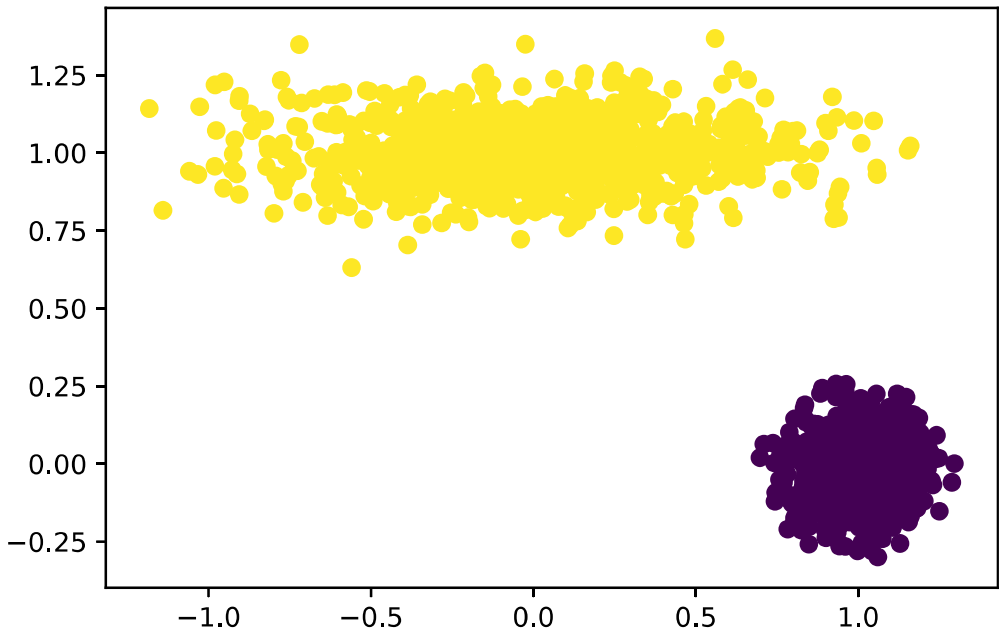
In [2]:
```
import numpy as np #numerical computing module
import matplotlib.pyplot as plt #plotting module
#when using a notebook, the line below will display any plots directly in it:
%matplotlib inline
from __future__ import division
from PIL import Image
import numpy.fft as fft
from scipy import signal
import scipy.io
from mpl_toolkits.mplot3d import Axes3D
```

In [456]:
```
import numpy as np

def perceptron_ans(train,trainlabel,step):
    """ PERCEPTRON performs perceptron learning algorithm for binary
     classification (online version).
     TRAIN provides the training data for the perceptron. It is a n by d
     matrix. n is the number of samples and d is the dimension of the data.
     TRAINLABEL provides the label (-1 or 1) for TRAIN. It is an n-by-1 vecto
r.
     The function outputs weights (w(1) is the weight for the bias unit). It
     is a d+1 by 1 vector.
     STEP is the step size.
     OUTPUT: "w", the weights of the perceptron."""

    #get dimensions of data matrix
    n = train.shape[0]
    d = train.shape[1]

    #appending the bias coefficient (1) to each data point
    train = np.concatenate([np.ones((n,1)), train],axis=1)

    #--------Your code below-----------

    #Hint: you might find numpy's np.random.permutation function to be useful


    w = np.zeros(d)
    w = np.insert(w, 0, 1)
    counter = 0
    f = np.zeros(n).reshape([n,1])

    while(counter < 30):
        for i in np.random.permutation(n):
            if (np.dot(w, train[i]) > 0):
                f[i] = 1
            else:
                f[i] = -1
            if (f[i]*trainlabel[i]) < 0:
                w = w + step * train[i] * trainlabel[i]
        if (f == trainlabel).all():
            break
        counter += 1
    return w, counter
```

In [396]:
```
d = scipy.io.loadmat('Data0a.mat')
trainlabel = d['trainlabel']
train = d['train']
```

In [77]:
```
plt.scatter(train[:,0], train[:,1], c=np.ndarray.flatten(trainlabel))
```

Out[77]:
```
<matplotlib.collections.PathCollection at 0xb637688>
```



1b.

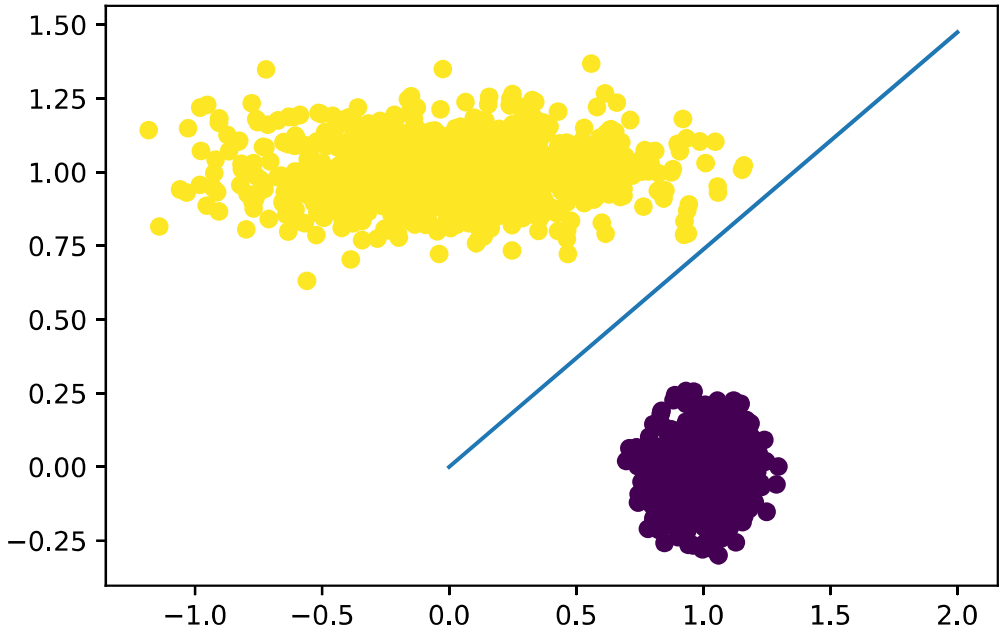
These two classes are linear separable

In [398]:
```
w, count = perceptron_ans(train, trainlabel, 1)
w, count
```

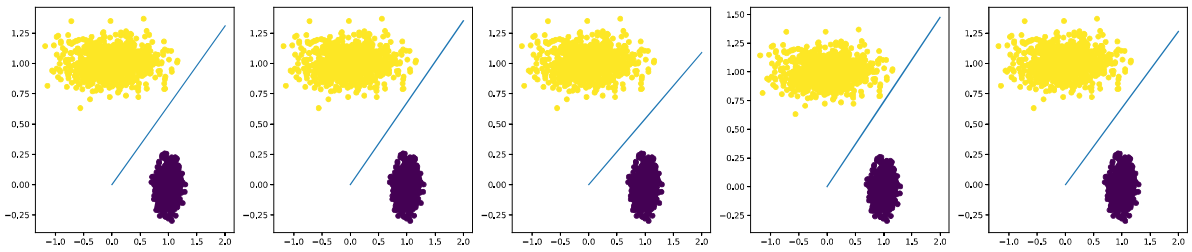
Out[398]:
```
(array([ 0.        , -1.25175731,  1.69760992]), 1)
```

In [417]:
```
x = np.arange(3)
y = (-w[0] - x*w[1]) / w[2]
plt.scatter(train[:,0], train[:,1], c=np.ndarray.flatten(trainlabel))
plt.plot(x,y)
```

Out[417]: [<matplotlib.lines.Line2D at 0x11580308>]



In [418]:
```
f, axes = plt.subplots(1,5, figsize=(25,5))
for i in range(5):
    w, count = perceptron_ans(train, trainlabel, 1)
    x = np.arange(3)
    y = (-w[0] - x*w[1]) / w[2]
    axes[i].scatter(train[:,0], train[:,1], c=np.ndarray.flatten(trainlabel))
    axes[i].plot(x,y)
```



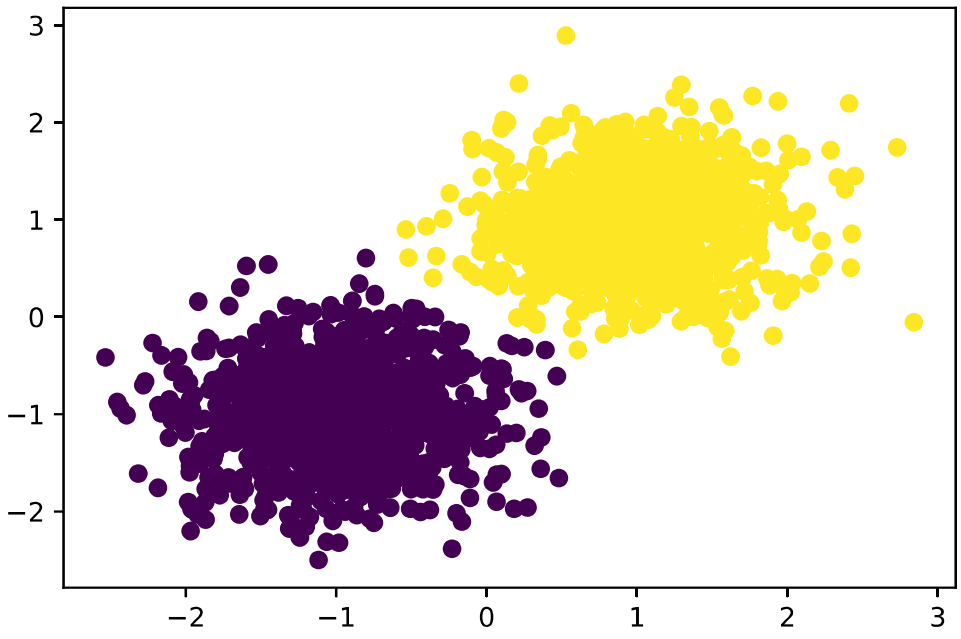
1b. continued:

Yes, the results differ from run to run. This is because we visit the training data in a random order so the weights are never the same each time we test our weights on a certain data point, which causes different f values which then causes the weights to be updated by different values each time.

In [455]:
```
d = scipy.io.loadmat('perceptron/Data0b.mat')
trainlabel = d['trainlabel']
train = d['train']
```

In [457]:
```
plt.scatter(train[:,0], train[:,1], c=np.ndarray.flatten(trainlabel))
```

Out[457]:
```
<matplotlib.collections.PathCollection at 0x13de80c8>
```



1c.

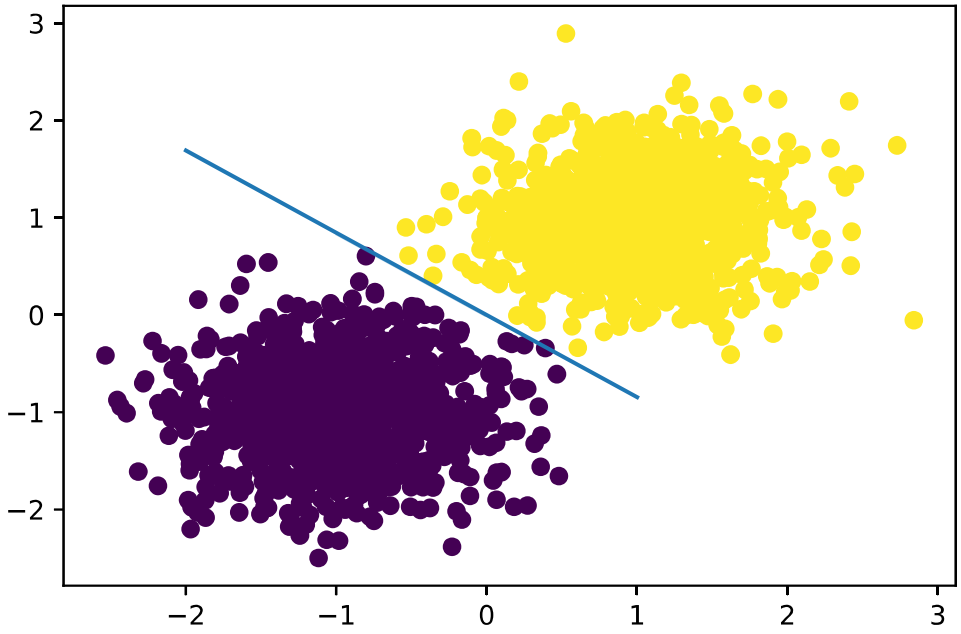
Yes, these are linearly separable, just barely

In [421]:
```
w, count = perceptron_ans(train, trainlabel, .1)
w, count
```

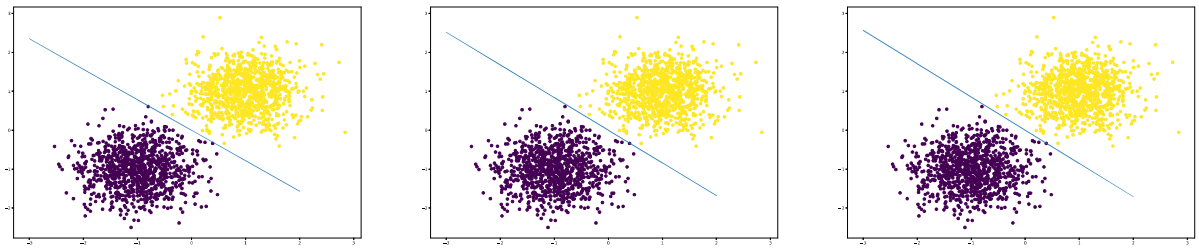
Out[421]:
```
(array([1.38777878e-16, 5.37140330e-01, 6.35548120e-01]), 3)
```

In [422]:
```
x = np.arange(-2,2)
y = (-w[0] - x*w[1]) / w[2]
plt.scatter(train[:,0], train[:,1], c=np.ndarray.flatten(trainlabel))
plt.plot(x,y)
```

Out[422]: [<matplotlib.lines.Line2D at 0x10a7c448>]



In [424]:
```
f, axes = plt.subplots(1,3, figsize=(50,10))
for i in range(3):
    w, count = perceptron_ans(train, trainlabel, .1)
    x = np.arange(-3,3)
    y = (-w[0] - x*w[1]) / w[2]
    axes[i].scatter(train[:,0], train[:,1], c=np.ndarray.flatten(trainlabel))
    axes[i].plot(x,y)
```



1c. continued:

The difference between this dataset and the last one is that this one is a lot closer together. Because of this, the results from multiple runs is less variable because there is less room between the two groups to fit a line through. It makes sense to use a small step size here so the perceptron doesn't overshoot and take a long time to figure out the classification, because the data is so close together.

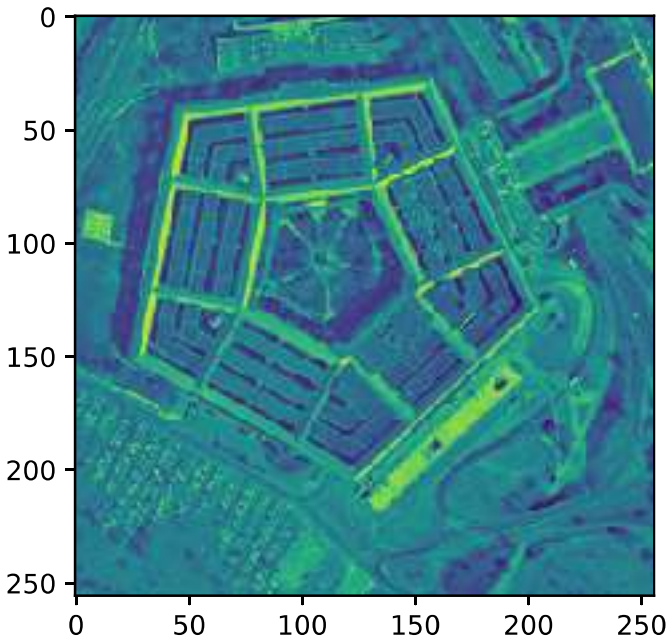
In [444]:
```
import numpy as np
def disparity(leftimg, rightimg, halfpatchsize, maxdisp=None):
        # """DISPARITY Calculates the disparity for a pair of rectified stereo
images.
        #   DISPMATRIX = DISPARITY(LEFTIMG, RIGHTIMG, HALFPATCHSIZE, MAXDISP)
 calculates
        #   the pointwise disparity between a pair of rectified stereo images.

        #   LEFTIMG is the left image of the stereo pair, while RIGHTIMG is th
e
        #   right image.

        #   HALFPATCHSIZE specifies half of the patch size (M in the problem
        #   statement in the homework). The patch size actually used ends up b
eing
        #   2*HALFPATCHSIZE + 1.

        #   MAXDISP is an optional parameter specifying the maximum absolute
        #   disparity to be tested for. If omitted, it defaults to HALFPATCHSI
ZE.

        #   The output argument DISP should contain the disparity between the
 two
        #   images at every point where a valid patch comparison can be made.
 Note
        #   that, due to boundary effects, the size of DISPMATRIX will be smal
ler.
        # """

    if maxdisp is None:
        maxdisp = halfpatchsize


        #get image dimensions
    nrows, ncols = leftimg.shape

    if nrows != rightimg.shape[0] or ncols != rightimg.shape[1]:
        raise ValueError("Left and right images aren't of the same size")


        #make sure you understand why the disparity matrix will be smaller tha
n the input images!
    dispmatrix = np.zeros((np.array(leftimg.shape) - 2*halfpatchsize - np.arra
y([0, 2*maxdisp])))

        #### i added the brackets below
        #range of values for the central patch positions
    y0s = np.arange(halfpatchsize,(nrows - halfpatchsize))
    x0s = np.arange((halfpatchsize + maxdisp),(ncols - halfpatchsize - maxdisp
))

    for j in range(len(x0s)):
        for i in range(len(y0s)):

                #get actual (x0,y0) position in the image
            x0 = x0s[j] #this is the horizontal index (col)
```

```
            y0 = y0s[i] #this is the vertical index (row)
            maxcos = 0
            dxvalue = -maxdisp
            leftpatch = leftimg[y0 - halfpatchsize:y0+halfpatchsize + 1, x0 -
halfpatchsize:x0+halfpatchsize+1]
            for dx in range(-maxdisp, maxdisp+1):
                rightpatch = rightimg[y0-halfpatchsize:y0 + halfpatchsize+1,x0
- halfpatchsize + dx:x0 + halfpatchsize+1 + dx]
                cos = np.sum(leftpatch*rightpatch) / np.sqrt(np.sum(np.power(l
eftpatch, 2))) / np.sqrt(np.sum(np.power(rightpatch,2)))
                if (maxcos < cos):
                    maxcos = cos
                    dxvalue = dx
            dispmatrix[i,j] = dxvalue    #disparity value found for current po
sition; # dx with max cos similarity
    return dispmatrix
```

In [445]:
```
pleft = 'stereo/Pentagon-Left.tiff'
pright = 'stereo/Pentagon-Right.tiff'
im1 = np.array(Image.open(pleft))
pentleft = im1.astype('float')/255
im2 = np.array(Image.open(pright))
pentright = im2.astype('float')/255
```

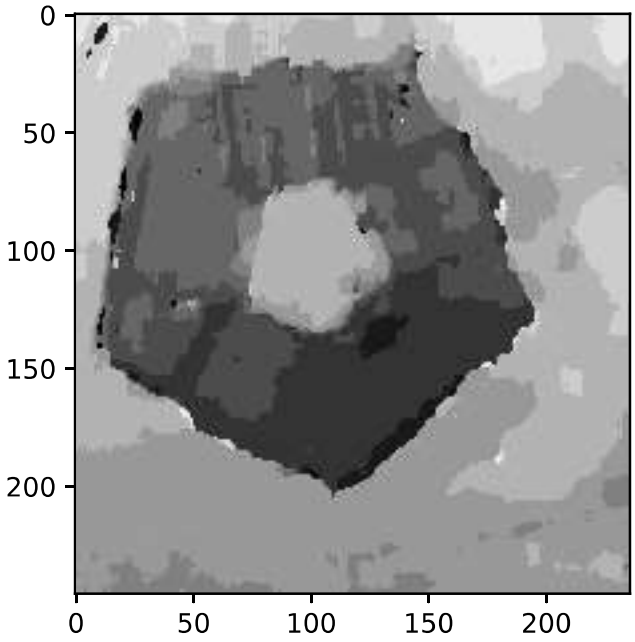
In [454]:
```
plt.imshow(pentleft)
```

Out[454]: <matplotlib.image.AxesImage at 0x1216d688>



In [446]:
```
dispmatrix = disparity(pentleft, pentright, 5)
plt.imshow(dispmatrix, cmap='gray')
```

Out[446]: <matplotlib.image.AxesImage at 0x104e4b48>

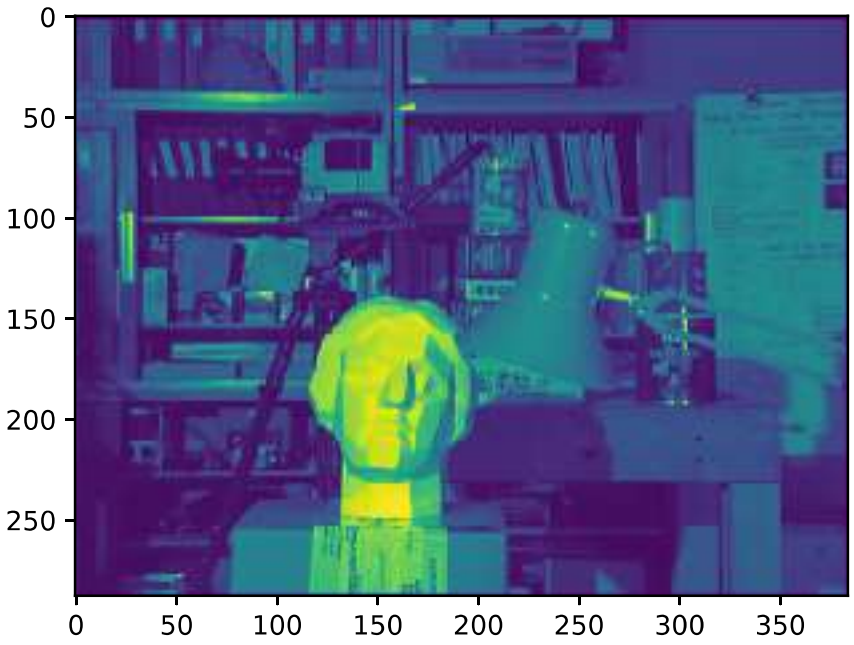


In [ ]:
```
```

In [449]:
```
pleft = 'stereo/Scene-Left.tiff'
pright = 'stereo/Scene-Right.tiff'
im1 = np.array(Image.open(pleft))
sceneleft = im1.astype('float')/255
im2 = np.array(Image.open(pright))
sceneright = im2.astype('float')/255
```

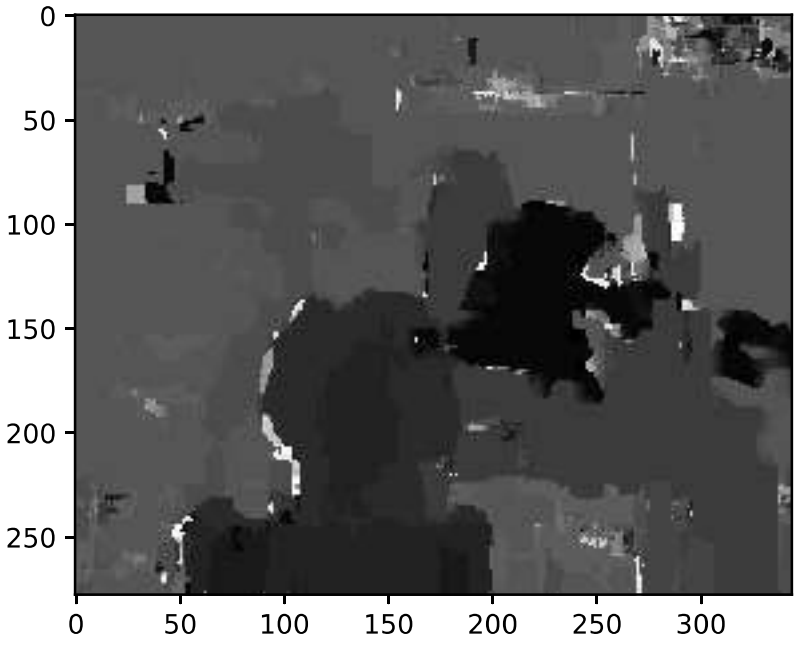
In [450]:
```
plt.imshow(sceneleft)
```

Out[450]: <matplotlib.image.AxesImage at 0x1461c688>



In [453]:
```
dispmatrix = disparity(sceneleft, sceneright, 5, maxdisp=15)
plt.imshow(dispmatrix, cmap='gray')
```

Out[453]: <matplotlib.image.AxesImage at 0x10ac2048>



Imports

In [112…
```
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
import numpy as np
import time
import os

from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import InputLayer, Input
from tensorflow.python.keras.layers import Reshape, MaxPooling2D
from tensorflow.python.keras.layers import Conv2D, Dense, Flatten,GlobalAveragePooling2
```

Load Data

In [113…
```
from tensorflow.python.keras.datasets.mnist import load_data

# load the data - it returns 2 tuples of digits & labels - one for
# the train set & the other for the test set
(x_train, y_train), (x_test, y_test) = load_data()

#convert image shape to the format (height, width, n_channels)
img_shape = x_train.shape[1:]# (height, width, n_channels)
if len(img_shape) == 2:
    x_train = x_train[...,None]
    x_test = x_test[...,None]
    img_shape = x_train.shape[1:]

#convert pixel intensities to floats in the range [0,1]
if x_train.dtype == np.uint8:
    x_train = x_train.astype('float32')/255.
    x_test = x_test.astype('float32')/255.
```

Q: How many traning and test examples are there? What shape is each image in the dataset? How many different classes are there?

In [114…
```
#YOUR CODE/ANSWER HERE
N_train = 60000
N_test = 10000
img_shape = (28, 28, 1) #use a tuple in the fomat: (height, width, n_channels)
n_classes = 10
```

Now convert the class numbers in `y_train` and `y_test` into one-hot encoded arrays. Use the function `tf.keras.utils.to_categorical` .

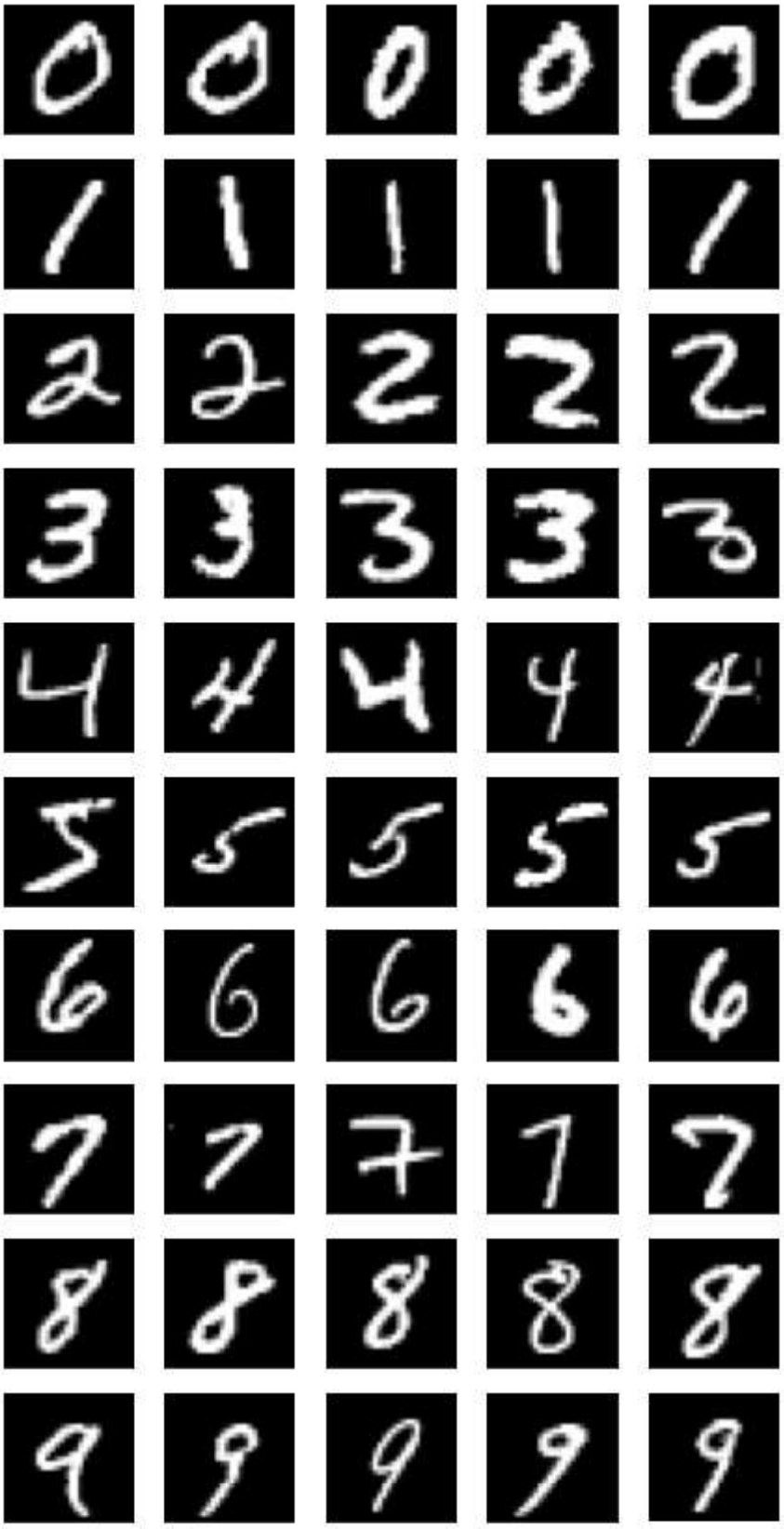
Hint: run `?tf.keras.utils.to_categorical` in a separate notebook cell to open up the docs for this function, or simply hit `shift+tab` twice once your cursor is inside the parentheses in `tf.keras.utils.to_categorical(...)` .

In [115…
```
y_train_cat = tf.keras.utils.to_categorical(y_train)
y_test_cat = tf.keras.utils.to_categorical(y_test)
```

Now we will plot the first 5 training examples of each class as subplots in an `n_classes` -by-5 grid. Name each class and put your class names in a list `class_names` .

In [116…

```
class_names = ["zero", "one", "two", "three", "four", "five", "six", "seven", "eight",
f, axes = plt.subplots(n_classes,5,figsize=(5*1.5,n_classes*1.5))
plt.axis('off')
for c in range(n_classes):
    class_examples = x_train[(y_train==c).ravel()]
    for i in range(5):
        ax = axes[c,i]
        ax.imshow(class_examples[i].squeeze(),cmap='gray')
        ax.set_xticks([]); ax.set_yticks([])
```



Now you will start building your network model. We will use the Sequential Model class for that. More info here.

Fully-connected layers only

In [117…

```
# Start construction of the Keras Sequential model.
```

```
model = Sequential()

# Input layer for setting the shape of the input images
model.add(InputLayer(input_shape=img_shape))

model.add(Flatten()) #Vectorize the input image

# (Hidden) fully-connected / dense layer with ReLU-activation.
model.add(Dense(64, activation='relu'))

# Last fully-connected / dense layer with softmax-activation
# for use in classification.
model.add(Dense(n_classes, activation='softmax'))
```

Check here for an interactive visualization of what a softmax layer does.

The next step is compiling your model using the optimizer chosen above. You should set the loss function to be used, together with any metrics you might want computed after each training step. More info on metrics here.

Usually, we use negative log-likelihood or cross entropy as the loss function for the softmax classifier (check this link).

In [118…
```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Learn more about Adam and other gradient descent-based methods here:
http://ruder.io/optimizing-gradient-descent/index.html

Training

Believe it or not, after these simple and intuitive steps, your network model is ready to be trained! Use the `fit` method, passing the inputs `x`, desired outputs `y`, and setting the number of `epochs` and `batch_size`. (Be careful: if you choose too large a batch size, your machine might get really slow if it runs short of RAM -- if that happens, simply choose smaller batches.)

In [119…
```
results = model.fit(x=x_train,
          y=y_train_cat,
          epochs=1, batch_size=64)
```

```
Epoch 1/1
60000/60000 [==============================] - 6s 101us/step - loss: 0.3590 - acc: 0.899
0
```

Testing

Another method available for the Sequential model class is `evaluate`. This runs your network in "test mode" without updating any weights (no backprop), while evaluating any metrics that have been set for the model.

In [120…
```
result = model.evaluate(x=x_test,
                        y=y_test_cat)
for name, value in zip(model.metrics_names, result):
    print(name, value)
```

```
10000/10000 [==============================] - 1s 123us/step
loss 0.20357884513884783
```

```
acc 0.9424
```

Adding convolution + pooling layers

Now we will build a convolutional "neural" network with two convolution layers and see if it can do a better job at classifying handwritten digits. For a tutorial on convolutional nets, please check this nice introduction: Part 1|Part 2.

Q: Fill in the missing parts below to specify the details of your convolution layer. Namely, the kernel (filter) size, stride length, and the number of filters to use.

In [121…

```
# Start construction of the Keras Sequential model.
model = Sequential()

# Input layer for setting the shape of the input images
model.add(InputLayer(input_shape=img_shape))

# First convolutional layer with ReLU-activation and max-pooling.
model.add(Conv2D(kernel_size= (2,2), ### choose kernel size as a 2-tuple, e.g. (5,5) (u
                 strides= 1, ### choose spacing between applications of the filters acr
                 filters= 100, ### choose number of filters for this layer
                 padding='same',
                 activation='relu',
                 name='layer_conv1'))

model.add(MaxPooling2D(pool_size=(2,2), strides=2))


# Flatten the output of the convolutional layers
# so it can be used as input to a fully-connected / dense layer.
model.add(Flatten())

model.add(Dense(64, activation='relu'))

# Last fully-connected / dense layer with softmax-activation
# for use in classification.
model.add(Dense(n_classes, activation='softmax'))

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Training

In [122…

```
results = model.fit(x=x_train,
          y=y_train_cat,
          epochs=1, batch_size=64)
```

```
Epoch 1/1
60000/60000 [==============================] - 215s 4ms/step - loss: 0.1949 - acc: 0.942
7
```

Testing

In [123…

```
result = model.evaluate(x=x_test,
                        y=y_test_cat)
for name, value in zip(model.metrics_names, result):
    print(name, value)
```

```
10000/10000 [==============================] - 11s 1ms/step
loss 0.08145608260594309
acc 0.9748
```

Do you observe any improvement?

Yes, it improved its accuracy, though at the expense of speed of computation. Training it with 100 filters took a long time

Tuning your conv net

Q: Modify your network model by making changes to the parameters you chose above (e.g. filter size, n. of filters) and/or including an additional conv and/or pooling layer. Check if your modifications improve or worsen the performance of your network.

In [53]:
```
### YOU CODE HERE
##### I changed around the values, making the strides and kernal_size smaller and added
# filters which increased the accurracy (see above)
##### When I added an additional conv layer and pooling layer, it performed worse (see
# below which resulted in .8342 accuracy)
##### I know the below code doesn't use nearly as many filters,
# but I compared the two both with low filter amounts and the single layer worked bette
##### I didn't really want to sit through the super long training time to keep testing
# Start construction of the Keras Sequential model.
model = Sequential()

# Input layer for setting the shape of the input images
model.add(InputLayer(input_shape=img_shape))

# First convolutional layer with ReLU-activation and max-pooling.
model.add(Conv2D(kernel_size= (2,2), ### choose kernel size as a 2-tuple, e.g. (5,5) (u
                 strides= 1, ### choose spacing between applications of the filters acr
                 filters= 2, ### choose number of filters for this layer
                 padding='same',
                 activation='relu',
                 name='layer_conv1'))

model.add(MaxPooling2D(pool_size=(2,2), strides=2))

# First convolutional layer with ReLU-activation and max-pooling.
model.add(Conv2D(kernel_size= (2,2), ### choose kernel size as a 2-tuple, e.g. (5,5) (u
                 strides= 2, ### choose spacing between applications of the filters acr
                 filters= 5, ### choose number of filters for this layer
                 padding='same',
                 activation='relu',
                 name='layer_conv2'))

model.add(MaxPooling2D(pool_size=(2,2), strides=2))


# Flatten the output of the convolutional layers
# so it can be used as input to a fully-connected / dense layer.
model.add(Flatten())

model.add(Dense(64, activation='relu'))

# Last fully-connected / dense layer with softmax-activation
# for use in classification.
model.add(Dense(n_classes, activation='softmax'))
```

```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

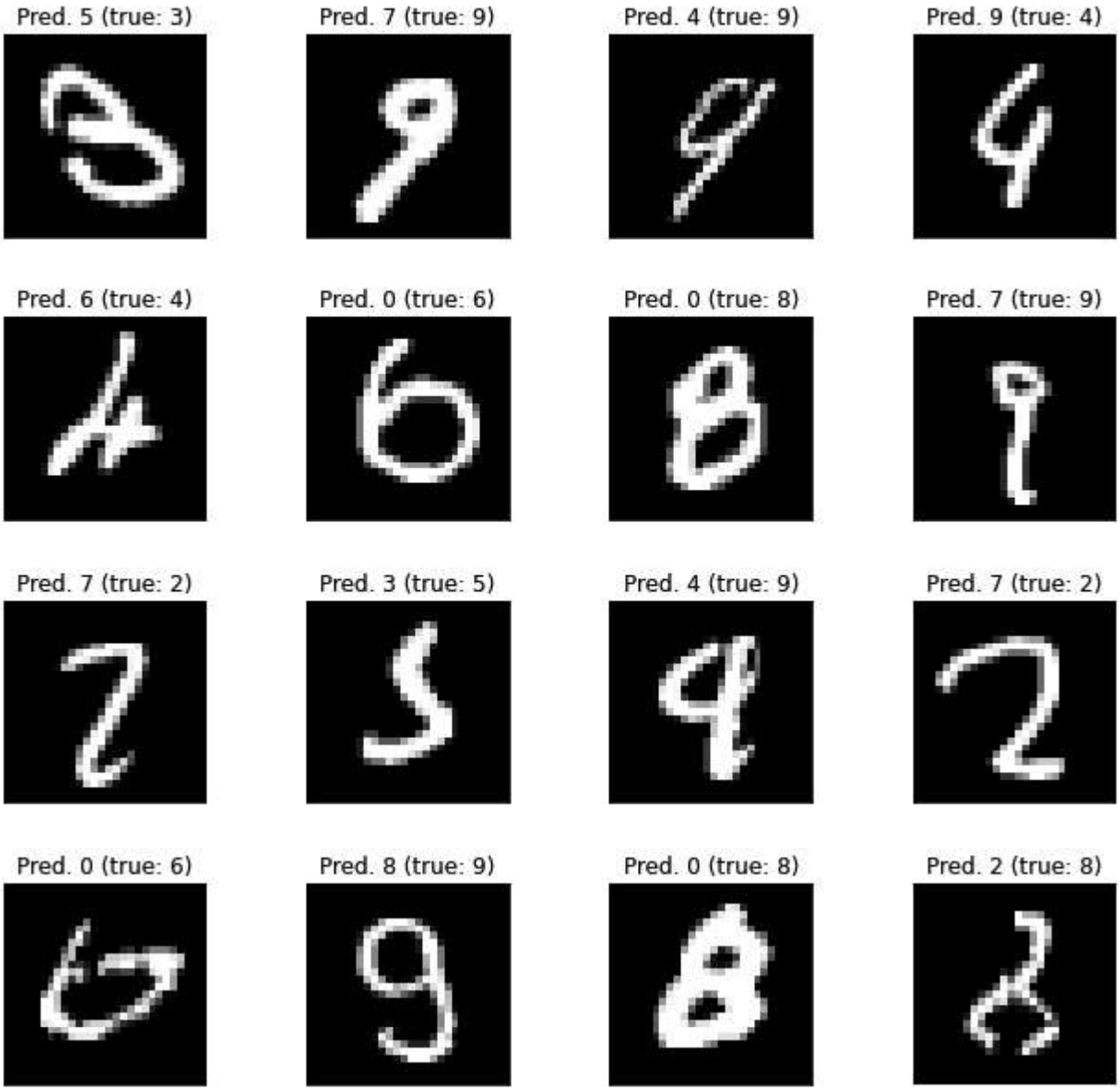
Analyzing your results

Now let's take a look at what kinds of images your network wasn't able to correctly classify. we can use `predict` to get the network output for a given input set of images:

In [124…
```
y_pred = model.predict(x=x_test)
```

We will plot below 16 examples of images in the test set that are incorrectly classified. They are labeled by their true class and the class predicted by the network.

In [125…
```
wrong_class = np.flatnonzero(y_pred.argmax(axis=1) != y_test)
class_names = range(10)
nrows = 4
ncols = 4
fig, axes = plt.subplots(4,4,figsize=(ncols*2.7,nrows*2.7)) #get a new fig and subplot
fig.subplots_adjust(wspace=.5)
plt.axis('off')
ii = 0
for row in range(nrows):
    for col in range(ncols):
        ax = axes[row,col]#get the axis for the current subplot
        img_index = wrong_class[ii]
        img = x_test[img_index]
        true_label = class_names[y_test[img_index]]
        pred_label = class_names[y_pred[img_index].argmax()]
        ax.imshow(img.squeeze(), cmap='gray')
        ax.set_title('Pred. {} (true: {})'.format(pred_label, true_label))
        ax.set_xticks([]); ax.set_yticks([])
        ii += 1
```



Visualization of Layer Weights and Outputs

In order to access the layers in your network, it might be helpful to print out a summary of your model.

In [126…
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
layer_conv1 (Conv2D)         (None, 28, 28, 100)       500
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 14, 14, 100)       0
_________________________________________________________________
flatten_8 (Flatten)          (None, 19600)             0
_________________________________________________________________
dense_16 (Dense)             (None, 64)                1254464
_________________________________________________________________
dense_17 (Dense)             (None, 10)                650
=================================================================
Total params: 1,255,614
Trainable params: 1,255,614
```

```
Non-trainable params: 0
_________________________________________________________________
```

We count the indices to get the layers we want. The first conv layer has index 0.

In [127…
```
layer_conv1 = model.layers[0]
```

Once we have the layers, easily get to their current weights with `get_weights()` .

In [128…
```
weights_conv1, bias_conv1 = layer_conv1.get_weights()

#alway check whether the shapes make sense!
weights_conv1.shape,bias_conv1.shape
```

Out[128… `((2, 2, 1, 100), (100,))`

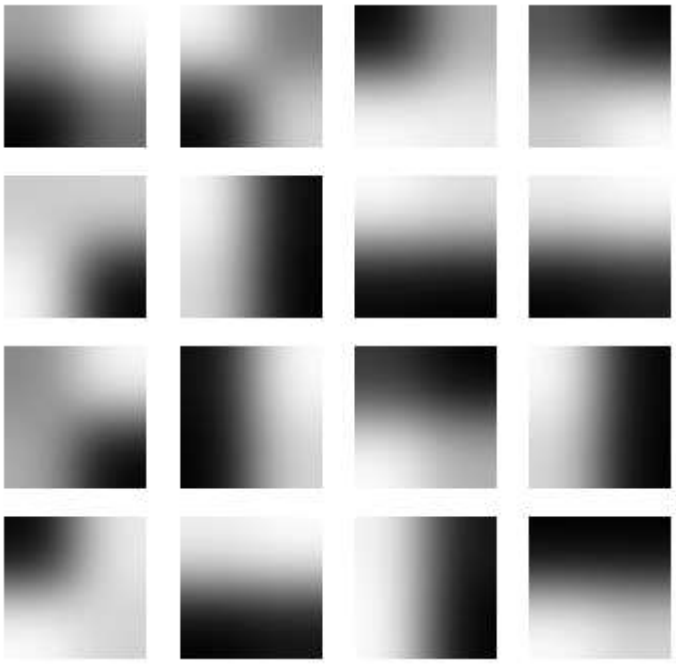
In [129…
```
w = weights_conv1[:,:,0,0]
plt.imshow(w,cmap='gray')
plt.axis('off')
```

Out[129… `(-0.5, 1.5, 1.5, -0.5)`

Q: Plot the first 16 filters for the first conv layer.

(you can try setting `interpolation='gaussian'` in `imshow` to see if the appearance of the filters improve with a little blurring)

In [130…
```
ncols = 4
nrows = 4
f,axes = plt.subplots(ncols,nrows,figsize=(6,6))
plt.axis('off')
for fmap in range(ncols*nrows):
    ax = axes[fmap//ncols,fmap%ncols]
    ax.axis('off')

    w = weights_conv1[:,:,:,fmap]# YOUR CODE HERE
    ax.imshow(w, cmap='gray', interpolation='gaussian')
```



Plotting the output of a convolutional layer

For this, we will use the `K` backend function which will create a function from a part of your Keras model. (It's actually less complicated than it sounds!)

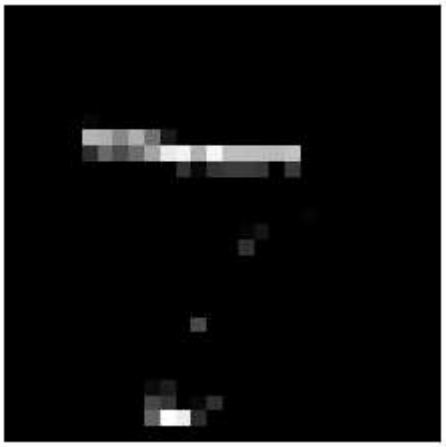
In [131…
```
from tensorflow.python.keras import backend as K

#since layer_conv1 is the first real layer in our model (index 0),
# the input to layer_conv1 are the actual input images
output_conv1 = K.function(inputs=[layer_conv1.input],
                          outputs=[layer_conv1.output])
```

In [132…
```
#pick an image from the test set to feed it to the network
im1 = np.expand_dims(x_test[0],0)

#put the image inside a list since the layer expects a 4-d input tensor as input
out1 = output_conv1([ im1 ])[0]
out1.shape
```

Out[132… `(1, 28, 28, 100)`

In [133…
```
fmap_to_use = 0
plt.imshow(out1[0,:,:,fmap_to_use],cmap='gray')
plt.axis('off')
```

Out[133… `(-0.5, 27.5, 27.5, -0.5)`



Q: Plot the outputs for several filters in that layer.

In [134…
```
### YOUR CODE HERE
f,axes = plt.subplots(4,4,figsize=(5,5))
plt.axis('off')
for i in range(16):
    ax = axes[i//4,i%4]
    ax.axis('off')
    ax.imshow(out1[0,:,:,i], cmap='gray')
```

Testing the network on variations of the dataset

It almost seems like the network might be learning something about shape. Is it really?

It does not seem like it since switching contrast makes it quite inaccurate and shifting the images is also pretty bad. This would indicate that it is not about shape but rather pixel values in specific locations I believe.

Opposite contrast

First let's test it against the same shapes, only with different contrast.

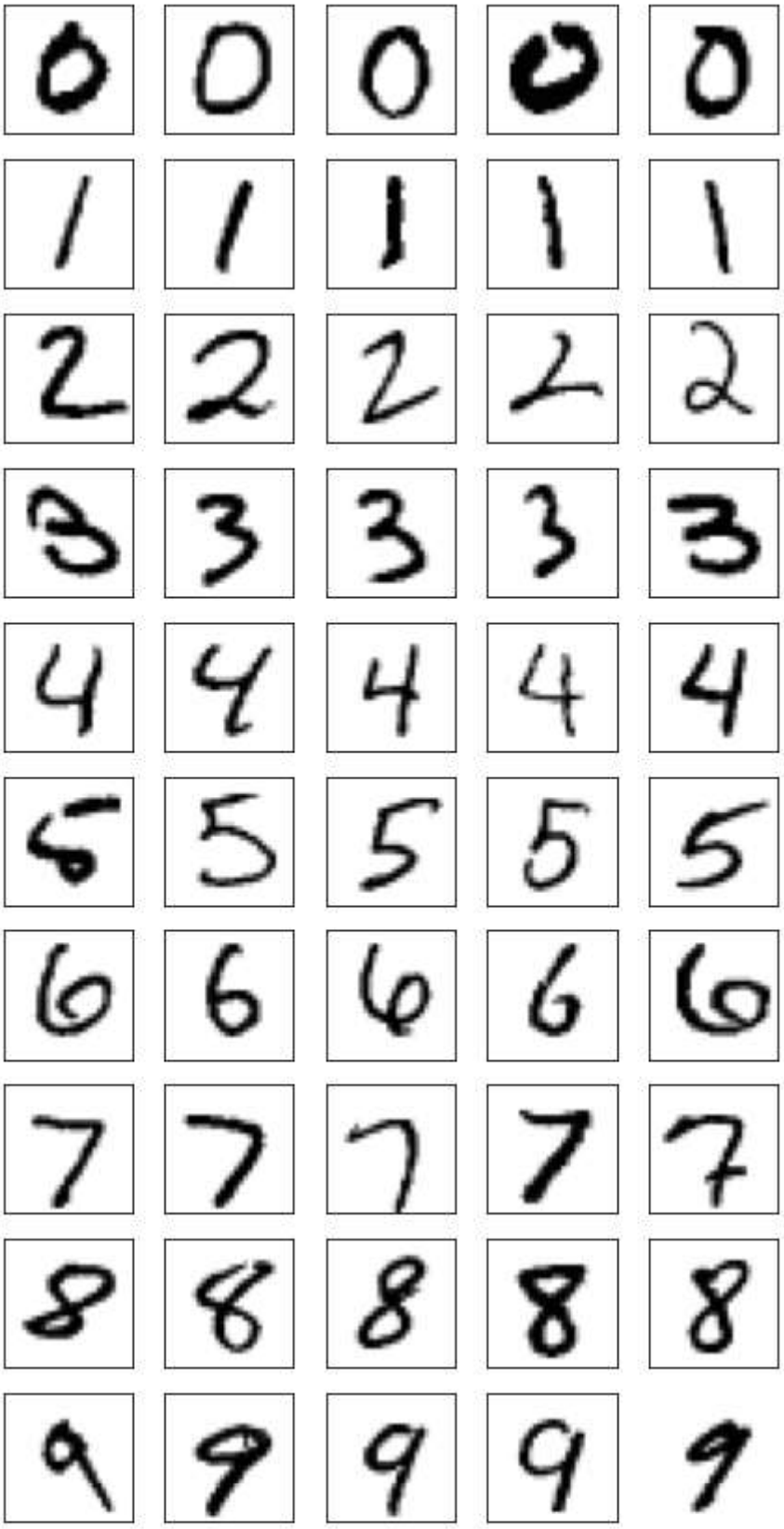
In [135…
```
# Since pixel vals are in the [0,1] range, subtracting them from 1 reverses the contras
neg_x_test = 1. - x_test
```

Here are examples of how the original images were altered -- can your brain still recognize the digits?

Yes, the numbers are now black while the background is white. This does not change readability for our brain

In [136…

```
f, axes = plt.subplots(n_classes,5,figsize=(5*1.5,n_classes*1.5))
plt.axis('off')
for c in range(n_classes):
    class_examples = neg_x_test[(y_test==c).ravel()]
    for i in range(5):
        ax = axes[c,i]
        ax.imshow(class_examples[i].squeeze(),cmap='gray')
        ax.set_xticks([]); ax.set_yticks([])
```



Q: Evaluate your trained model on this new test set to see how well it does against a contrast reversal of each image.

In [141…

```
# CHeck accuracy against contrast-reversed images.
result = model.evaluate(x=neg_x_test,
                        y=y_test_cat)
```

```
for name, value in zip(model.metrics_names, result):
    print(name, value)
```

```
10000/10000 [==============================] - 11s 1ms/step
loss 5.596608749389649
acc 0.3127
```

Shifting

Perhaps it can generalize to slightly shifted versions of the original images?

In [138…

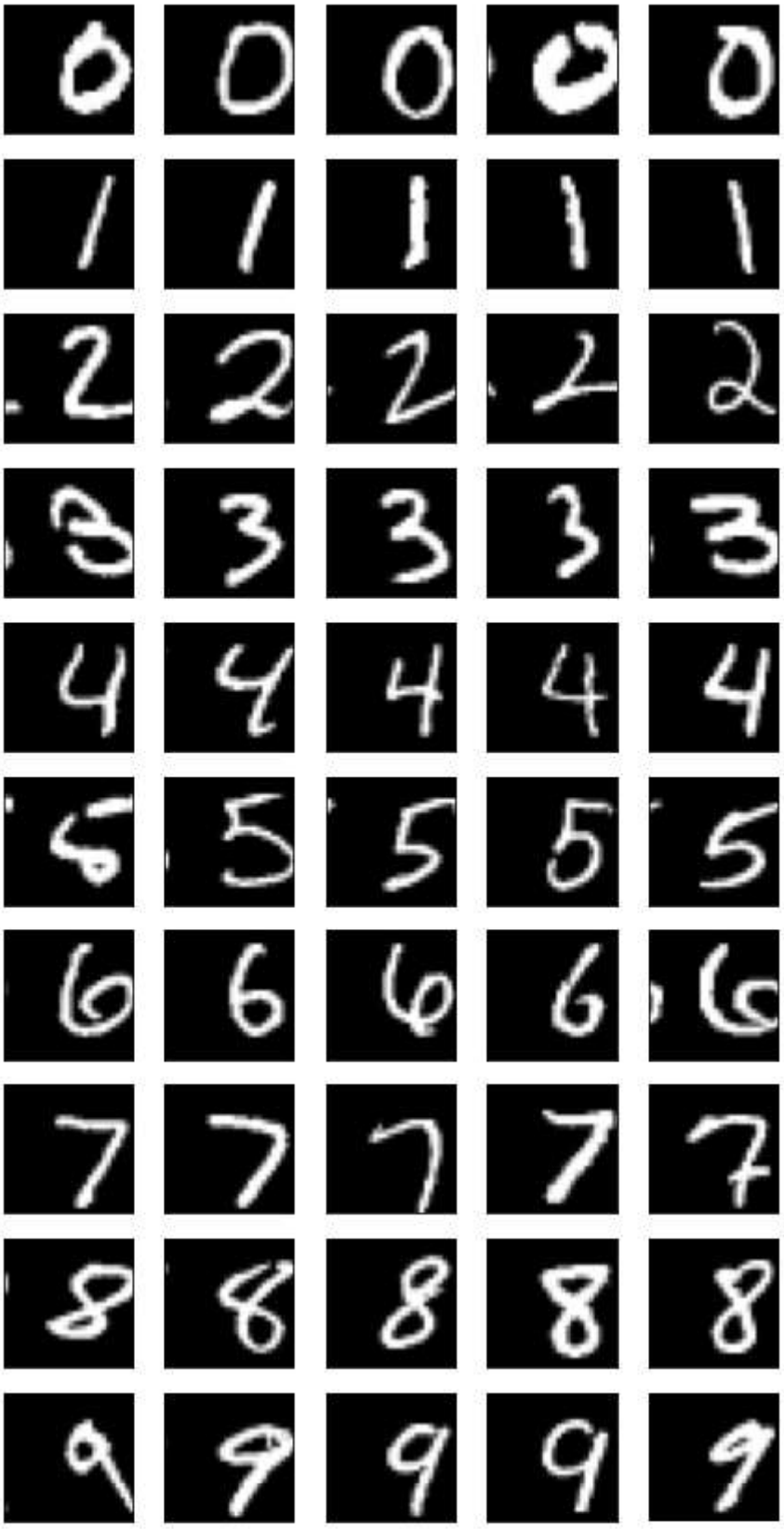
```
rolled_x_test = np.roll(x_test,5,2) #this shifts all images horizontally by 5 pixels
```

Here are examples of how the original images were altered -- can your brain still recognize the digits?

Yes, my brain can still recognize them

In [139…

```
f, axes = plt.subplots(n_classes,5,figsize=(5*1.5,n_classes*1.5))
plt.axis('off')
for c in range(n_classes):
    class_examples = rolled_x_test[(y_test==c).ravel()]
    for i in range(5):
        ax = axes[c,i]
        ax.imshow(class_examples[i].squeeze(),cmap='gray')
        ax.set_xticks([]); ax.set_yticks([])
```



Q: Evaluate your trained model on this new test set to see how well it does against a small shift of each image.

In [140…

```
result = model.evaluate(x=rolled_x_test,
                        y=y_test_cat)
for name, value in zip(model.metrics_names, result):
    print(name, value)
```

```
10000/10000 [==============================] - 11s 1ms/step
loss 3.31437773475647
acc 0.2809
```