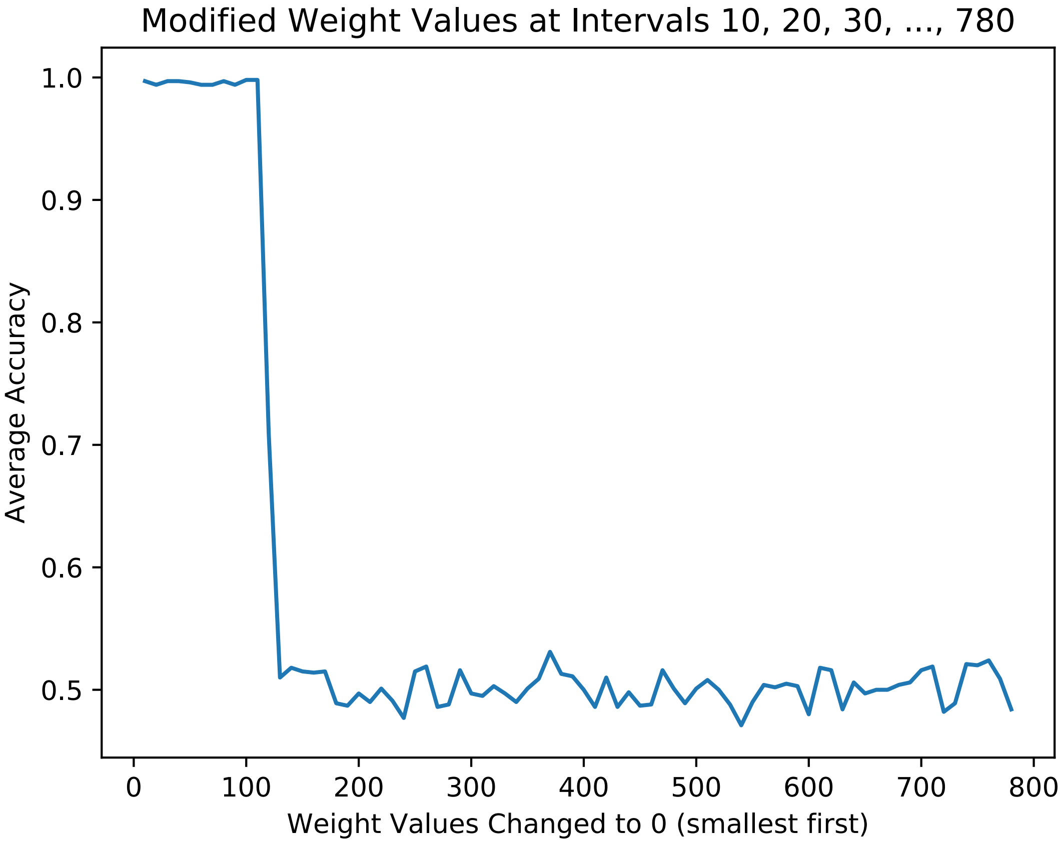
```
    after enough training. Make this a plot (with colors for accuracy
rather than numbers).
    Does it match your intuitions about which pairs should be easy vs.
hard? Why or why not?

    ANSWER: The matrix does approximately match my intuition. All
digits compared with
    themselves (0 and 0 for example) have 100% accuracy. The
perceptron is 96 - 100%
    accurate for most pairs. I would have expected (4, 9), (3, 5), and
(5, 8) to be harder
    to linearly separate due to the amount of overlap these digits
tend to have in handwriting
    and my matrix reflects this intuition with 90-92% accuracy for
these digit pairs.
    """

def allDigitsAcc(trainDict, unseenDict, N):
    digitArray = np.zeros(100)
    digitArray = digitArray.reshape(10, 10)
    xlabels = np.arange(0, 10)
    ylabels = np.arange(0, 10)

    for k in range(len(trainDict.items())):
        for j in range(len(trainDict.items())):
            correct = 0
            percept = Perceptron(N, trainDict, k, j)
            percept.train(.97, 5, 25)
            result = testClassification(1000, k, j, percept,
unseenDict, percept.get_weights())
            digitArray[k, j] = result / 1000

    return digitArray, xlabels, ylabels

#uncomment below to run code for Problem 5
# grid, xAx, yAx = allDigitsAcc(trainingImages, unseenImages, N)
# np.save('accMatrix', grid)
# np.save('accMatrix_X', xAx)
# np.save('accMatrix_Y', yAx)
#
grid = np.load('accMatrix.npy')
xAx = np.load('accMatrix_X.npy')
yAx = np.load('accMatrix_Y.npy')

plt.rcParams['xtick.bottom'] = plt.rcParams['xtick.labelbottom'] =
False
plt.rcParams['xtick.top'] = plt.rcParams['xtick.labeltop'] = True

fig, ax = plt.subplots() #figsize=(10,10)
matrix = plt.imshow(grid, cmap='inferno', extent=[0, 9, 9, 0])
```

```
#'coolwarm'
figure_title = 'Classification Accuracy For Each Pair of Digits'
plt.text(0.5, 1.13, figure_title,
         horizontalalignment='center',
         fontsize=12,
         transform = ax.transAxes)
fig.colorbar(matrix, orientation='horizontal', fraction=.05)
ax.set_xticks(xAx)
ax.set_yticks(yAx)

plt.savefig('a7p5accuracy_4.pdf')

plt.show()


#end
```

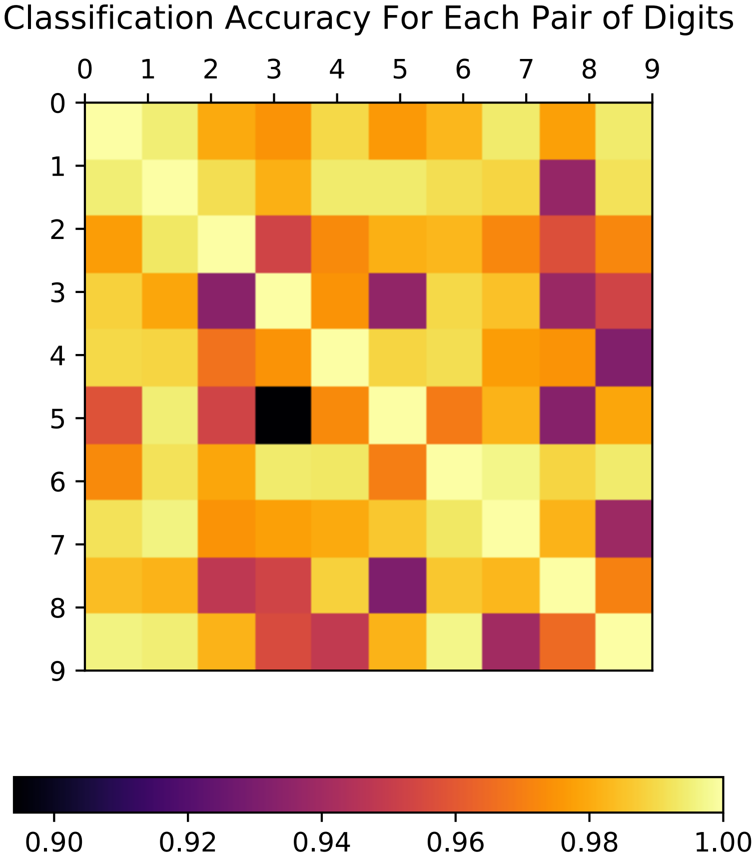

P1: Training Accuracy for Digits 0 and 1
Accuracy
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0
100
200
300
400
500
Blocks of 25
Converged to 0.9949 Accuracy in 526 Blocks

Heat Map of Trained Weight Matrix For Digits 0 and 1





Modified Weight Values at Intervals 10, 20, 30, ..., 780
Average Accuracy
Weight Values Changed to 0 (smallest first)
1.0
0.9
0.8
0.7
0.6
0.5
0
100
200
300
400
500
600
700
800


Classification Accuracy For Each Pair of Digits
0
1
2
3
4
5
6
7
8
9
0.90
0.92
0.94
0.96
0.98
1.00