Gurjus Singh

MSDS 422 – Practical Machine Learning

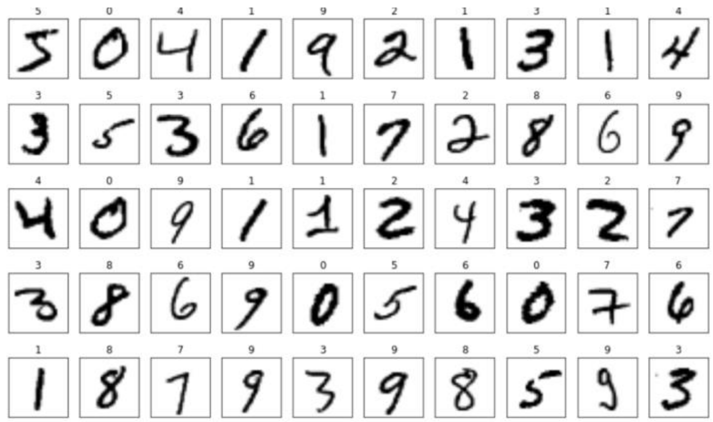
October 25^{th}, 2020

MSDS 422 ASSIGNMENT#6 Neural Networks

Data preparation, exploration, visualization

In this data analysis project, I got a chance to go over a learning algorithm that tries to mimic neurons in the brain. The algorithm is called is called Deep Neural Networks which was used for supervised learning in this analysis project. I used a dataset which I had used before called the MNIST dataset. The MNIST Dataset consisted of pixels which represented handwritten numbers, and to determine which number the pixels were it also included a label. The MNIST dataset was used from Kaggle which was used in a previous project that I did before using KMeans, but this time it was more about accuracy and less about grouping similar items [1]. I started the analysis by first loading the necessary packages such as Tensorflow, Keras, Pandas, and Matplotlib. Tensorflow was particularly important this time around as it was used for the Deep Neural Networks.

One thing that was special this time around is the Keras package had a load data function which automatically loaded the dataset from a specific location on a server. It was then split into train and test data directly. I then saw the shape of the training set and test set what I saw was that it was shape of 70,000 rows each with 28x28 set of features which is the intensity of how dark or light the image is ranging between 0 and 1 [3]. I then looked at the initial labels. I then I wanted to see how good the split was between train and test sets. It was pretty balanced meaning it was a good split to use. I then plotted the images in a subplot to see each number image represented in 1-1.



Number Image 1-1

Next I had to one hot encode the images so that the neural network could take it in. This was done by using to_categorical method where a row represented 1-9 class and 1 was used to indicate which class that image belonged to and 0s across the row indicated which classes the image was not a part of. I then reshaped the images to make it to 784 features instead of 28x28. This made the pixel values 0-255 for one image instead of 0-1. Before running my neural network, I then had to normalize the feature data, so it was values between 0 and 1. To this I had to convert both feature data sets for test and train to float32 and divide them by 255. Now our model was ready for Training.

Review research design and modeling methods

Before running the models, it is important to explain my model. I used a Deep Neural Network, a type of Artificial Neural Network, which means it has two or more layers which transforms an input which is the MNIST data into probabilities to determine which class a number is part of. To explain an Artificial Neural Network's creation was motivated by the structure of the brain, where nodes are like Neurons and the connections are like Synapse connections from one neuron to the other represent in figure 1-2 from Geron's [2].

As you can imagine the brain has a multiple layer of tissue involved as depicted in image 1-3. This is where Deep Learning Neural Networks creation came on where there are input layers, hidden layers, and output layers. One can represent number of nodes by the parameter of units in each layer [1]. There is also an activation function involved from options such as ReLu or Tanh [2]. ReLu is the main type of activation function I used which has a long name of "Rectified Linear Unit Function" [2].

ReLu works in that it will output positive number directly and it will output negative numbers as zero [2]. Tanh also known as the "hyperbolic tangent function" outputs any number between -1 and 1. Softmax and Sigmoid are also different functions but are mostly used on the output layers only during classification. They both out probabilities of what class a particular instance belongs to [2]. Softmax is used on multiclass classification [2].

For the model I used it used an optimizer I used 'RMSProp' which is used to fit the model by using gradients [3]. I used Crossentropy for the loss as it is used to find errors in the predictions [3].

Neural Networks are important to study as we can kind of how to study how the brain works. The Brain learns quite well, but does not have that much computational power like the computer. Neural Networks are beneficial in this way. This is a great equivalent to the brain. As mentioned, this is kind of an example of how the planes was created like the bird.

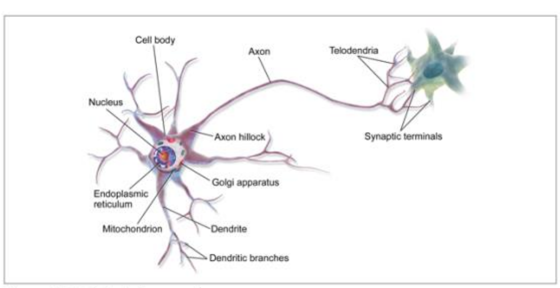
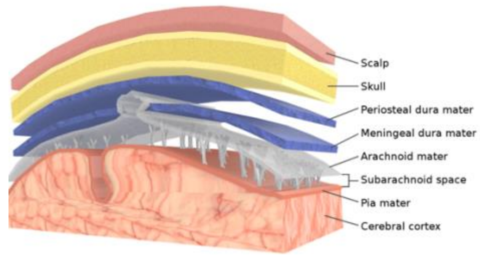
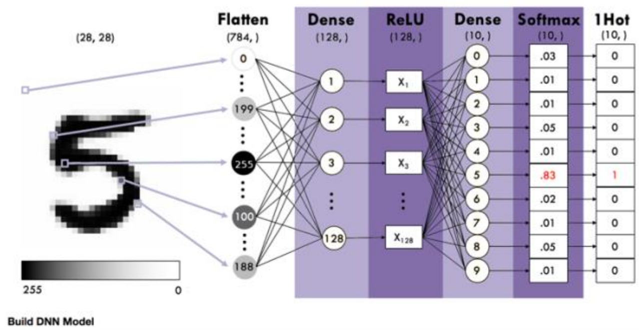


Figure 10-1. Biological neuron[3]

Neuron 1-2



Layers of the Brain 1-3



Neural Network for 1-4

In figure 1-4 is an example of the model I am running. I will input the reshaped one hot encoded dataset. I will use the ReLu activation function and will use Softmax activation function as there are 10 classes. In order to create the model I used the Keras package. I created 1 and 2 layers (with 1 hidden layer) with 64 nodes each, 1 and 2 (with 1 hidden layer) layers with 128 nodes each, and 1 and 2 (with 1 hidden layer) layers with 256 nodes each.

Review results, evaluate models

Out[508]:

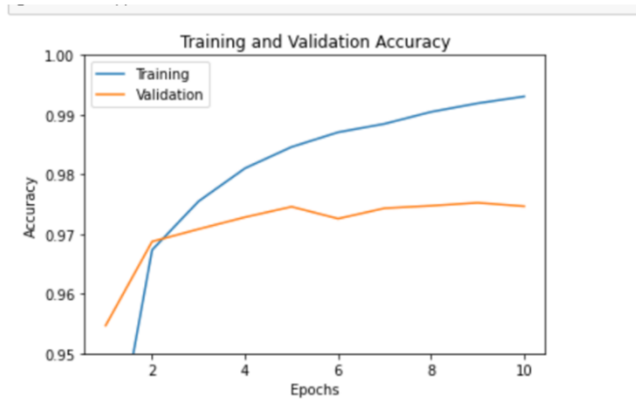
	Experiment	Processing Times	Nodes Per Layer	Layers	Train Set Accuracy	Test Set Accuracy
0	1	84.879273	128	1	0.990817	0.9748
1	2	204.390874	128	2	0.991650	0.9765
2	3	90.921688	64	2	0.989117	0.9742
3	4	71.909145	64	1	0.983983	0.9709
4	5	105.874952	256	1	0.991750	0.9798
5	6	143.033359	256	2	0.992450	0.9791

Dataframe 1-5 Results

From looking at the six experiments that I did above in 1-5 and looking at the plots, I believe experiment 3 performed the best and did not overfit as much as the others. I tried to dive deeper and what I found is my results were consistent below as I Plot 1-8 confirmed that the accuracy between the Train and Val were close for Experiment 4 compared to the others. I got an accuracy for train data of 0.9868 while Val accuracy was 0.9726.

Looking at the numbers for the other experiments confirmed that Experiment 4 had a much lower gap between train accuracy, test accuracy and validation accuracy. For example, even though Experiment 5 performed the highest for Test Data, it still had more of the gap between the Train Data and Test Data.. I thought Experiment 5 was overfitting by just looking at difference between Train Accuracy and Test Accuracy as well as Validation Set. The difference between Train and Test was 0.1335 and the Train and Val Difference for Experiment 5 was 0.0193. For Experiment 4 the differences were 0.013083 for Train and Test Data, while for Train and Val Data it was 0.0142.

I then wanted to dive a little deeper into the results for confusion matrix for the experiments. I looked at Confusion Matrices for both Train and Test Sets in Figures 1-12 to 1-21. I did not see Experiment 4 perform particularly better on the Matrices, and Experiment 5 performed the best. The Confusion Matrix did not provide an overall picture of which Experiment performed the best. I think it was the accuracies which were the biggest indicator as well as the differences between them.

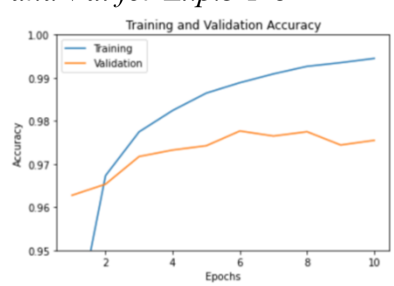


Train and Validation for Experiment 2 1-7

Train and Validation Accuracy for Experiment 1 1-6

Training and Val for Exp. 4 1-9

Train and Val for Exp.3 1-8



Training and Val for Exp 5. 1-10

Training and Val for Exp. 6 1-11

```
Out[452]: <tf.Tensor: id=2534160, shape=(10, 10), dtype=int32, numpy=
          array([[5902,    1,    5,    1,    1,    1,    2,    1,    3,    6],
                 [   0, 6717,    6,    5,    2,    0,    2,    3,    7,    0],
                 [   5,   11, 5885,   25,    3,    0,    1,   13,   12,    3],
                 [   0,    1,   11, 6096,    0,    7,    0,    4,    6,    6],
                 [   3,   10,    3,    3, 5768,    0,    4,    7,    4,   40],
                 [  12,    1,    5,   45,    2, 5319,   13,    1,    8,   15],
                 [  17,    2,    3,    1,    6,    5, 5879,    0,    5,    0],
                 [   1,    9,   10,   10,    3,    2,    0, 6215,    2,   13],
                 [  16,    9,    5,   17,    3,    6,    3,    6, 5770,   16],
                 [   3,    1,    0,    7,    9,    3,    0,   23,    5, 5898]], dtype=int32)>
```

Confusion Train Matrix for Exp. 1 1-12

```
Out[465]: <tf.Tensor: id=2593441, shape=(10, 10), dtype=int32, numpy=
array([[5906,    0,    4,    1,    1,    2,    3,    0,    4,    2],
       [   0, 6710,    4,   12,    0,    0,    3,    7,    6,    0],
       [   7,    4, 5867,   51,    2,    1,    3,    6,   16,    1],
       [   1,    0,    7, 6104,    0,   10,    0,    0,    6,    3],
       [   2,    8,    3,    0, 5787,    0,    6,    9,    1,   26],
       [   2,    0,    4,   26,    1, 5357,   13,    1,   12,    5],
       [   9,    1,    0,    1,    1,    3, 5898,    0,    5,    0],
       [   3,    4,    5,   41,    3,    3,    0, 6192,    4,   10],
       [   4,    6,    0,   21,    2,   10,    5,    1, 5798,    4],
       [   9,    2,    0,   18,   10,   10,    0,   12,    8, 5880]], dtype=int32)>
```

Confusion Train Matrix for Exp.2 1-13

```
Out[478]: <tf.Tensor: id=2666294, shape=(10, 10), dtype=int32, numpy=
array([[5886,    0,    4,    1,    0,    2,   12,    0,   14,    4],
       [   0, 6671,   10,    4,    9,    2,    8,   12,   22,    4],
       [   3,    2, 5902,   14,    3,    1,    5,    9,   15,    4],
       [   3,    0,   22, 6000,    1,   45,    1,    7,   38,   14],
       [   1,    3,    3,    0, 5798,    1,    7,    4,    4,   21],
       [   3,    1,    3,   11,    2, 5354,   18,    1,   14,   14],
       [   4,    0,    2,    0,    3,    4, 5903,    0,    2,    0],
       [   2,    3,   15,    6,   11,    7,    0, 6183,   10,   28],
       [   4,    4,   12,    9,    0,   16,   12,    3, 5782,    9],
       [   4,    1,    0,    3,   29,   11,    1,   20,   12, 5868]], dtype=int32)>
```

ConfusionTrain Matrix for Exp.3 1-14

```
Out[489]: <tf.Tensor: id=2732210, shape=(10, 10), dtype=int32, numpy=
array([[5891,    0,    2,    1,    0,    4,   10,    1,    8,    6],
       [   1, 6695,   10,    6,    0,    0,    5,    3,   17,    5],
       [  15,    7, 5865,   17,    1,    3,    6,   14,   26,    4],
       [   3,    3,   25, 6006,    0,   36,    2,   10,   35,   11],
       [   5,   14,   10,    2, 5607,    4,   24,    9,   17,  150],
       [   4,    3,    4,   19,    1, 5340,   16,    3,   17,   14],
       [  12,    1,    1,    0,    6,    8, 5885,    0,    5,    0],
       [   4,   21,   16,    8,    8,    3,    1, 6135,   17,   52],
       [   9,   10,    5,   26,    0,   19,   15,    3, 5752,   12],
       [   8,    4,    0,   20,    8,   18,    0,   14,   14, 5863]], dtype=int32)>
```

Confusion Train Matrix for Exp.4 1-13

```
Out[498]: <tf.Tensor: id=2798126, shape=(10, 10), dtype=int32, numpy=
array([[5894,    3,    6,    1,    0,    1,    2,    1,    9,    6],
       [   0, 6731,    3,    3,    1,    0,    1,    1,    2,    0],
       [   2,    9, 5927,    5,    1,    1,    1,    3,    9,    0],
       [   0,    4,   26, 6045,    0,    4,    1,    9,   38,    4],
       [   0,   12,    9,    0, 5789,    0,    4,    5,    3,   20],
       [   2,    4,    8,   36,    1, 5301,   16,    1,   43,    9],
       [   2,    6,    2,    0,    4,    2, 5896,    0,    6,    0],
       [   1,   20,    8,    3,    2,    2,    0, 6223,    3,    3],
       [   3,    9,    7,    6,    1,    1,    2,    2, 5816,    4],
       [   3,    1,    0,    4,   13,    4,    0,   24,   17, 5883]], dtype=int32)>
```

Confusion Train Matrix for Exp.5 1-14

```
Out[507]: <tf.Tensor: id=2864193, shape=(10, 10), dtype=int32, numpy=
array([[5883,    2,    7,    2,    0,    5,   13,    1,    9,    1],
       [   0, 6720,    2,    1,    4,    0,    2,    7,    4,    2],
       [   3,    4, 5922,    8,    4,    0,    1,    6,   10,    0],
       [   0,    0,   13, 6077,    0,   20,    0,    5,    9,    7],
       [   0,    5,    1,    0, 5799,    0,    9,    3,    1,   24],
       [   1,    0,    3,   18,    2, 5371,   10,    1,   11,    4],
       [   1,    0,    1,    1,    1,   11, 5901,    0,    2,    0],
       [   0,    3,    9,   10,   10,    7,    0, 6207,    2,   17],
       [   3,    5,    3,   25,    2,    4,    3,    4, 5794,    8],
       [   4,    1,    1,    7,   29,   14,    0,   13,    7, 5873]], dtype=int32)>
```

Confusion Train Matrix for Exp.6 1-15

```
Out[525]: <tf.Tensor: id=2929343, shape=(10, 10), dtype=int32, numpy=
array([[ 971,    0,    1,    2,    1,    1,    1,    1,    2,    0],
       [   0, 1116,    4,    1,    0,    1,    2,    2,    9,    0],
       [   4,    2,  998,    8,    2,    0,    2,    7,    9,    0],
       [   0,    0,    4,  986,    0,    7,    0,    2,    7,    4],
       [   1,    0,    4,    1,  963,    0,    3,    3,    0,    7],
       [   2,    0,    0,    8,    1,  867,    7,    1,    5,    1],
       [   4,    1,    0,    1,    8,    2,  941,    0,    1,    0],
       [   1,    2,    7,    2,    1,    0,    0, 1007,    2,    6],
       [   3,    0,    4,    6,    5,    5,    5,    1,  941,    4],
       [   3,    3,    0,    8,    9,    4,    0,    6,    4,  972]], dtype=int32)>
```

Confusion Test Matrix for Exp. 1 1-16

```
ut[538]: <tf.Tensor: id=2987067, shape=(10, 10), dtype=int32, numpy=
array([[ 971,    2,    0,    1,    1,    0,    3,    1,    1,    0],
       [   0, 1127,    2,    1,    0,    0,    2,    1,    2,    0],
       [   3,    1, 1014,    2,    1,    0,    3,    4,    4,    0],
       [   0,    0,    6,  985,    0,   10,    0,    4,    3,    2],
       [   0,    2,    5,    1,  956,    0,    3,    1,    0,   14],
       [   2,    0,    0,   12,    1,  866,    4,    1,    3,    3],
       [   3,    2,    1,    1,    2,    4,  943,    0,    2,    0],
       [   0,    4,   11,    1,    1,    0,    0, 1004,    0,    7],
       [   2,    1,    5,    4,    2,    2,    1,    5,  947,    5],
       [   2,    3,    0,    8,    5,    2,    0,    8,    3,  978]], dtype=int32)>
```

Confusion Test Matrix for Exp. 2 1-17

```
Out[551]: <tf.Tensor: id=3058363, shape=(10, 10), dtype=int32, numpy=
array([[ 957,    0,    2,    1,    0,   12,    4,    1,    1,    2],
       [   0, 1124,    2,    1,    0,    0,    2,    1,    5,    0],
       [   1,    2, 1003,    7,    0,    1,    3,    2,   11,    2],
       [   0,    1,    3,  982,    0,   17,    0,    3,    4,    0],
       [   1,    2,    3,    1,  949,    0,    6,    1,    2,   17],
       [   2,    0,    0,    3,    0,  878,    2,    0,    5,    2],
       [   2,    3,    1,    0,    5,   15,  931,    0,    1,    0],
       [   0,    6,   12,    6,    1,    2,    0,  981,    2,   18],
       [   0,    0,    3,    3,    1,    7,    1,    2,  953,    4],
       [   0,    2,    0,    8,    7,    7,    1,    3,    2,  979]], dtype=int32)>
```

Confusion Test Matrix for Exp. 3 1-18

```
Out[562]: <tf.Tensor: id=3122722, shape=(10, 10), dtype=int32, numpy=
array([[ 970,    0,    1,    0,    1,    2,    4,    1,    1,    0],
       [   0, 1119,    3,    1,    0,    1,    5,    0,    6,    0],
       [   6,    1,  989,    9,    3,    0,    4,    7,   13,    0],
       [   1,    0,    2,  987,    0,    6,    0,    8,    6,    0],
       [   2,    0,    4,    0,  972,    0,    1,    0,    1,    2],
       [   5,    1,    0,    8,    3,  854,   11,    0,    7,    3],
       [   4,    2,    0,    0,    5,    6,  939,    0,    2,    0],
       [   1,    2,    6,    8,    4,    0,    0,  996,    3,    8],
       [   6,    0,    4,    7,    8,    3,    4,    2,  937,    3],
       [   3,    3,    1,    7,   19,    3,    1,    4,    5,  963]], dtype=int32)>
```

Confusion Test Matrix for Exp. 4 1-19

```
ut[571]: <tf.Tensor: id=3187081, shape=(10, 10), dtype=int32, numpy=
array([[ 970,    0,    1,    0,    2,    1,    3,    1,    2,    0],
       [   0, 1124,    3,    1,    0,    1,    2,    2,    2,    0],
       [   4,    2, 1004,    2,    3,    0,    2,    7,    8,    0],
       [   0,    0,    2,  997,    0,    2,    0,    4,    2,    3],
       [   0,    0,    2,    1,  970,    0,    4,    1,    0,    4],
       [   3,    0,    0,   18,    1,  859,    5,    0,    3,    3],
       [   3,    2,    0,    1,    8,    7,  935,    0,    2,    0],
       [   0,    3,   11,    3,    7,    0,    0,  996,    3,    5],
       [   1,    0,    6,    5,    7,    3,    1,    2,  944,    5],
       [   1,    2,    0,    4,   13,    2,    0,    1,    1,  985]], dtype=int32)>
```

Confusion Test Matrix for Exp. 5 1-20

```
Out[580]: <tf.Tensor: id=3251591, shape=(10, 10), dtype=int32, numpy=
array([[ 968,    1,    0,    1,    2,    2,    3,    1,    1,    1],
       [   0, 1118,    2,    2,    0,    1,    3,    3,    5,    1],
       [   6,    0,  999,    4,    5,    0,    1,    9,    8,    0],
       [   0,    0,    2,  990,    0,    8,    0,    5,    4,    1],
       [   1,    0,    2,    0,  965,    0,    2,    1,    1,   10],
       [   2,    0,    0,    6,    1,  873,    3,    0,    3,    4],
       [   4,    2,    0,    0,   20,    3,  926,    0,    2,    1],
       [   0,    0,    5,    3,    4,    1,    0, 1005,    5,    5],
       [   3,    0,    3,    2,    5,    3,    3,    6,  945,    4],
       [   1,    1,    0,    4,   15,    2,    0,    6,    4,  976]], dtype=int32)>
```

Confusion Test Matrix for Exp. 6 1-21

Implementation and programming

For programming implementation, it was important to import the packages and use p**ip install** when necessary. The packages that made a big difference this week were **Tensorflow version 2.0.0 or above and Keras package 2.4.0 as seen in 1-22.** I also needed **time module** to calculate the processing time. In order to import the time, I needed **.load_data()** method from

Keras package. I had to use 4 varaiables mainly x and y train and x and y test to break up the data to do the test and data split. I use **.shape** attribute to see the shape and noticed the features were 28x28 of gray scale values [3]. I then saw which numbers were in the data set in train and test sets and noticed they were balanced. I found the most common number by using **Counter(),most_common().**

I then showed the subplots of each number using **plt.subplot() and also plt.imshow()** which stitched together the images. Once I saw a subset of the data, I then did one hot encoding to so the target variables were binary values 0 and 1 where 1 was which class it was in while 0 was which class it was not in. I used **to_categorical()** function to convert the target variable/column to binary. I then **.reshape** the features to a set of 784 features from 28 by 28. I then normalized the dataset by first using .**astype('float32') to convert the data and then divided by 255 with the syntax /255.**

It was time to create the model. I needed to use **Sequential()** object to create the model. In the Sequential object I need to use **Dense()** object to create the layers. In the Dense object I passed in the name particularly when I wanted to create hidden layers. I also specified **units/nodes/neurons** for each layer and specified the activation function. I also had to specify shape for **input_layer** which was 784. I then set the Sequential object equal to model. I then saw the summary of the model using **.summary().** I then used kera.utils.plot_model() to see a picture of the inputs and outputs to each layer. I then had to use .**compile() for functions** on how to compute loss, accuracy, and optimize the model. The optimizer was **'rmsprop'**, and the loss was **'categorical_crossentropy'**. I then used **model.fit()** to train the model splitting 20% to validation set to find out accuracy and see if model was overfitting. I then used **model.evaluate()** to find the accuracy on test set. I used **matplotlib to** plot accuracy scores for 10 accuracy scores for 10 epochs for the train and val set. This was used in order to find out if the model was overfitting or underfitting. I used **.predict()** to see how good the model was predicting on data which returned the classes it predicted for data features. I then used **tf.math.confusion_matrix() from Tensorflow package** to create a confusion matrix which gave us a good indicator of if the classes were right by seeing high numbers on the diagonals of the matrix.

```
In [6]: # Helper libraries
import datetime
import time
import pydot_ng as pydot
from packaging import version
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler

from collections import Counter
import numpy as np
import pandas as pd

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from tensorflow import keras
from tensorflow.keras import models
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
from plot_keras_history import plot_history
```

Import Packages 1-22

Exposition, problem description, and management recommendations

From initially coming into this analysis learning KMeans I was really impressed, because I wanted to see if there was a learning model that could actually predict accuracy. I was disappointed in KMeans for the fact it did not do clustering that well and could not classify. Neural Networks indeed accomplished my goal! I believe Experiment 4 performed by looking at Data frame 1-5 as the discrepancy between Train and Test Accuracy was low, as well as Train and Validation Accuracy. I computed the differences and found there was a difference of 0.0193 and 0.0142 which was why I chose Experiment 4. Even though Experiment 5 had the best Test Accuracy I did realize it did overfit more. Therefore, **I recommend to management** to use Experiment which was the model that had **1 layer with 64 nodes; which is Experiment 4**. In the future I hope to continue experimenting with Neural Networks and playing around with its complexity as I find Neural Networks related to what I studied in Cognitive Science and learning about the brain in my undergrad at UC Berkeley.

References

[1] https://www.kaggle.com/c/digit-recognizer

[2] Geron, A. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Incorporated.

[3] https://conx.readthedocs.io/en/latest/MNIST.html

[4] https://keras.io/api/optimizers/

Appendix

In [6]:
```
# Helper libraries
import datetime
import time
import pydot_ng as pydot
from packaging import version
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler

from collections import Counter
import numpy as np
import pandas as pd

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow.keras.utils import to_categorical
from tensorflow import keras
from tensorflow.keras import models
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist
from plot_keras_history import plot_history
```

In [7]:
```
%matplotlib inline
np.set_printoptions(precision=3, suppress=True)
```

In [8]:
```
print("This notebook requires TensorFlow 2.0 or above")
print("TensorFlow version: ", tf.__version__)
assert version.parse(tf.__version__).release[0] >=2
```

```
This notebook requires TensorFlow 2.0 or above
TensorFlow version:  2.0.0
```

In [9]:
```
print("Keras version: ", keras.__version__)
```

```
Keras version:  2.2.4-tf
```

In [10]:
```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

In [11]:
```
(x_train, y_train), (x_test, y_test)= tf.keras.datasets.mnist.load_d
```

In [12]:
```
print('x_train:\t{}'.format(x_train.shape))
```

```
print('y_train:\t{}'.format(y_train.shape))
print('x_test:\t\t{}'.format(x_test.shape))
print('y_test:\t\t{}'.format(y_test.shape))
```

```
x_train:        (60000, 28, 28)
y_train:        (60000,)
x_test:         (10000, 28, 28)
y_test:         (10000,)
```

In [13]:

```
print("First ten labels training dataset:\n {}\n".format(y_train[0:1
```

```
First ten labels training dataset:
 [5 0 4 1 9 2 1 3 1 4]
```

In [437]:

```
x_train
```

Out[437]:

```
array([[[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]],

       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
```

In [14]:
```
Counter(y_train).most_common()
```

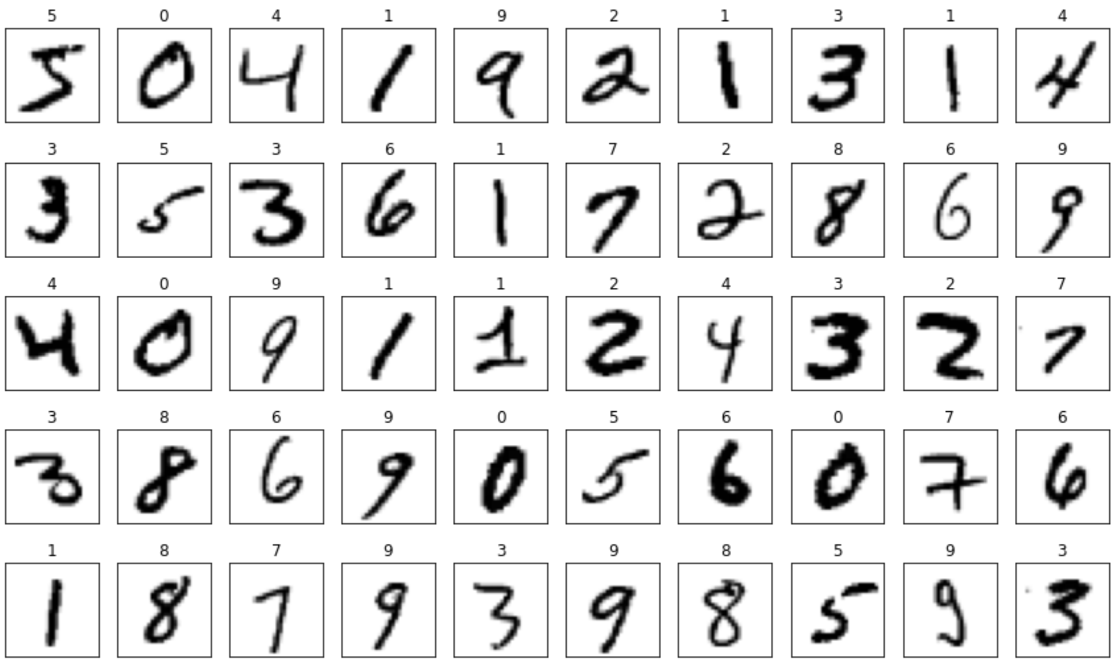
Out[14]:
```
[(1, 6742),
 (7, 6265),
 (3, 6131),
 (2, 5958),
 (9, 5949),
 (0, 5923),
 (6, 5918),
 (8, 5851),
 (4, 5842),
 (5, 5421)]
```

In [15]:
```
Counter(y_test).most_common()
```

Out[15]:
```
[(1, 1135),
 (2, 1032),
 (7, 1028),
 (3, 1010),
 (9, 1009),
 (4, 982),
 (0, 980),
 (8, 974),
 (6, 958),
 (5, 892)]
```

In [16]:
```
fig = plt.figure(figsize = (15, 9))

for i in range(50):
    plt.subplot(5, 10, 1+i)
    plt.title(y_train[i])
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_train[i].reshape(28,28), cmap='binary')
```



In [17]:
```

y_train_encoded = to_categorical(y_train)
y_test_encoded = to_categorical(y_test)

print("First ten entries of y_train:\n {}\n".format(y_train[0:10]))
print("First ten rows of one-hot y_train:\n {}".format(y_train_encod
```

```
First ten entries of y_train:
 [5 0 4 1 9 2 1 3 1 4]

First ten rows of one-hot y_train:
 [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
```

In [18]:
```
print('y_train_encoded shape: ', y_train_encoded.shape)
```

```
print('y_test_encoded shape: ', y_test_encoded.shape)
```

```
y_train_encoded shape:  (60000, 10)
y_test_encoded shape:  (10000, 10)
```

In [19]:
```
print('x_train:\t{}'.format(x_train.shape))
print('x_test:\t\t{}'.format(x_test.shape))
```

```
x_train:        (60000, 28, 28)
x_test:         (10000, 28, 28)
```

In [20]:
```
x_train_reshaped = np.reshape(x_train, (60000, 784))
x_test_reshaped = np.reshape(x_test, (10000, 784))

print('x_train_reshaped shape: ', x_train_reshaped.shape)
print('x_test_reshaped shape: ', x_test_reshaped.shape)
```

```
x_train_reshaped shape:  (60000, 784)
x_test_reshaped shape:  (10000, 784)
```

In [21]:
```
print(set(x_train_reshaped[0]))
```

```
{0, 1, 2, 3, 9, 11, 14, 16, 18, 23, 24, 25, 26, 27, 30, 35, 36, 39,
43, 45, 46, 49, 55, 56, 64, 66, 70, 78, 80, 81, 82, 90, 93, 94, 10
7, 108, 114, 119, 126, 127, 130, 132, 133, 135, 136, 139, 148, 150,
154, 156, 160, 166, 170, 171, 172, 175, 182, 183, 186, 187, 190, 19
5, 198, 201, 205, 207, 212, 213, 219, 221, 225, 226, 229, 238, 240,
241, 242, 244, 247, 249, 250, 251, 252, 253, 255}
```

In [22]:
```
np.set_printoptions(linewidth=np.inf)
print("{}".format(x_train[2020]))
```

```
[[   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0  167  208   19
0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0   13  235  254   99
0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0   74  254  234    4
0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0  154  254  145    0
0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0  224  254   92    0
0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0   51  245  211   13    0
0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    2  169  254  101    0    0
0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0   27  254  254   88    0    0
0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0   72  255  241   15    0    0
0    0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0   88  254  153    0    0   33
53  155  156  102   15    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0  130  254   31    0  128  235  2
54  254  254  254  186   10    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0  190  254   51  178  254  246  2
13  111  109  186  254  145    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0  192  254  229  254  216   90
0    0    0   57  254  234    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0  235  254  254  247   85    0
0    0    0   32  254  234    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0  235  254  254  118    0    0
0    0    0  107  254  201    0    0    0    0    0    0]
```

In [23]:
```python
x_train_norm = x_train_reshaped.astype('float32') / 255
x_test_norm = x_test_reshaped.astype('float32') / 255
```

In [24]:
```python
print(set(x_train_norm[0]))
```

```
{0.0, 0.011764706, 0.53333336, 0.07058824, 0.49411765, 0.6862745,
0.101960786, 0.6509804, 1.0, 0.96862745, 0.49803922, 0.11764706, 0.
```

MODEL 1

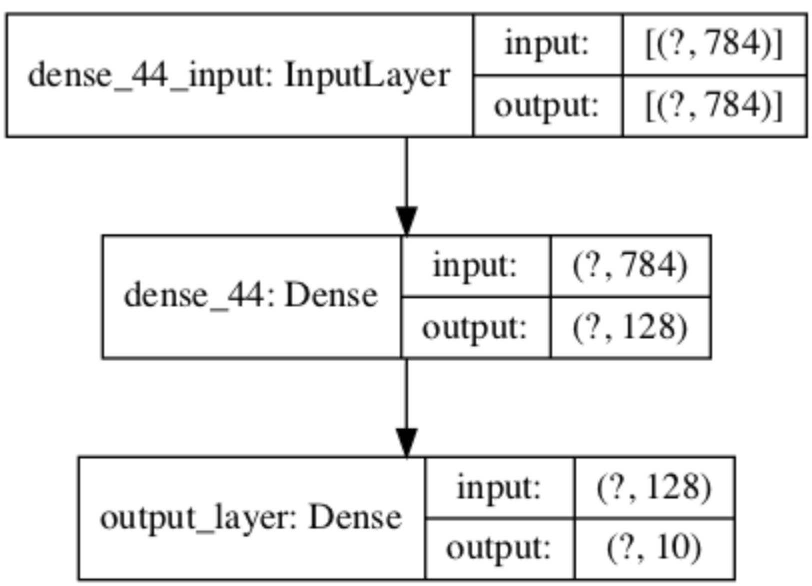
In [511]:
```
model = Sequential([
    Dense(input_shape=[784], units = 128, activation = tf.nn.relu),
    Dense(name = "output_layer", units = 10, activation = tf.nn.soft
])
```

In [512]:
```
model.summary()
```

```
Model: "sequential_44"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_44 (Dense)             (None, 128)               100480
_________________________________________________________________
output_layer (Dense)         (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

In [513]:
```
keras.utils.plot_model(model, "mnist_model.png", show_shapes=True)
```

Out[513]:



In [514]:
```
model.compile(optimizer='rmsprop',
              loss = 'categorical_crossentropy',
              metrics=['accuracy'])
```

In [515]:
```
t1 = time.time()
historytrain = model.fit(
    x_train_norm,
    y_train_encoded,
    epochs = 10,
    validation_split=0.20
    )
time1 = time.time() - t1
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [==============================] - 10s 218us/sample - l
oss: 0.2892 - accuracy: 0.9170 - val_loss: 0.1606 - val_accuracy:
0.9530
Epoch 2/10
48000/48000 [==============================] - 9s 184us/sample - lo
ss: 0.1366 - accuracy: 0.9594 - val_loss: 0.1278 - val_accuracy: 0.
9608
Epoch 3/10
48000/48000 [==============================] - 10s 204us/sample - l
oss: 0.0994 - accuracy: 0.9714 - val_loss: 0.1192 - val_accuracy:
0.9661
Epoch 4/10
48000/48000 [==============================] - 10s 205us/sample - l
oss: 0.0793 - accuracy: 0.9771 - val_loss: 0.0990 - val_accuracy:
0.9719
Epoch 5/10
48000/48000 [==============================] - 9s 179us/sample - lo
ss: 0.0652 - accuracy: 0.9820 - val_loss: 0.1022 - val_accuracy: 0.
9735
Epoch 6/10
48000/48000 [==============================] - 9s 180us/sample - lo
ss: 0.0558 - accuracy: 0.9845 - val_loss: 0.1012 - val_accuracy: 0.
9723
Epoch 7/10
48000/48000 [==============================] - 8s 167us/sample - lo
ss: 0.0480 - accuracy: 0.9865 - val_loss: 0.1011 - val_accuracy: 0.
9750
Epoch 8/10
48000/48000 [==============================] - 8s 166us/sample - lo
ss: 0.0420 - accuracy: 0.9887 - val_loss: 0.1105 - val_accuracy: 0.
9747
Epoch 9/10
48000/48000 [==============================] - 8s 168us/sample - lo
ss: 0.0368 - accuracy: 0.9900 - val_loss: 0.1073 - val_accuracy: 0.
9750
Epoch 10/10
48000/48000 [==============================] - 8s 166us/sample - lo
ss: 0.0319 - accuracy: 0.9910 - val_loss: 0.1172 - val_accuracy: 0.
9747
```

In [516]:
```
loss, accuracy = model.evaluate(x_test_norm, y_test_encoded)
print('test set accuracy: ', accuracy * 100)
```

```
10000/1 [==========================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
```

In [517]:
```
trainloss, trainaccuracy = model.evaluate(x_train_norm, y_train_enco
print('train set accuracy: ', trainaccuracy * 100)
```

```
60000/1 [==========================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
```

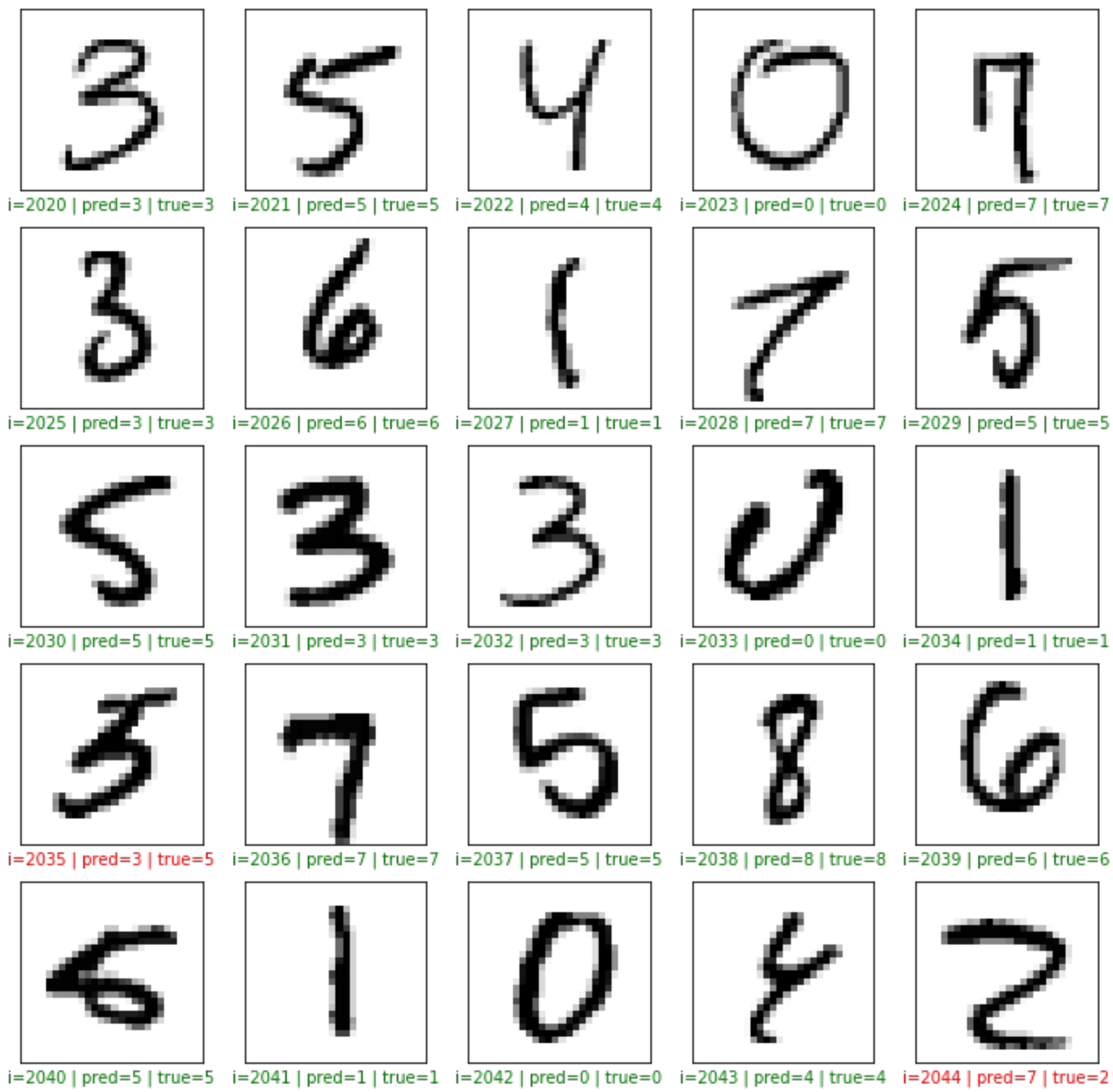
In [518]:
```
preds = model.predict(x_test_norm)
print('shape of preds: ', preds.shape)
```

```
shape of preds:  (10000, 10)
```

In [519]:
```
plt.figure(figsize = (12, 12))
```

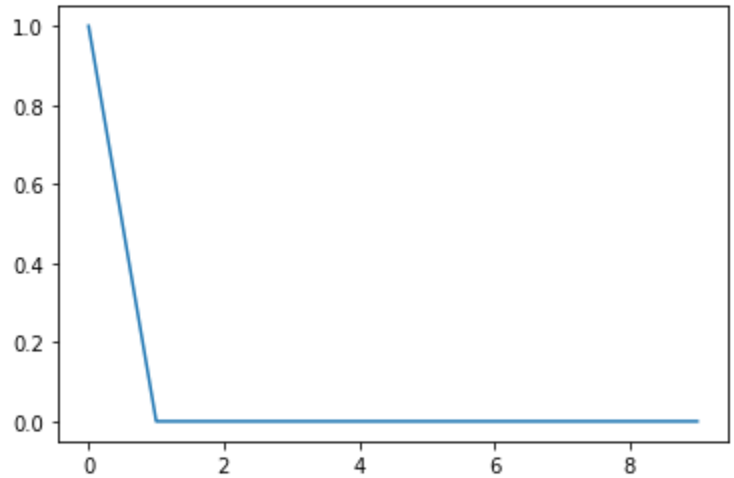
```
start_index = 2020

for i in range(25):
    plt.subplot(5, 5, i + 1)
    plt.grid(False)
    plt.xticks([])
    plt.yticks([])
    pred = np.argmax(preds[start_index + i])
    actual = np.argmax(y_test_encoded[start_index + i])
    col = 'g'
    if pred != actual:
        col = 'r'
    plt.xlabel('i={} | pred={} | true={}'.format(start_index + i, pr
    plt.imshow(x_test[start_index + i], cmap='binary')
plt.show()
```



In [520]:
```
"""
Enter the index value in place of the value 17 below for the predict
that you want to plot the probability scores for
"""
index = 2042
```

```
plt.plot(preds[index])
plt.show()
```

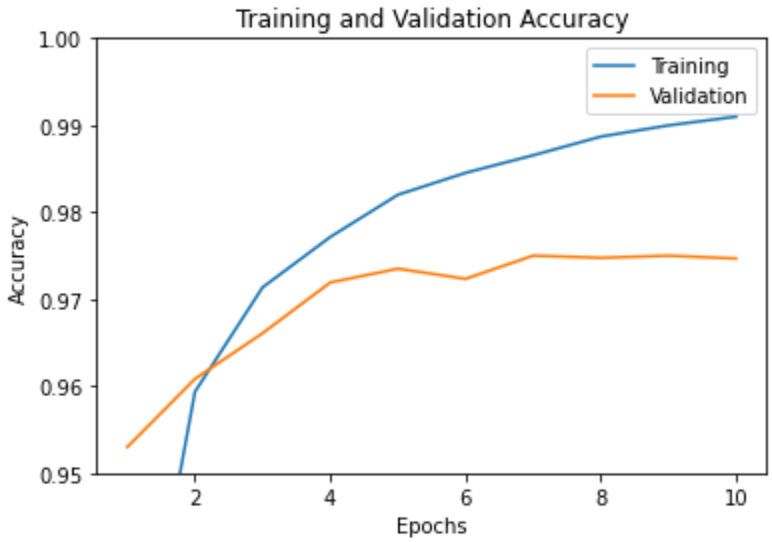


In [521]:
```
history_dict = history.history
history_dict.keys()
```

Out[521]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [522]:
```
acctrain = historytrain.history['accuracy']
val_acctrain = historytrain.history['val_accuracy']
losstrain = historytrain.history['loss']
val_losstrain = historytrain.history['val_loss']
```

In [523]:
```
plt.plot(range(1, len(acctrain) + 1), historytrain.history['accuracy
plt.plot(range(1, len(val_acctrain) + 1), historytrain.history['val_
plt.ylim([0.95, 1.0])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [524]:
```
# Get the predicted classes:
# pred_classes = model.predict_classes(x_train_norm)# give deprecati
pred_classes = np.argmax(model.predict(x_test_norm), axis=-1)
pred_classes
```

Out[524]:
```
array([7, 2, 1, ..., 4, 5, 6])
```

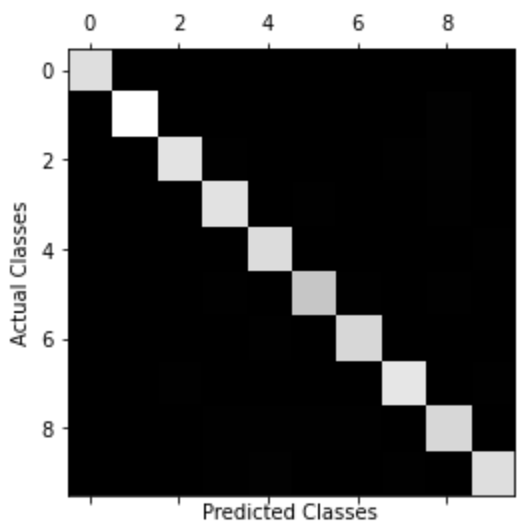
In [525]:
```
conf_mx = tf.math.confusion_matrix(y_test, pred_classes)
conf_mx
```

Out[525]:

```
<tf.Tensor: id=2929343, shape=(10, 10), dtype=int32, numpy=
array([[ 971,    0,    1,    2,    1,    1,    1,    1,    2,
0],
       [    0, 1116,    4,    1,    0,    1,    2,    2,    9,
```

In [526]:
```
def plot_confusion_matrix(matrix):
    """If you prefer color and a colorbar"""
    fig = plt.figure(figsize=(8,8))
    ax = fig.add_subplot(111)
    cax = ax.matshow(matrix)
    fig.colorbar(cax)
```

In [527]:
```
plt.matshow(conf_mx, cmap=plt.cm.gray)
plt.xlabel("Predicted Classes")
plt.ylabel("Actual Classes")
plt.show()
```



In [528]:
```

def plot_digits(instances, pos, images_per_row=5, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    images = [instance.reshape(size,size) for instance in instances]
    n_rows = (len(instances) - 1) // images_per_row + 1
    row_images = []
    n_empty = n_rows * images_per_row - len(instances)
    images.append(np.zeros((size, size * n_empty)))
    for row in range(n_rows):
        rimages = images[row * images_per_row : (row + 1) * images_p
        row_images.append(np.concatenate(rimages, axis=1))
    image = np.concatenate(row_images, axis=0)
    pos.imshow(image, cmap = 'binary', **options)
    pos.axis("off")
```

In [529]:
```
cl_a, cl_b = 4, 9
X_aa = x_train_norm[(y_train == cl_a) & (pred_classes == cl_a)]
X_ab = x_train_norm[(y_train == cl_a) & (pred_classes == cl_b)]
```

```
X_ba = x_train_norm[(y_train == cl_b) & (pred_classes == cl_a)]
X_bb = x_train_norm[(y_train == cl_b) & (pred_classes == cl_b)]

plt.figure(figsize=(6,6))

p1 = plt.subplot(221)
p2 = plt.subplot(222)
p3 = plt.subplot(223)
p4 = plt.subplot(224)

plot_digits(X_aa[:25], p1, images_per_row=5);
plot_digits(X_ab[:25], p2, images_per_row=5);
plot_digits(X_ba[:25], p3, images_per_row=5);
plot_digits(X_bb[:25], p4, images_per_row=5);


p1.set_title(f"{cl_a}'s classified as {cl_a}'s")
p2.set_title(f"{cl_a}'s classified as {cl_b}'s")
p3.set_title(f"{cl_b}'s classified as {cl_a}'s")
p4.set_title(f"{cl_b}'s classified as {cl_b}'s")

# plt.savefig("error_analysis_digits_plot_EXP1_valid")

plt.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-529-a05f5d2a4a28> in <module>
      1 cl_a, cl_b = 4, 9
----> 2 X_aa = x_train_norm[(y_train == cl_a) & (pred_classes == cl_a)]
      3 X_ab = x_train_norm[(y_train == cl_a) & (pred_classes == cl_b)]
      4 X_ba = x_train_norm[(y_train == cl_b) & (pred_classes == cl_a)]
      5 X_bb = x_train_norm[(y_train == cl_b) & (pred_classes == cl_b)]

ValueError: operands could not be broadcast together with shapes (60000,) (10000,)
```

MODEL 2

In [530]:

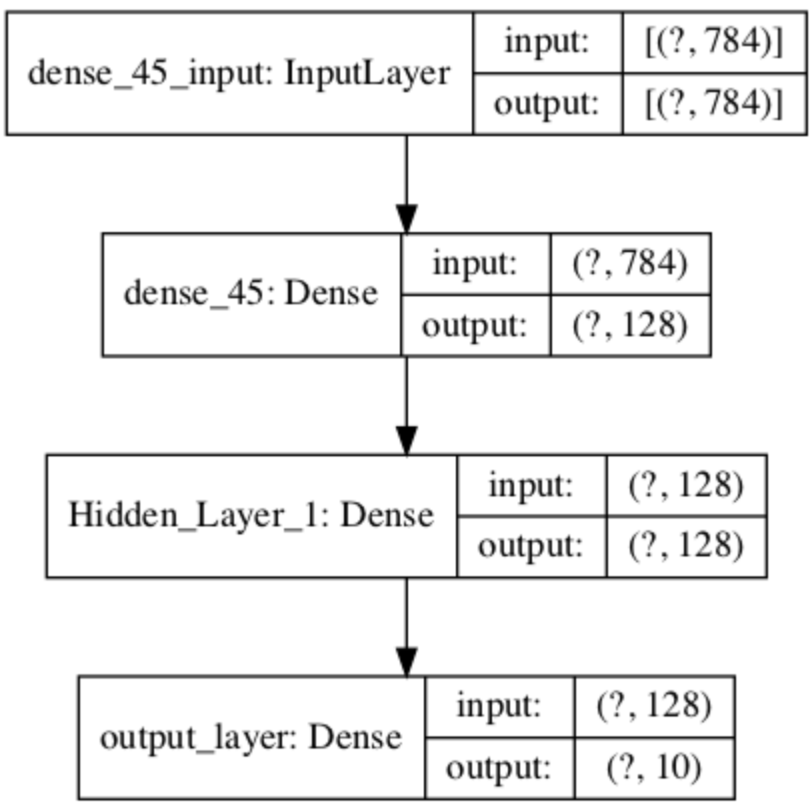
```
model2 = Sequential([
    Dense(input_shape=[784], units = 128, activation = tf.nn.relu),
    Dense(name = "Hidden_Layer_1", units = 128, activation = tf.nn.r
    Dense(name = "output_layer", units = 10, activation = tf.nn.soft
])
```

In [531]:
```
model2.summary()
```

```
Model: "sequential_45"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_45 (Dense)             (None, 128)               100480    
_________________________________________________________________
Hidden_Layer_1 (Dense)       (None, 128)               16512     
_________________________________________________________________
output_layer (Dense)         (None, 10)                1290      
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
_________________________________________________________________
```

In [532]:
```
keras.utils.plot_model(model2, "mnist_model.png", show_shapes=True)
```

Out[532]:



In [533]:
```
model2.compile(optimizer='rmsprop',
               loss = 'categorical_crossentropy',
               metrics=['accuracy'])
```

In [534]:
```
t1 = time.time()
history2train = model2.fit(
```

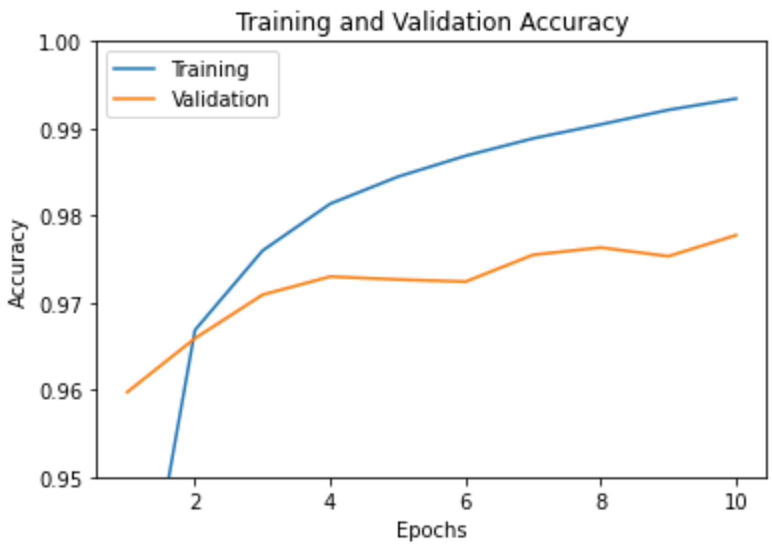
```
    x_train_norm,
    y_train_encoded,
    epochs = 10,
    validation_split=0.20
    )
time2 = time.time() -t1
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [==============================] - 11s 222us/sample - l
oss: 0.2593 - accuracy: 0.9232 - val_loss: 0.1373 - val_accuracy:
0.9597
Epoch 2/10
48000/48000 [==============================] - 10s 213us/sample - l
oss: 0.1127 - accuracy: 0.9669 - val_loss: 0.1281 - val_accuracy:
0.9659
Epoch 3/10
48000/48000 [==============================] - 10s 206us/sample - l
oss: 0.0812 - accuracy: 0.9760 - val_loss: 0.1103 - val_accuracy:
0.9709
Epoch 4/10
48000/48000 [==============================] - 9s 196us/sample - lo
ss: 0.0641 - accuracy: 0.9814 - val_loss: 0.1055 - val_accuracy: 0.
9730
Epoch 5/10
48000/48000 [==============================] - 11s 228us/sample - l
oss: 0.0542 - accuracy: 0.9845 - val_loss: 0.1290 - val_accuracy:
0.9727
Epoch 6/10
48000/48000 [==============================] - 10s 216us/sample - l
oss: 0.0462 - accuracy: 0.9869 - val_loss: 0.1248 - val_accuracy:
0.9724
Epoch 7/10
48000/48000 [==============================] - 12s 246us/sample - l
oss: 0.0387 - accuracy: 0.9889 - val_loss: 0.1283 - val_accuracy:
0.9755
Epoch 8/10
48000/48000 [==============================] - 9s 191us/sample - lo
ss: 0.0344 - accuracy: 0.9905 - val_loss: 0.1417 - val_accuracy: 0.
9763
Epoch 9/10
48000/48000 [==============================] - 9s 190us/sample - lo
ss: 0.0290 - accuracy: 0.9921 - val_loss: 0.1412 - val_accuracy: 0.
9753
Epoch 10/10
48000/48000 [==============================] - 9s 188us/sample - lo
ss: 0.0255 - accuracy: 0.9934 - val_loss: 0.1354 - val_accuracy: 0.
9778
```

In [535]:

```
acctrain = history2train.history['accuracy']
val_acctrain = history2train.history['val_accuracy']
losstrain = history2train.history['loss']
val_losstrain = history2train.history['val_loss']
```

In [536]:
```
plt.plot(range(1, len(acctrain) + 1), history2train.history['accurac
plt.plot(range(1, len(val_acc) + 1), history2train.history['val_accu
plt.ylim([0.95, 1.0])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [537]:
```
pred_classes = np.argmax(model2.predict(x_test_norm), axis=-1)
pred_classes
```

Out[537]: array([7, 2, 1, ..., 4, 5, 6])

In [538]:
```
conf_mx = tf.math.confusion_matrix(y_test, pred_classes)
conf_mx
```

Out[538]:

```
<tf.Tensor: id=2987067, shape=(10, 10), dtype=int32, numpy=
array([[ 971,    2,    0,    1,    1,    0,    3,    1,    1,
```

In [539]:
```
loss2, accuracy2 = model2.evaluate(x_test_norm, y_test_encoded)
print('test set accuracy: ', accuracy2 * 100)
```

```
10000/1 [==========================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
```

In [540]:
```
trainloss2, trainaccuracy2 = model2.evaluate(x_train_norm, y_train_e
print('train set accuracy: ', trainaccuracy2 * 100)
```

```
60000/1 [==========================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
```

MODEL 3

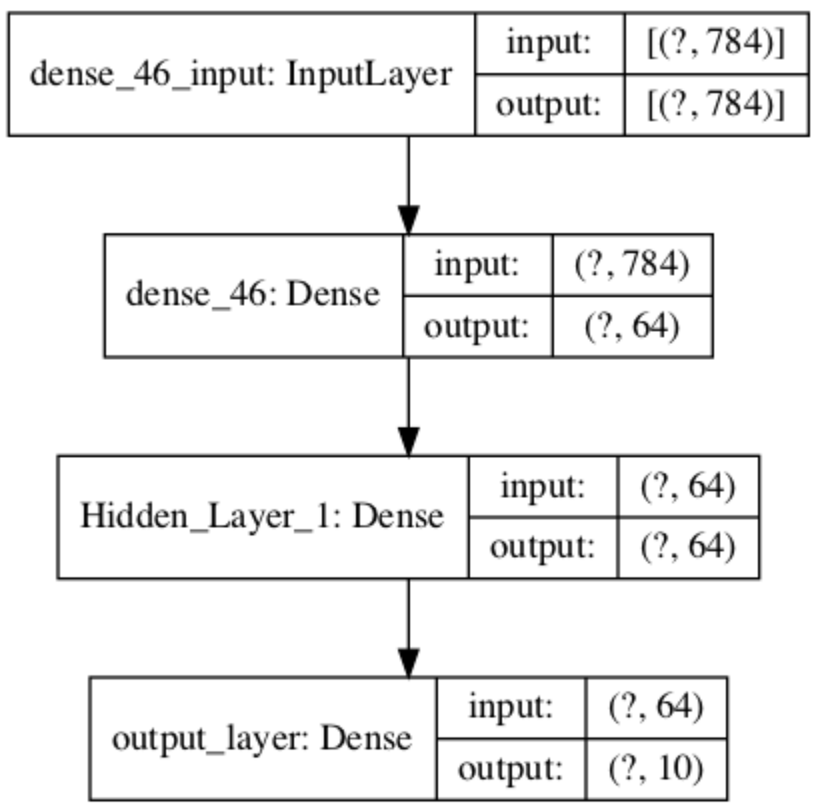
In [541]:
```
model3 = Sequential([
    Dense(input_shape=[784], units = 64, activation = tf.nn.relu),
    Dense(name = "Hidden_Layer_1", units = 64, activation = tf.nn.re
    Dense(name = "output_layer", units = 10, activation = tf.nn.soft
])
```

In [542]:
```
model3.summary()
```

```
Model: "sequential_46"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_46 (Dense)             (None, 64)                50240
_________________________________________________________________
Hidden_Layer_1 (Dense)       (None, 64)                4160
_________________________________________________________________
output_layer (Dense)         (None, 10)                650
=================================================================
Total params: 55,050
Trainable params: 55,050
Non-trainable params: 0
_________________________________________________________________
```

In [543]:
```
keras.utils.plot_model(model3, "mnist_model.png", show_shapes=True)
```

Out[543]:



In [544]:
```
model3.compile(optimizer='rmsprop',
```

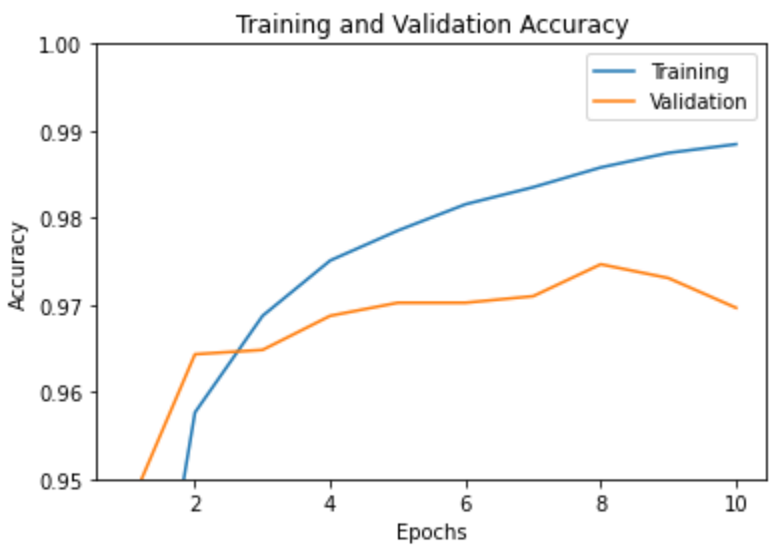
```
                    loss = 'categorical_crossentropy',
                    metrics=['accuracy'])
```

In [545]:
```
t1 = time.time()
history3train = model3.fit(
    x_train_norm,
    y_train_encoded,
    epochs = 10,
    validation_split=0.20
    )
time3 = time.time() - t1
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [==============================] - 10s 203us/sample - l
oss: 0.3079 - accuracy: 0.9113 - val_loss: 0.1837 - val_accuracy:
0.9462
Epoch 2/10
48000/48000 [==============================] - 9s 188us/sample - lo
ss: 0.1419 - accuracy: 0.9576 - val_loss: 0.1213 - val_accuracy: 0.
9643
Epoch 3/10
48000/48000 [==============================] - 9s 181us/sample - lo
ss: 0.1049 - accuracy: 0.9688 - val_loss: 0.1277 - val_accuracy: 0.
9648
Epoch 4/10
48000/48000 [==============================] - 8s 164us/sample - lo
ss: 0.0854 - accuracy: 0.9751 - val_loss: 0.1146 - val_accuracy: 0.
9688
Epoch 5/10
48000/48000 [==============================] - 8s 157us/sample - lo
ss: 0.0729 - accuracy: 0.9785 - val_loss: 0.1195 - val_accuracy: 0.
9703
Epoch 6/10
48000/48000 [==============================] - 8s 158us/sample - lo
ss: 0.0628 - accuracy: 0.9816 - val_loss: 0.1163 - val_accuracy: 0.
9703
Epoch 7/10
48000/48000 [==============================] - 10s 201us/sample - l
oss: 0.0559 - accuracy: 0.9835 - val_loss: 0.1262 - val_accuracy:
0.9710
Epoch 8/10
48000/48000 [==============================] - 9s 197us/sample - lo
ss: 0.0492 - accuracy: 0.9858 - val_loss: 0.1225 - val_accuracy: 0.
9747
Epoch 9/10
48000/48000 [==============================] - 8s 161us/sample - lo
ss: 0.0445 - accuracy: 0.9875 - val_loss: 0.1289 - val_accuracy: 0.
9731
Epoch 10/10
48000/48000 [==============================] - 9s 182us/sample - lo
ss: 0.0403 - accuracy: 0.9885 - val_loss: 0.1383 - val_accuracy: 0.
9697
```

In [546]:
```
acctrain = history3train.history['accuracy']
val_acctrain = history3train.history['val_accuracy']
losstrain = history3train.history['loss']
val_losstrain = history3train.history['val_loss']
```

In [547]:
```
plt.plot(range(1, len(acctrain) + 1), history3train.history['accurac
plt.plot(range(1, len(val_acctrain) + 1), history3train.history['val
plt.ylim([0.95, 1.0])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [548]:
```
loss3, accuracy3 = model3.evaluate(x_test_norm, y_test_encoded)
print('test set accuracy: ', accuracy3 * 100)
```

```
10000/1 [==============================================================
--------------------------------------------------------------
```

In [549]:
```
trainloss3, trainaccuracy3 = model3.evaluate(x_train_norm, y_train_e
print('train set accuracy: ', trainaccuracy3 * 100)
```

```
60000/1 [==============================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
--------------------------------------------------------------
```

In [550]:
```
pred_classes = np.argmax(model3.predict(x_test_norm), axis=-1)
pred_classes
```

Out[550]:
```
array([7, 2, 1, ..., 4, 5, 6])
```

In [551]:
```
conf_mx = tf.math.confusion_matrix(y_test, pred_classes)
conf_mx
```

Out[551]:

```
<tf.Tensor: id=3058363, shape=(10, 10), dtype=int32, numpy=
```

Model 4

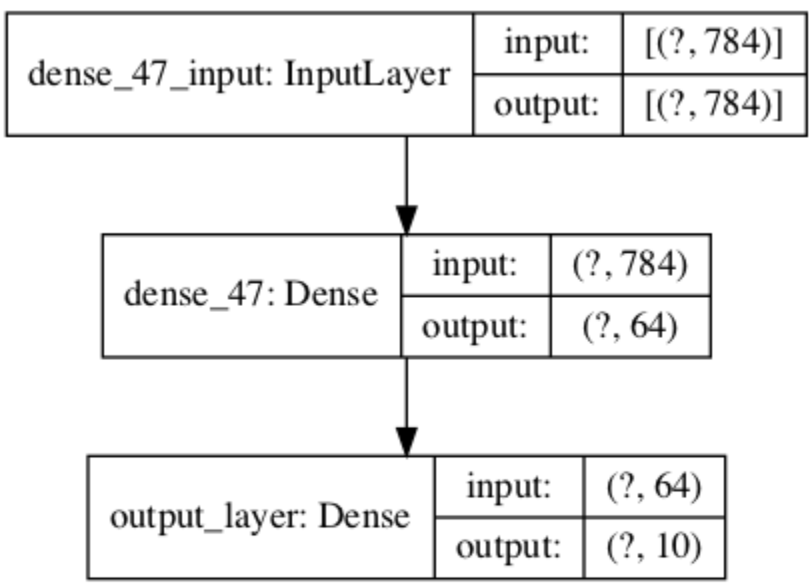
In [552]:
```
model4 = Sequential([
    Dense(input_shape=[784], units = 64, activation = tf.nn.relu),
    Dense(name = "output_layer", units = 10, activation = tf.nn.soft
])
```

In [553]:
```
model4.summary()
```

```
Model: "sequential_47"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_47 (Dense)             (None, 64)                50240
_________________________________________________________________
output_layer (Dense)         (None, 10)                650
=================================================================
Total params: 50,890
Trainable params: 50,890
Non-trainable params: 0
_________________________________________________________________
```

In [554]:
```
keras.utils.plot_model(model4, "mnist_model.png", show_shapes=True)
```

Out[554]:



In [555]:
```
model4.compile(optimizer='rmsprop',
               loss = 'categorical_crossentropy',
               metrics=['accuracy'])
```

In [556]:
```
t1 = time.time()
```

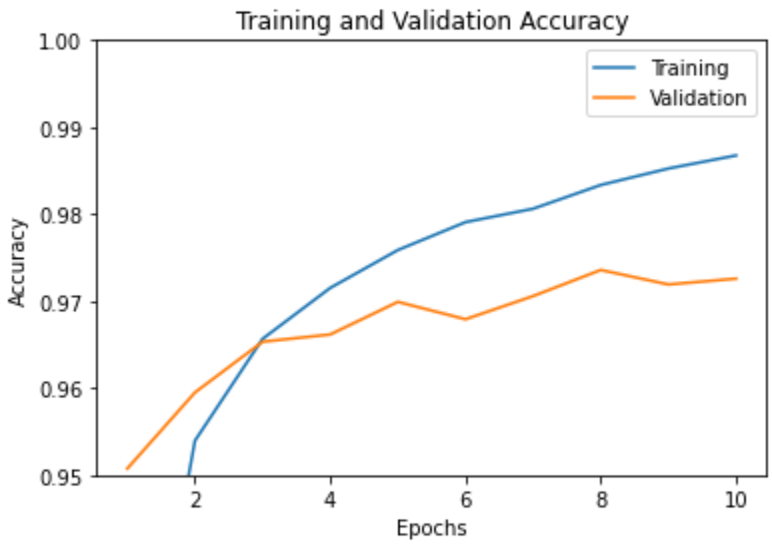
```
history4train = model4.fit(
    x_train_norm,
    y_train_encoded,
    epochs = 10,
    validation_split=0.20
    )
time4 = time.time() - t1
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [==============================] - 8s 172us/sample - lo
ss: 0.3158 - accuracy: 0.9111 - val_loss: 0.1727 - val_accuracy: 0.
9507
Epoch 2/10
48000/48000 [==============================] - 8s 163us/sample - lo
ss: 0.1571 - accuracy: 0.9540 - val_loss: 0.1413 - val_accuracy: 0.
9595
Epoch 3/10
48000/48000 [==============================] - 8s 169us/sample - lo
ss: 0.1185 - accuracy: 0.9657 - val_loss: 0.1227 - val_accuracy: 0.
9653
Epoch 4/10
48000/48000 [==============================] - 7s 150us/sample - lo
ss: 0.0974 - accuracy: 0.9715 - val_loss: 0.1173 - val_accuracy: 0.
9662
Epoch 5/10
48000/48000 [==============================] - 8s 160us/sample - lo
ss: 0.0836 - accuracy: 0.9759 - val_loss: 0.1057 - val_accuracy: 0.
9699
Epoch 6/10
48000/48000 [==============================] - 7s 156us/sample - lo
ss: 0.0737 - accuracy: 0.9791 - val_loss: 0.1174 - val_accuracy: 0.
9679
Epoch 7/10
48000/48000 [==============================] - 8s 163us/sample - lo
ss: 0.0670 - accuracy: 0.9806 - val_loss: 0.1077 - val_accuracy: 0.
9706
Epoch 8/10
48000/48000 [==============================] - 8s 163us/sample - lo
ss: 0.0601 - accuracy: 0.9834 - val_loss: 0.1023 - val_accuracy: 0.
9736
Epoch 9/10
48000/48000 [==============================] - 7s 154us/sample - lo
ss: 0.0534 - accuracy: 0.9852 - val_loss: 0.1073 - val_accuracy: 0.
9719
Epoch 10/10
48000/48000 [==============================] - 9s 186us/sample - lo
ss: 0.0493 - accuracy: 0.9868 - val_loss: 0.1119 - val_accuracy: 0.
9726
```

In [557]:

```
acctrain = history4train.history['accuracy']
val_acc = history4train.history['val_accuracy']
loss = history4train.history['loss']
val_loss = history4train.history['val_loss']
```

In [558]:
```
plt.plot(range(1, len(acctrain) + 1), history4train.history['accurac
plt.plot(range(1, len(val_acctrain) + 1), history4train.history['val_
plt.ylim([0.95, 1.0])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [559]:
```
loss4, accuracy4 = model4.evaluate(x_test_norm, y_test_encoded)
print('test set accuracy: ', accuracy4 * 100)
```

```
10000/1 [==========================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
```

In [560]:
```
trainloss4, trainaccuracy4 = model4.evaluate(x_train_norm, y_train_e
print('train set accuracy: ', trainaccuracy4 * 100)
```

```
60000/1 [==========================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
```

In [561]:
```
pred_classes = np.argmax(model4.predict(x_test_norm), axis=-1)
pred_classes
```

Out[561]:
```
array([7, 2, 1, ..., 4, 5, 6])
```

In [562]:
```
conf_mx = tf.math.confusion_matrix(y_test, pred_classes)
conf_mx
```

Out[562]:
```
<tf.Tensor: id=3122722, shape=(10, 10), dtype=int32, numpy=
array([[ 970,    0,    1,    0,    1,    2,    4,    1,    1,
0],
       [   0, 1119,    3,    1,    0,    1,    5,    0,    6,
0],
       [   6,    1,  989,    9,    3,    0,    4,    7,   13,
0],
       [   1,    0,    2,  987,    0,    6,    0,    8,    6,
0],
       [   2,    0,    4,    0,  972,    0,    1,    0,    1,
2],
       [   5,    1,    0,    8,    3,  854,   11,    0,    7,
3],
       [   4,    2,    0,    0,    5,    6,  939,    0,    2,
0],
       [   1,    2,    6,    8,    4,    0,    0,  996,    3,
8],
       [   6,    0,    4,    7,    8,    3,    4,    2,  937,
3],
       [   3,    3,    1,    7,   19,    3,    1,    4,    5,   96
3]], dtype=int32)>
```

MODEL 5

In [563]:
```
model5 = Sequential([
    Dense(input_shape=[784], units = 256, activation = tf.nn.relu),
    Dense(name = "output_layer", units = 10, activation = tf.nn.soft
])
```

In [564]:
```
model5.compile(optimizer='rmsprop',
               loss = 'categorical_crossentropy',
               metrics=['accuracy'])
```

In [565]:
```
t1 = time.time()
history5train = model5.fit(
    x_train_norm,
    y_train_encoded,
    epochs = 10,
    validation_split=0.20
    )
time5 = time.time() - t1
```

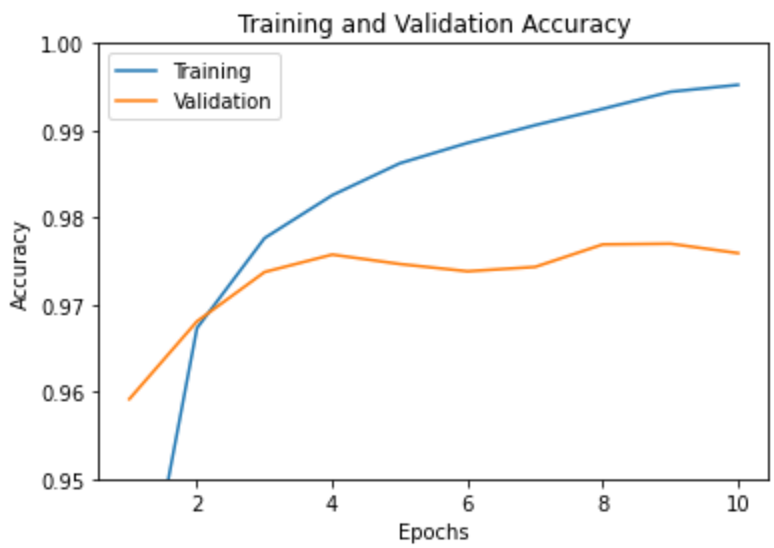
```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [==============================] - 13s 262us/sample - l
oss: 0.2509 - accuracy: 0.9263 - val_loss: 0.1366 - val_accuracy:
0.9592
Epoch 2/10
48000/48000 [==============================] - 11s 221us/sample - l
oss: 0.1111 - accuracy: 0.9673 - val_loss: 0.1044 - val_accuracy:
```

In [566]:

```
acctrain = history5train.history['accuracy']
val_acctrain = history5train.history['val_accuracy']
losstrain = history5train.history['loss']
val_losstrain = history5train.history['val_loss']
```

In [567]:

```
plt.plot(range(1, len(acctrain) + 1), history5train.history['accurac
plt.plot(range(1, len(val_acctrain) + 1), history5train.history['val
plt.ylim([0.95, 1.0])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [568]:

```
loss5, accuracy5 = model5.evaluate(x_test_norm, y_test_encoded)
print('test set accuracy: ', accuracy5 * 100)
```

```
10000/1 [==========================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
```

In [569]:

```python
trainloss5, trainaccuracy5 = model5.evaluate(x_train_norm, y_train_e
print('train set accuracy: ', trainaccuracy5 * 100)
```

```
60000/1 [==========================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
==================================================================
------------------------------------------------------------------
```

In [570]:

```python
pred_classes = np.argmax(model5.predict(x_test_norm), axis=-1)
pred_classes
```

Out[570]:

```
array([7, 2, 1, ..., 4, 5, 6])
```

In [571]:

```python
conf_mx = tf.math.confusion_matrix(y_test, pred_classes)
conf_mx
```

Out[571]:

```
<tf.Tensor: id=3187081, shape=(10, 10), dtype=int32, numpy=
array([[ 970,    0,    1,    0,    2,    1,    3,    1,    2,
0],
       [    0, 1124,    3,    1,    0,    1,    2,    2,    2,
0],
       [    4,    2, 1004,    2,    3,    0,    2,    7,    8,
0],
       [    0,    0,    2,  997,    0,    2,    0,    4,    2,
3],
       [    0,    0,    2,    1,  970,    0,    4,    1,    0,
```

MODEL 6

In [572]:

```
model6 = Sequential([
    Dense(input_shape=[784], units = 256, activation = tf.nn.relu),
    Dense(name = "Hidden_Layer_1", units = 256, activation = tf.nn.r
    Dense(name = "output_layer", units = 10, activation = tf.nn.soft
])
```

In [573]:

```
model6.compile(optimizer='rmsprop',
               loss = 'categorical_crossentropy',
               metrics=['accuracy'])
```

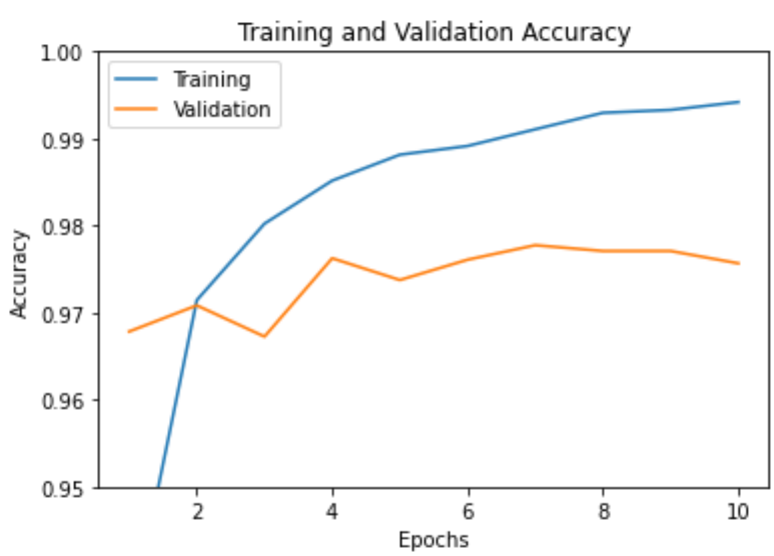
In [574]:

```
t1 = time.time()
history6 = model6.fit(
    x_train_norm,
    y_train_encoded,
    epochs = 10,
    validation_split=0.20
    )
time6 = time.time() - t1
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
48000/48000 [==============================] - 17s 356us/sample - l
oss: 0.2180 - accuracy: 0.9337 - val_loss: 0.1093 - val_accuracy:
0.9678
Epoch 2/10
48000/48000 [==============================] - 13s 279us/sample - l
oss: 0.0981 - accuracy: 0.9714 - val_loss: 0.1030 - val_accuracy:
0.9708
Epoch 3/10
48000/48000 [==============================] - 13s 280us/sample - l
oss: 0.0722 - accuracy: 0.9802 - val_loss: 0.1397 - val_accuracy:
0.9672
Epoch 4/10
48000/48000 [==============================] - 14s 287us/sample - l
oss: 0.0551 - accuracy: 0.9852 - val_loss: 0.1188 - val_accuracy:
0.9762
Epoch 5/10
48000/48000 [==============================] - 14s 289us/sample - l
oss: 0.0472 - accuracy: 0.9881 - val_loss: 0.1499 - val_accuracy:
0.9737
Epoch 6/10
48000/48000 [==============================] - 14s 283us/sample - l
```

In [575]:
```
acctrain = history6.history['accuracy']
val_acctrain = history6.history['val_accuracy']
losstrain = history6.history['loss']
val_losstrain = history6.history['val_loss']
```

In [576]:
```
plt.plot(range(1, len(acctrain) + 1), history6.history['accuracy'],
plt.plot(range(1, len(val_acctrain) + 1), history6.history['val_accu
plt.ylim([0.95, 1.0])
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [577]:
```
loss6, accuracy6 = model6.evaluate(x_test_norm, y_test_encoded)
print('test set accuracy: ', accuracy6 * 100)
```

```
10000/1 [==========================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
-------------------------------------------------------------------
```

In [578]:
```
trainloss6, trainaccuracy6 = model6.evaluate(x_train_norm, y_train_e
print('train set accuracy: ', trainaccuracy6 * 100)
```

```
60000/1 [==========================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
===================================================================
-------------------------------------------------------------------
```

In [579]:
```
pred_classes = np.argmax(model6.predict(x_test_norm), axis=-1)
pred_classes
```

Out[579]:
```
array([7, 2, 1, ..., 4, 5, 6])
```

In [580]:
```
conf_mx = tf.math.confusion_matrix(y_test, pred_classes)
```

```
conf_mx
```

Out[580]:

```
<tf.Tensor: id=3251591, shape=(10, 10), dtype=int32, numpy=
array([[ 968,    1,    0,    1,    2,    2,    3,    1,    1,
1],
       [   0, 1118,    2,    2,    0,    1,    3,    3,    5,
1],
       [   6,    0,  999,    4,    5,    0,    1,    9,    8,
0],
       [   0,    0,    2,  990,    0,    8,    0,    5,    4,
1],
       [   1,    0,    2,    0,  965,    0,    2,    1,    1,   1
0],
       [   2,    0,    0,    6,    1,  873,    3,    0,    3,
4],
       [   4,    2,    0,    0,   20,    3,  926,    0,    2,
1],
       [   0,    0,    5,    3,    4,    1,    0, 1005,    5,
5],
       [   3,    0,    3,    2,    5,    3,    3,    6,  945,
4],
       [   1,    1,    0,    4,   15,    2,    0,    6,    4,   97
6]], dtype=int32)>
```

In [581]:

```
Experiments = [1, 2, 3, 4, 5, 6]
training  = [trainaccuracy, trainaccuracy2, trainaccuracy3, trainacc
times = [time1, time2, time3, time4, time5, time6]
Nodes = [128, 128, 64, 64, 256, 256]
Layers = [1, 2, 2, 1, 1, 2]
testaccuracy = [accuracy, accuracy2, accuracy3, accuracy4, accuracy5
Results = pd.DataFrame({"Experiment": Experiments, "Processing Times
Results
```

Out[581]:

	Experiment	Processing Times	Nodes Per Layer	Layers	Train Set Accuracy	Test Set Accuracy
0	1	88.251570	128	1	0.990050	0.9762
1	2	100.667954	128	2	0.993217	0.9791
2	3	86.110318	64	2	0.985783	0.9737
3	4	78.631330	64	1	0.985767	0.9726
4	5	120.206586	256	1	0.992700	0.9784
5	6	139.119087	256	2	0.992017	0.9765

In [ ]: