Handwritten Digit Recognition

The goal of this project is to create a model classify grayscale 28 x 28 pixel images of handwritten digits. The data for each image is already in flattened format.

In [7]:
```
import sys
!{sys.executable} -m pip install tensorflow
import tensorflow as tf
from tensorflow import keras
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Importing, Previewing, and Preprocessing Datasets

Import the datasets.

In [8]:
```
train = pd.read_csv(r'C:\Datasets\traindigits.csv')
test = pd.read_csv(r'C:\Datasets\testdigits.csv')
```

Preview the dataset.

In [31]:
```
train.head(10)
```

Out[31]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775
0	1	0	0	0	0	0	0	0	0	0	...	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0
5	0	0	0	0	0	0	0	0	0	0	...	0	0
6	7	0	0	0	0	0	0	0	0	0	...	0	0
7	3	0	0	0	0	0	0	0	0	0	...	0	0
8	5	0	0	0	0	0	0	0	0	0	...	0	0
9	3	0	0	0	0	0	0	0	0	0	...	0	0

In [37]:
```
train.head(10).min(axis=1)
```

Out[37]:
```
0    0
1    0
2    0
3    0
4    0
5    0
6    0
7    0
8    0
9    0
dtype: int64
```

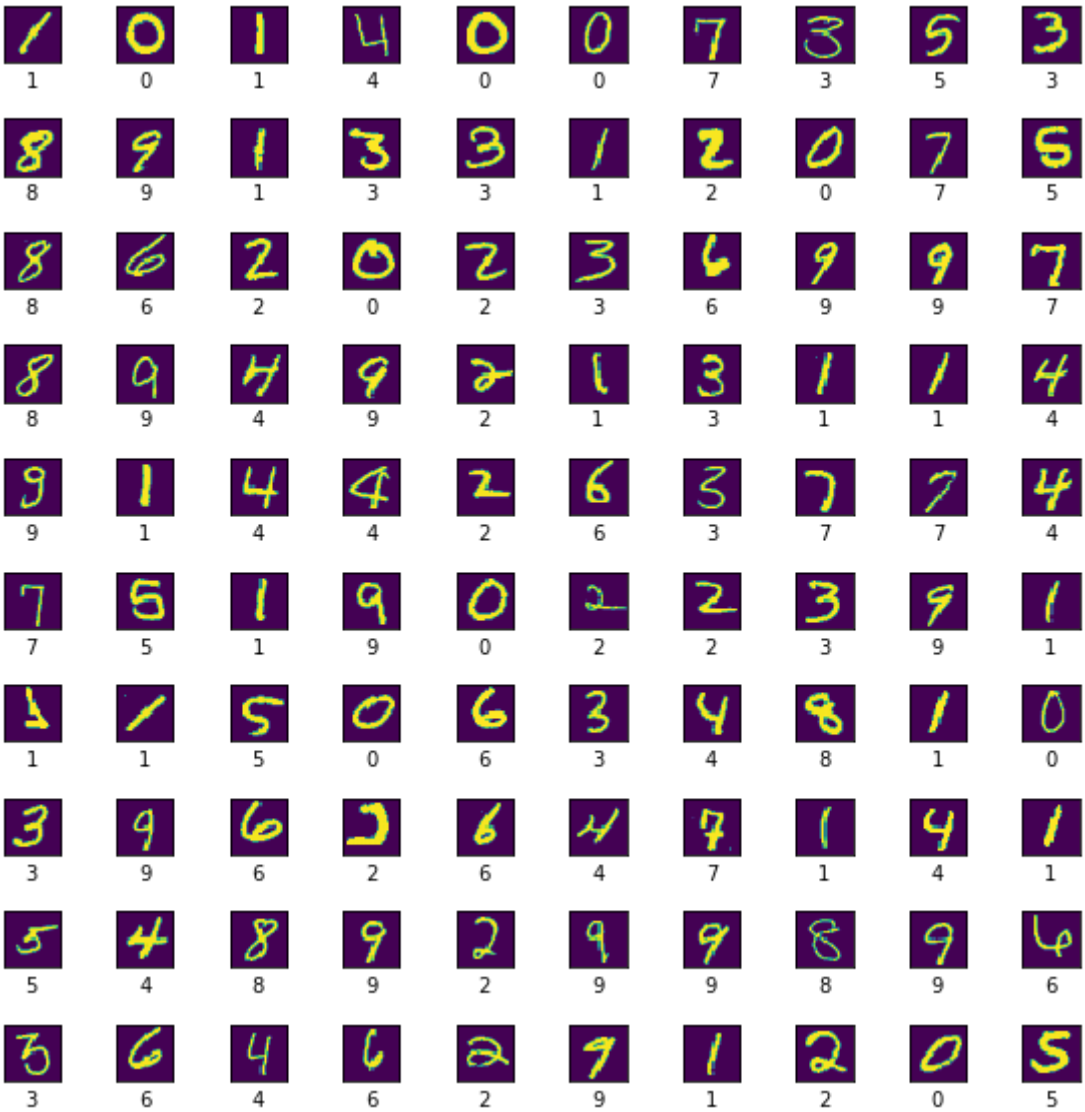
In [36]:
```
train.head(10).max(axis=1)
```

Out[36]:
```
0    255
1    255
2    255
3    255
4    254
5    255
6    255
7    255
8    255
9    255
dtype: int64
```

As shown above, the image values range from 0 to 255 (black to white).

Next we can preview the images as 28x28 arrays.

In [30]:
```
fig, ax = plt.subplots(10,10)
for i in range(10):
    for j in range(10):
        ax[i,j].imshow(np.reshape(np.array(train.iloc[10*i+j,1:]),(28,28)))
        ax[i,j].set_xticks([])
        ax[i,j].set_yticks([])
        ax[i,j].set_xlabel(train.iloc[10*i+j,0])
fig.set_size_inches((8,8))
fig.tight_layout()
```



The numbers in the images all are fairly centered and scaled to about the same size, and they are mostly black and white (not much grey), so a simple neural network should work for this data set.

Before modeling, let's normalize all the images.

In [39]:
```
train.iloc[:,1:]=train.iloc[:,1:]/255
test = test/255
```

Modeling

To avoid overfitting, we should perform cross validation to select model hyperparameters. We will do a 5-fold CV.

In [56]:
```
def CVsplit(i):
    if i==1:
        return (train.iloc[:8400,:],train.iloc[8400:,:])
    elif i==5:
        return (train.iloc[33600:,:],train.iloc[:33600,:])
    else:
        return (train.iloc[8400*(i-1):8400*i,:], train.iloc[:8400*(i-1),:].append(tr
```

This function will perform the CV given specified hyperparameters.

In [78]:
```
def evalmodel(hidden, activation1, activation2, optimizer1, epochs1):
    meanacc = 0
    for i in range(1,6):
        CVsets = CVsplit(i)
        model = keras.Sequential([keras.layers.Dense(hidden, activation=activation1)
            keras.layers.Dense(10, activation = activation2)])
        model.compile(optimizer=optimizer1, loss='sparse_categorical_crossentropy',
        model.fit(np.array(CVsets[1].iloc[:,1:]), np.array(CVsets[1].label), epochs=
        resultloss, resultacc = model.evaluate(np.array(CVsets[0].iloc[:,1:]), np.ar
        meanacc += resultacc
    meanacc /=5
    print('Hidden size',hidden, ', Activation 1:', str(activation1).split(' ')[1],',
```

Hyperparameter optimization is performed below. First the activation functions are optimized.

In [85]:
```
for j in [tf.nn.relu,tf.nn.softmax]:
    for k in [tf.nn.relu,tf.nn.softmax]:
        evalmodel(125,j,k,'adam',5)
```

```
8400/8400 [==============================] - 2s 226us/sample - loss: 2.2927 - acc:
0.1032
8400/8400 [==============================] - 2s 224us/sample - loss: 2.3012 - acc:
0.0921
8400/8400 [==============================] - 2s 246us/sample - loss: 2.3020 - acc:
0.1021
8400/8400 [==============================] - 2s 247us/sample - loss: 2.2850 - acc:
0.1055
8400/8400 [==============================] - 2s 257us/sample - loss: 2.3026 - acc:
0.1014
Hidden size 125 , Activation 1: relu , Activation 2: relu , Optimizer: adam , 5 epo
chs.  Accuracy:  0.10088095217943191
8400/8400 [==============================] - 2s 229us/sample - loss: 0.1034 - acc:
0.9696
8400/8400 [==============================] - 2s 244us/sample - loss: 0.1055 - acc:
0.9695
8400/8400 [==============================] - 2s 244us/sample - loss: 0.1092 - acc:
0.9654
8400/8400 [==============================] - 2s 245us/sample - loss: 0.1017 - acc:
0.9698
8400/8400 [==============================] - 2s 229us/sample - loss: 0.1103 - acc:
0.9656
Hidden size 125 , Activation 1: relu , Activation 2: softmax , Optimizer: adam , 5
epochs.  Accuracy:  0.967976188659668
8400/8400 [==============================] - 2s 234us/sample - loss: 2.3026 - acc:
0.0992
8400/8400 [==============================] - 2s 241us/sample - loss: 2.3026 - acc:
0.0915
8400/8400 [==============================] - 2s 235us/sample - loss: 2.3033 - acc:
0.1019
8400/8400 [==============================] - 2s 250us/sample - loss: 2.3026 - acc:
0.0979
8400/8400 [==============================] - 2s 246us/sample - loss: 2.3026 - acc:
0.1014
Hidden size 125 , Activation 1: softmax , Activation 2: relu , Optimizer: adam , 5
epochs.  Accuracy:  0.09838095158338547
8400/8400 [==============================] - 2s 257us/sample - loss: 0.4337 - acc:
0.8682
8400/8400 [==============================] - 2s 250us/sample - loss: 0.4140 - acc:
0.8846
8400/8400 [==============================] - 2s 260us/sample - loss: 0.5472 - acc:
0.8146
8400/8400 [==============================] - 2s 251us/sample - loss: 0.5481 - acc:
0.8570
8400/8400 [==============================] - 2s 252us/sample - loss: 0.5995 - acc:
0.7971
Hidden size 125 , Activation 1: softmax , Activation 2: softmax , Optimizer: adam ,
5 epochs.  Accuracy:  0.8443333387374878
```

Next, the number of epochs and the gradient descent method is optimized.

In [87]:
```
for l in ['RMSprop','adam','Nadam']:
    for m in [3,5,10]:
        evalmodel(125,tf.nn.relu,tf.nn.softmax,l,m)
```

```
8400/8400 [==============================] - 2s 292us/sample - loss: 0.1324 - acc:
0.9645
8400/8400 [==============================] - 2s 280us/sample - loss: 0.1167 - acc:
0.9663
8400/8400 [==============================] - 2s 289us/sample - loss: 0.1363 - acc:
0.9607
8400/8400 [==============================] - 2s 293us/sample - loss: 0.1377 - acc:
0.9587
8400/8400 [==============================] - 3s 306us/sample - loss: 0.1122 - acc:
0.9685
Hidden size 125 , Activation 1: relu , Activation 2: softmax , Optimizer: RMSprop ,
3 epochs.  Accuracy:  0.9637380957603454
8400/8400 [==============================] - 3s 301us/sample - loss: 0.1294 - acc:
0.9664
8400/8400 [==============================] - 3s 324us/sample - loss: 0.1108 - acc:
0.9670
8400/8400 [==============================] - 3s 314us/sample - loss: 0.1249 - acc:
0.9667
8400/8400 [==============================] - 2s 293us/sample - loss: 0.1061 - acc:
0.9708
8400/8400 [==============================] - 3s 312us/sample - loss: 0.1035 - acc:
0.9707
Hidden size 125 , Activation 1: relu , Activation 2: softmax , Optimizer: RMSprop ,
5 epochs.  Accuracy:  0.9683333277702332
8400/8400 [==============================] - 3s 313us/sample - loss: 0.1285 - acc:
0.9707
8400/8400 [==============================] - 2s 295us/sample - loss: 0.1060 - acc:
0.9725
8400/8400 [==============================] - 3s 317us/sample - loss: 0.1215 - acc:
0.9688
8400/8400 [==============================] - 3s 314us/sample - loss: 0.1188 - acc:
0.9702
8400/8400 [==============================] - 3s 317us/sample - loss: 0.1115 - acc:
0.9724
Hidden size 125 , Activation 1: relu , Activation 2: softmax , Optimizer: RMSprop ,
10 epochs.  Accuracy:  0.9709285736083985
8400/8400 [==============================] - 3s 308us/sample - loss: 0.1288 - acc:
0.9629
8400/8400 [==============================] - 3s 323us/sample - loss: 0.1069 - acc:
0.9682
8400/8400 [==============================] - 3s 311us/sample - loss: 0.1240 - acc:
0.9642
8400/8400 [==============================] - 3s 323us/sample - loss: 0.1203 - acc:
0.9640
8400/8400 [==============================] - 3s 322us/sample - loss: 0.1105 - acc:
0.9670
Hidden size 125 , Activation 1: relu , Activation 2: softmax , Optimizer: adam , 3
epochs.  Accuracy:  0.9652618885040283
8400/8400 [==============================] - 3s 321us/sample - loss: 0.1033 - acc:
0.9701
8400/8400 [==============================] - 3s 321us/sample - loss: 0.0991 - acc:
0.9700
8400/8400 [==============================] - 3s 325us/sample - loss: 0.1202 - acc:
```

```
0.9657
8400/8400 [==============================] - 3s 324us/sample - loss: 0.0986 - acc:
0.9705
8400/8400 [==============================] - 3s 326us/sample - loss: 0.0935 - acc:
0.9715
Hidden size 125 , Activation 1: relu , Activation 2: softmax , Optimizer: adam , 5
epochs.  Accuracy:  0.9695714354515076
8400/8400 [==============================] - 3s 320us/sample - loss: 0.1099 - acc:
0.9700
8400/8400 [==============================] - 3s 341us/sample - loss: 0.1178 - acc:
0.9698
8400/8400 [==============================] - 3s 342us/sample - loss: 0.1233 - acc:
0.9682
8400/8400 [==============================] - 3s 333us/sample - loss: 0.0969 - acc:
0.9724
8400/8400 [==============================] - 3s 336us/sample - loss: 0.1007 - acc:
0.9730
Hidden size 125 , Activation 1: relu , Activation 2: softmax , Optimizer: adam , 10
epochs.  Accuracy:  0.9706666707992554
8400/8400 [==============================] - 4s 454us/sample - loss: 0.1203 - acc:
0.9655
8400/8400 [==============================] - 3s 338us/sample - loss: 0.1204 - acc:
0.9635
8400/8400 [==============================] - 3s 356us/sample - loss: 0.1315 - acc:
0.9592
8400/8400 [==============================] - 3s 348us/sample - loss: 0.1260 - acc:
0.9614
8400/8400 [==============================] - 3s 359us/sample - loss: 0.1231 - acc:
0.9610
Hidden size 125 , Activation 1: relu , Activation 2: softmax , Optimizer: Nadam , 3
epochs.  Accuracy:  0.9620952486991883
8400/8400 [==============================] - 3s 358us/sample - loss: 0.0941 - acc:
0.9727
8400/8400 [==============================] - 3s 359us/sample - loss: 0.1035 - acc:
0.9700
8400/8400 [==============================] - 3s 359us/sample - loss: 0.1112 - acc:
0.9679
8400/8400 [==============================] - 3s 349us/sample - loss: 0.1098 - acc:
0.9660
8400/8400 [==============================] - 3s 403us/sample - loss: 0.1094 - acc:
0.9671
Hidden size 125 , Activation 1: relu , Activation 2: softmax , Optimizer: Nadam , 5
epochs.  Accuracy:  0.9687381029129029
8400/8400 [==============================] - 3s 375us/sample - loss: 0.1055 - acc:
0.9725
8400/8400 [==============================] - 3s 407us/sample - loss: 0.1067 - acc:
0.9724
8400/8400 [==============================] - 3s 372us/sample - loss: 0.1164 - acc:
0.9713
8400/8400 [==============================] - 3s 392us/sample - loss: 0.1080 - acc:
0.9701
8400/8400 [==============================] - 3s 372us/sample - loss: 0.1070 - acc:
0.9707
Hidden size 125 , Activation 1: relu , Activation 2: softmax , Optimizer: Nadam , 1
0 epochs.  Accuracy:  0.971404767036438
```

Finally, the size of the hidden layer is optimized.

In [90]:
```
for i in [125, 250, 400]:
    evalmodel(i,tf.nn.relu,tf.nn.softmax,'Nadam',10)
```

```
8400/8400 [==============================] - 4s 420us/sample - loss: 0.1130 - acc:
0.9721
8400/8400 [==============================] - 3s 397us/sample - loss: 0.1005 - acc:
0.9735
8400/8400 [==============================] - 3s 405us/sample - loss: 0.1179 - acc:
0.9710
8400/8400 [==============================] - 3s 390us/sample - loss: 0.1179 - acc:
0.9686
8400/8400 [==============================] - 3s 388us/sample - loss: 0.0960 - acc:
0.9732
Hidden size 125 , Activation 1: relu , Activation 2: softmax , Optimizer: Nadam , 1
0 epochs.  Accuracy:  0.971666669845581
8400/8400 [==============================] - 3s 407us/sample - loss: 0.1057 - acc:
0.9719
8400/8400 [==============================] - 3s 406us/sample - loss: 0.0970 - acc:
0.9760
8400/8400 [==============================] - 4s 421us/sample - loss: 0.1108 - acc:
0.9752
8400/8400 [==============================] - 4s 429us/sample - loss: 0.1229 - acc:
0.9661
8400/8400 [==============================] - 4s 443us/sample - loss: 0.1128 - acc:
0.9729
Hidden size 250 , Activation 1: relu , Activation 2: softmax , Optimizer: Nadam , 1
0 epochs.  Accuracy:  0.9724047541618347
8400/8400 [==============================] - 4s 454us/sample - loss: 0.1277 - acc:
0.9726
8400/8400 [==============================] - 4s 453us/sample - loss: 0.0951 - acc:
0.9765
8400/8400 [==============================] - 4s 452us/sample - loss: 0.1243 - acc:
0.9713
8400/8400 [==============================] - 4s 455us/sample - loss: 0.1072 - acc:
0.9735
8400/8400 [==============================] - 4s 459us/sample - loss: 0.1237 - acc:
0.9707
Hidden size 400 , Activation 1: relu , Activation 2: softmax , Optimizer: Nadam , 1
0 epochs.  Accuracy:  0.9729285717010498
```

A similar model with a similar number of total weights but using 2 hidden layers instead of 1 is evaluated below.

In [94]:
```
meanacc = 0
for i in range(1,6):
    CVsets = CVsplit(i)
    model = keras.Sequential([keras.layers.Dense(300, activation=tf.nn.relu),keras.l
        keras.layers.Dense(10, activation = tf.nn.softmax)])
    model.compile(optimizer='Nadam', loss='sparse_categorical_crossentropy', metrics
    model.fit(np.array(CVsets[1].iloc[:,1:]), np.array(CVsets[1].label), epochs=10,
    resultloss, resultacc = model.evaluate(np.array(CVsets[0].iloc[:,1:]), np.array(
    meanacc += resultacc
meanacc /=5
print('Accuracy: ',meanacc)
```

```
8400/8400 [==============================] - 4s 488us/sample - loss: 0.1281 - acc:
0.9723
8400/8400 [==============================] - 4s 489us/sample - loss: 0.1091 - acc:
0.9782
8400/8400 [==============================] - 4s 500us/sample - loss: 0.1417 - acc:
0.9726
8400/8400 [==============================] - 4s 497us/sample - loss: 0.1147 - acc:
0.9745
8400/8400 [==============================] - 4s 488us/sample - loss: 0.1124 - acc:
0.9762
Accuracy:  0.9747618913650513
```

The above model was the best performing model thus far, in terms of cross validation. The model will be fit and predictions made on the test set.

In [95]:
```
model.fit(np.array(train.iloc[:,1:]), np.array(train.label), epochs=10, verbose = 0)
resultloss, resultacc = model.evaluate(np.array(train.iloc[:,1:]), np.array(train.la
print('Accuracy: ',resultacc)
```

```
42000/42000 [==============================] - 12s 288us/sample - loss: 0.0100 - ac
c: 0.9971
Accuracy:  0.99707144
```

In [96]:
```
predictions = model.predict(np.array(test))
```

In [102]:
```
pd.DataFrame(np.argmax(predictions,axis=1)).to_csv(r'C:\Datasets\digitspredict.csv')
```