Objective 0

Importing the neccessary packages to pull in the two pickle datasets. Immediately reading the data into two different dataframes. Inspecting the shapes to ensure both have the same amount of columns.

In [5]:
```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

In [6]:
```
import numpy as np
import pandas as pd
import os
import pickle
```

In [7]:
```
os.listdir()
```

Out[7]:
```
['.ipynb_checkpoints',
 'assign-3-part-1-test.pickle',
 'assign-3-part-1-train.pickle',
 'Muniz_Assignment_3_P1.ipynb']
```

The train and test data is imported. The shape, value count by data type and columns are displayed below.

In [8]:
```
train = pd.read_pickle('assign-3-part-1-train.pickle')
test = pd.read_pickle('assign-3-part-1-test.pickle')
```

In [9]:
```
train.shape
test.shape
```

Out[9]:
```
(40320, 786)
```

Out[9]:
```
(1680, 786)
```

In [10]:
```
train.dtypes.value_counts()
test.dtypes.value_counts()
```

Out[10]:
```
int64    786
dtype: int64
int64      785
```

Out[10]:
```
float64      1
dtype: int64
```

Ensuring below that the labels in the train set are correct. In the train set the labels should be 0 through 9 which is the case.

In [12]:
```
train['label'].max()
train['label'].min()
```

Out[12]: 9

Out[12]: 0

Checking the train set and test set to ensure that the pickles are withing [0,255]

In [61]:
```
tot_1 = []
for column in train:
    if  train[column].max() <= 255:
        tot_1 = 0
    else:
        tot_1 +1
print(tot_1)
```

Out[61]: 1

0

In [60]:
```
tot = []
for column in test:
    if  test[column].max() <= 255:
        tot = 0
    else:
        tot +1
print(tot)
```

0

Splitting the train dataset into a features and labels. x is the pixels while y is the lables.

In [15]:
```
y = train.label.to_numpy()
x = train.iloc[:,1:785]
x.shape
y.shape
```

Out[15]: (40320, 784)

Out[15]: (40320,)

rescaling the pixels so they are all between 0 and 1

In [16]:
```
from sklearn.preprocessing import MinMaxScaler
rescaler = MinMaxScaler()
x_rescaled = rescaler.fit_transform(x)
x_rescaled.max()
x_rescaled.min()
x_rescaled.shape
```

Out[16]: 1.0

Out[16]: 0.0

Out[16]: (40320, 784)

Creating a train, validation, and test set with a random 60,20,20 split. The first split is to split the train and test 80/20 this creates the train and test set. The second split is a train and val split 75/25. This will lead to an XTrain and yTrain that contains 60% of the data, an Xval and yVal that contains 20% and an XTest and yTest that contains 20%.

In [17]:
```
from sklearn.model_selection import train_test_split
XTrain, XTest, yTrain, yTest = train_test_split(x_rescaled,y,train_size = 0.60, test_size = 0.20, random_state=11)
XTrain,XVal,yTrain,yVal = train_test_split(XTrain,yTrain, train_size = 0.75, test_size = 0.25 )
```

Objective 1

In [18]:
```
import tensorflow as tf
from tensorflow import keras
```

Creating the first MLP, creating 2 layers with relu activation. For this model we will keep the number of neaurons consistent between both layers. The output layer has 10 nuerons as we have ten potential outputs, 0 through 9. The output activation layer is softmax.

In [19]:
```
model1_low = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[784]),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
```

```
    keras.layers.Dense(10, activation="softmax")
])
```

Compiling the model1 defined about to define the loss function, the optimizer and accuracy as the metric. I am using the sparse categorical corssentropy and my loss function due to the pixels not being one hot encoded.

In [21]:
```
model1_low.compile(loss="sparse_categorical_crossentropy",
              optimizer = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999),
              metrics=["accuracy"])
```

Fitting the MLP with the train data and I am using the val data to determine the accuracy of the model. I set epochs to 100 but placed an early stop with a patience of 5.

In [62]:
```
history = model1_low.fit(XTrain, yTrain, epochs=30,verbose = 0, validation_data=(XVal, yVal))
```

Below I am saving the final loss and accuracy metrics for the first MLP model model1

In [73]:
```
Model1_low_Accuracy = model1_low.evaluate(XTest, yTest)
```

```
252/252 [==============================] - 0s 2ms/step - loss: 0.2215 - accuracy: 0.9710
```

In [74]:
```
model1_low.save(r'C:\Users\jonah.muniz\OneDrive - Accenture\Masters Program\Practical Machine Learning\model1_low.h5
```

The model took 62 seconds to run.

Model1 was the low case in terms of number of layers. Now we will train a MLP with four layers instead of 2. I will keep the amount of neurons across all layers the same, 100.

In [26]:
```
model1_high = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[784]),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

Using the same hyperparameteres as model1_low to ensure the accuracys can be compared to truly see how the increase in the amount of layers effects accuracy.

In [28]:
```
model1_high.compile(loss="sparse_categorical_crossentropy",
              optimizer = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999),
              metrics=["accuracy"])
```

In [64]:
```
 history = model1_high.fit(XTrain, yTrain, epochs=30,verbose = 0, validation_data=(XVal, yVal))
```

In [71]:
```
Model1_high_Accuracy = model1_high.evaluate(XTest, yTest)
```

```
252/252 [==============================] - 1s 2ms/step - loss: 0.2287 - accuracy: 0.9725
```

In [72]:
```
model1_high.save(r'C:\Users\jonah.muniz\OneDrive - Accenture\Masters Program\Practical Machine Learning\model1_high.h
```

The model took 62 seconds to run.

These next models will contain the same amount of layers and the amount of neaurons per each layer will be changed to understand if a relatively low or high amount of nuerons will increase of decrease accuracy. The low version will have two hidden layers each with 50 neurons.

In [33]:
```
model2_low = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[784]),
    keras.layers.Dense(50, activation="relu"),
    keras.layers.Dense(50, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

I am using the same exact loss function, optimizer and metric to ensure all models can be compared to one and another.

In [35]:
```
model2_low.compile(loss="sparse_categorical_crossentropy",
              optimizer = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999),
              metrics=["accuracy"])
```

In [65]:
```
history2_low = model2_low.fit(XTrain, yTrain, epochs=30,verbose = 0, validation_data=(XVal, yVal))
```

In [69]:
```
Model2_low_Accuracy = model2_low.evaluate(XTest, yTest)
```

```
252/252 [==============================] - 0s 2ms/step - loss: 0.2507 - accuracy: 0.9676
```

In [70]:
```
model2_low.save(r'C:\Users\jonah.muniz\OneDrive - Accenture\Masters Program\Practical Machine Learning\model2_low.h5
```

The model took 56 seconds to run.

Below is the high neuron version of model 2. Each hidden layer will have 200 nuerons.

In [40]:
```
model2_high = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[784]),
    keras.layers.Dense(200, activation="relu"),
    keras.layers.Dense(200, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

Again, using the same loss functions, optimizer and metrics across all NN's to ensure all four can be compared.

In [42]:
```
model2_high.compile(loss="sparse_categorical_crossentropy",
              optimizer = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999),
              metrics=["accuracy"])
```

In [66]:
```
history2_high = model2_high.fit(XTrain, yTrain, epochs=30,verbose = 0, validation_data=(XVal, yVal))
```

In [67]:
```
Model2_high_Accuracy = model2_high.evaluate(XTest, yTest)
```

```
252/252 [==============================] - 0s 2ms/step - loss: 0.2527 - accuracy: 0.9757
```

In [68]:
```
model2_high.save(r'C:\Users\jonah.muniz\OneDrive - Accenture\Masters Program\Practical Machine Learning\model2_high.h
```

The model took 62 seconds to run.

Now that all four models have been compiled and trained, a table is created to compare the accuracy of the four models to the test set to determine the best overall model.

In [75]:
```
model_results = pd.DataFrame(data = {'Model 1 Low:Two Layers 100 Neurons': Model1_low_Accuracy,
                                     'Model 1 High:Four Layers 100 Neurons': Model1_high_Accuracy,
                                    'Model 2 Low: Two Layers 50 Neurons': Model2_low_Accuracy,
                                    'Model 2 High: Two Layers 200 Neurons': Model2_high_Accuracy}, index = ['Loss','A
model_results
```

Out[75]:

	Model 1 Low:Two Layers 100 Neurons	Model 1 High:Four Layers 100 Neurons	Model 2 Low: Two Layers 50 Neurons	Model 2 High: Two Layers 200 Neurons
Loss	0.221530	0.228685	0.250686	0.252672
Accuracy	0.970982	0.972470	0.967634	0.975694

To wrap up the above table. In this experiment we have two models varying one hyperparameter to see how it would effect overall loss and accuracy. For model 1 the number of layers was the variable I was changing. As can be seen above, increasing the number of layers in the MLP NN did not increase accuracy or slightly increased loss. For model 2, the number of layers was kept constant and the amount of neurons in each layer was changed. As can be seen above, as accuracy increased and loss descreased when you add more neurons to each layer. Across all four models the model that had the highest model was Model 2 High: Two Layers with 200 neurons with 97.6% accuracy. The fastest model to train was model 2 low which took 56 seconds to run while the other four took 62 seconds.

Objective 2

Model 1 will be loaded back into the notebook and will be used to predict the digits of the test images.

rescaling the test data like the train data was rescaled then using the rescaled test data to generate an array of predictions

In [55]:
```
test = test.drop(columns = ['label','imageID'])
```

Out[55]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pix
38840	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
6003	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
8977	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
27361	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
16614	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
35999	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
35298	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
1684	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
34412	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
27506	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	

1680 rows × 784 columns

In [56]:
```
rescaler = MinMaxScaler()
test_rescaled = rescaler.fit_transform(test)
test_rescaled.max()
test_rescaled.min()
test_rescaled.shape
```

Out[56]: 1.0

Out[56]: 0.0

Out[56]: (1680, 784)

In [58]:
```
from tensorflow.keras import models
best_model = models.load_model(r'C:\Users\jonah.muniz\OneDrive - Accenture\Masters Program\Practical Machine Learning
predictions = best_model.predict(test_rescaled)
```

In [ ]: