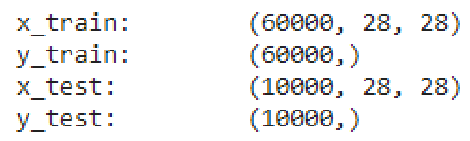
MSDS 422 Assignment 6
Husein Adenwala

Introduction

The Purpose of this assignment is to compare 4 neural net models (2x2 crossed design). The MNIST dataset of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples.

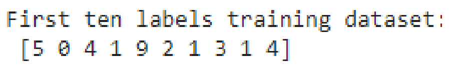
Data preparation, exploration, visualization

The train and test dataset are imported from the Keras dataset package and the data was conveniently divided into 60000 train and 10000 test numpy arrays of target variable (y), which contains label values from 0 to 9 and of features (x) which is in a 28 by 28 array and contains values from 0 to 255. The images below show the general structure of the datasets:



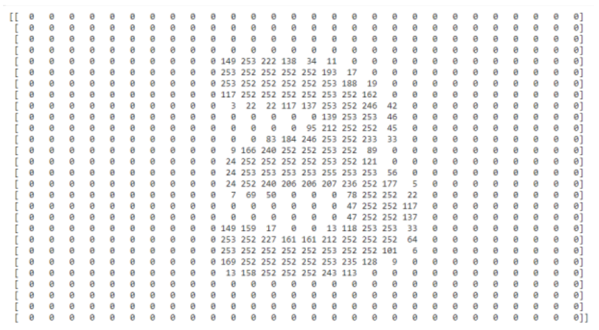
```
x_train:        (60000, 28, 28)
y_train:        (60000,)
x_test:         (10000, 28, 28)
y_test:         (10000,)
```

Example of Target variable



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

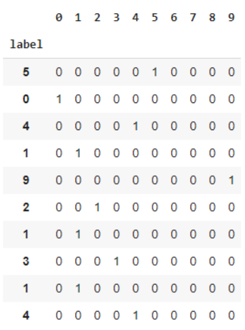
Example of 28 x 28 matrix of a feature instance



```
[[  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   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   0 149 253 222 138  34  11   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 253 252 252 252 252 193  17   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 253 252 252 252 252 253 188  19   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 117 252 252 252 252 253 252 162   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   3  22  22 117 137 253 252 246  42   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 139 253 253  46   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  95 212 252 252  45   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  83 184 246 253 252 233  33   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   9 166 240 252 252 253 252  89   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 252 252 252 252 253 252 121   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 253 253 253 253 255 253 253  56   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 252 240 206 206 207 236 252 177   5   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   7  69  50   0   0   0  78 252 252  22   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  47 252 252 117   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  47 252 252 137   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 149 159  17   0   0  13 118 253 253  33   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 253 252 227 161 161 212 252 252 252  64   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 253 252 252 252 252 253 252 252 101   6   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 169 252 252 252 252 253 235 128   9   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  13 158 252 252 252 243 113   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   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]]
```

Transforming Target variable: one-hot encoding

One-hot encoding involves representing each categorical variable with a binary vector that has one element for each unique label and marking the class label with a 1 and all other elements 0. I created a 10-class label of the target variable as seen in the example below.



	0	1	2	3	4	5	6	7	8	9
label										
5	0	0	0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0

Feature Transformation: Reshape and Normalization

I unrolled each 28 x 28 array to create a (784, 1) vector, to make it a series of columns.

```
x_train:        (60000, 28, 28)    x_train_reshaped shape:  (60000, 784)
x_test:         (10000, 28, 28)    x_test_reshaped shape:  (10000, 784)
```

```
{0, 1, 2, 3, 9, 11, 14, 16, 18, 23, 24, 25, 26, 27, 30, 35, 36, 39, 43, .......
```

I then did min max normalization to scale the data from 0 to 1, as normalizing the data generally speeds up learning and leads to faster convergence.

```
{0.0, 0.011764706, 0.53333336, 0.07058824, 0.49411765, 0.6862745, 0.101960786, ...
```

Review research design and modeling methods

For this analysis, I used the Deep Neural Network (DNN) model to classify the hand-written data . DNN can solve classification problems and its strength is its ability to dynamically create complex prediction functions and emulate human thinking.
DNN architecture design has the most impact on accuracy. I did a 2 x 2 factorial design for number of layers and number of nodes and evaluated 4 models.

I did hyperparameter tuning to find an optimal model for one hidden layer model and for two hidden layers model.

As per the hyperparameter:
For 1 hidden layer model: 448 nodes gave optimum accuracy
For 2 hidden layers model: 320 nodes gave optimum accuracy

Additionally, for each model, I used the regularization technique of batch normalization and the dropout method to reduce any overfitting and improve model performance. I used 10 Epoch, Rmsprop optimizer and Relu activation function in all the models.

I also tried the early stopping regularization technique in all the models, but it reduced the accuracy and therefore I did not use it in the models.

Review results, evaluate models

All models include the following parametes,

Optimizer: RMSprop

Regularization: Batch normalization and dropout

Epoch: 10

Activation function: Relu (hidden layers), Softmax(output)

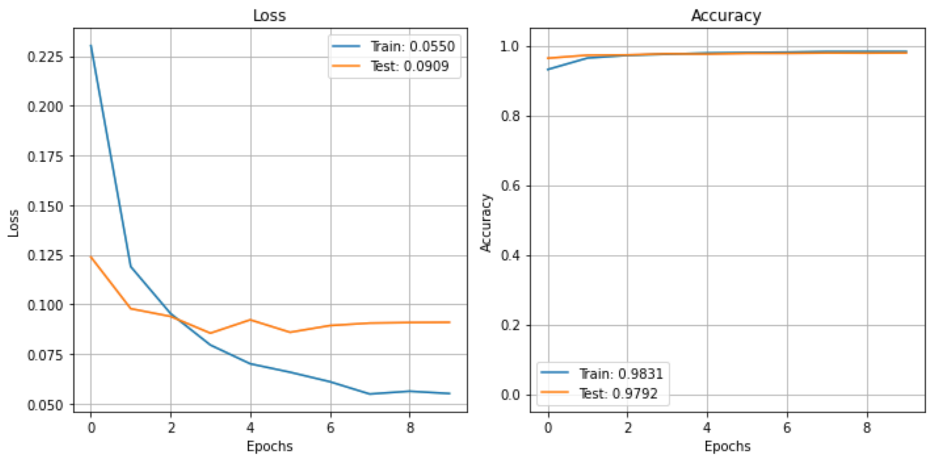
Model 1: 1 layer model with 448 Nodes

The model input layer has 784 nodes that are connected to one hidden layer of 448 nodes which utilizes the Relu activation function and applies batch normalization and dropout (0.2) which is connected to 10 output nodes which applies the SoftMax activation function.

Time to process: 2min 28s

Train: `loss: 0.0285 - accuracy: 0.9918`

Test: `loss: 0.0765 - accuracy: 0.9811`



The confusion matrix shows that the model performs well on train and test data. The rows of the confusion matrix represent the actual label starting from 0 to 9 and the columns represent the predicted label. Each column label is from 0-9.

Based on the test data, hand-written 9,7 and 0 are harder to classify and get misclassified as 4 ,2 and 6 respectively. This makes intuitive sense when looking at the actual images.

Train

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[5893,    1,    1,    0,    0,    1,   16,    0,    9,    2],
       [   0, 6705,    7,    2,    7,    0,    3,    7,   10,    1],
       [  10,    6, 5904,    8,    1,    0,    2,    7,   20,    0],
       [   2,    4,   17, 6028,    0,   22,    1,    8,   38,   11],
       [   2,    4,    2,    0, 5807,    2,    5,    2,    2,   16],
       [   2,    2,    4,    6,    3, 5366,   19,    1,   10,    8],
       [   5,    0,    0,    0,    8,    6, 5898,    0,    1,    0],
       [   2,    6,   10,    5,    8,    1,    0, 6215,    1,   17],
       [   9,    4,    1,    4,    2,   10,    7,    4, 5806,    4],
       [   9,    2,    0,    6,   24,   13,    0,   17,    9, 5869]], dtype=int32)>
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      5923
           1       1.00      0.99      1.00      6742
           2       0.99      0.99      0.99      5958
           3       0.99      0.98      0.99      6131
           4       0.99      0.99      0.99      5842
           5       0.99      0.99      0.99      5421
           6       0.99      1.00      0.99      5918
           7       0.99      0.99      0.99      6265
           8       0.98      0.99      0.99      5851
           9       0.99      0.99      0.99      5949

    accuracy                           0.99     60000
   macro avg       0.99      0.99      0.99     60000
weighted avg       0.99      0.99      0.99     60000
```

Test

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 967,    1,    0,    0,    0,    1,    7,    1,    2,    1],
       [   0, 1130,    1,    1,    0,    0,    1,    0,    2,    0],
       [   5,    3, 1011,    1,    1,    0,    1,    6,    4,    0],
       [   2,    0,    4,  987,    1,    5,    0,    3,    5,    3],
       [   1,    0,    3,    1,  959,    0,    4,    0,    1,   13],
       [   1,    0,    0,    3,    1,  875,    7,    1,    3,    1],
       [   5,    2,    0,    1,    2,    5,  942,    0,    1,    0],
       [   2,    3,    7,    2,    2,    0,    1, 1005,    2,    4],
       [   2,    0,    0,    4,    4,    3,    2,    1,  957,    1],
       [   6,    2,    1,    4,   10,    2,    0,    3,    3,  978]], dtype=int32)>
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       980
           1       0.99      1.00      0.99      1135
           2       0.98      0.98      0.98      1032
           3       0.98      0.98      0.98      1010
           4       0.98      0.98      0.98       982
           5       0.98      0.98      0.98       892
           6       0.98      0.98      0.98       958
           7       0.99      0.98      0.98      1028
           8       0.98      0.98      0.98       974
           9       0.98      0.97      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

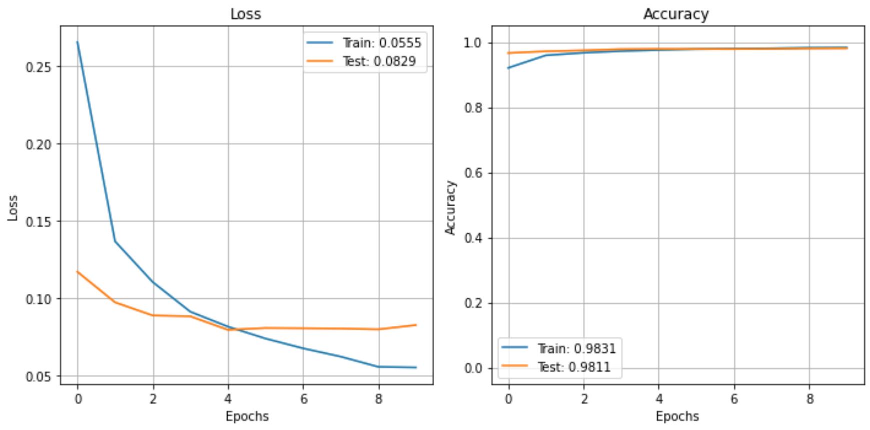
Model 2: 2 layer model with 448 Nodes each

The model input layer has 784 nodes that are connected to two hidden layers of 448 nodes, which utilize the Relu activation function and apply batch normalization and dropout (0.2) to both layers. This is then connected to 10 output nodes which apply the SoftMax activation function.

Time to process: 3min 44s

Train:
```
loss: 0.0288 - accuracy: 0.9925
```

Test:
```
loss: 0.0679 - accuracy: 0.9823
```



The confusion matrix shows that the model performs well on train and test data. The rows of the confusion matrix represent the actual label starting from 0 to 9 and the columns represent the predicted label. Each column label is from 0-9.

Based on the test data, the hand-written 9 ,2 and 5 are harder to classify and get misclassified as 4, 7 and 3 respectively. This makes intuitive sense based on the images.

Train

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[5903,    0,    5,    0,    1,    4,    5,    2,    2,    1],
       [   0, 6716,    4,    5,    3,    0,    3,    6,    2,    3],
       [   3,    3, 5921,    3,    1,    1,    2,   20,    4,    0],
       [   4,    1,   12, 6081,    0,    8,    0,    7,   14,    4],
       [   1,    4,    1,    1, 5791,    0,    7,    7,    2,   28],
       [   6,    1,    3,   30,    2, 5346,   14,    1,   13,    5],
       [  12,    1,    2,    0,    1,    7, 5891,    0,    4,    0],
       [   1,   10,    7,    2,    3,    4,    0, 6234,    0,    4],
       [   6,    4,    8,    7,    1,    8,    5,    4, 5805,    3],
       [   8,    2,    1,    7,   14,    6,    0,   41,    8, 5862]], dtype=int32)>
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      5923
           1       1.00      1.00      1.00      6742
           2       0.99      0.99      0.99      5958
           3       0.99      0.99      0.99      6131
           4       1.00      0.99      0.99      5842
           5       0.99      0.99      0.99      5421
           6       0.99      1.00      0.99      5918
           7       0.99      1.00      0.99      6265
           8       0.99      0.99      0.99      5851
           9       0.99      0.99      0.99      5949

    accuracy                           0.99     60000
   macro avg       0.99      0.99      0.99     60000
weighted avg       0.99      0.99      0.99     60000
```

Test

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 973,    1,    0,    1,    1,    1,    1,    1,    1,    0],
       [   0, 1127,    1,    1,    0,    0,    2,    2,    2,    0],
       [   3,    2, 1016,    1,    2,    0,    0,    7,    1,    0],
       [   1,    0,    0,  997,    0,    3,    0,    5,    2,    2],
       [   0,    1,    4,    0,  963,    0,    3,    1,    2,    8],
       [   3,    0,    0,   11,    0,  870,    3,    1,    1,    3],
       [   3,    2,    1,    1,    1,    2,  947,    0,    1,    0],
       [   1,    3,    8,    4,    0,    1,    0, 1009,    1,    1],
       [   3,    0,    3,    4,    4,    3,    0,    7,  947,    3],
       [   2,    2,    0,    7,    6,    3,    2,   11,    2,  974]], dtype=int32)>
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.98      0.98      1032
           3       0.97      0.99      0.98      1010
           4       0.99      0.98      0.98       982
           5       0.99      0.98      0.98       892
           6       0.99      0.99      0.99       958
           7       0.97      0.98      0.97      1028
           8       0.99      0.97      0.98       974
           9       0.98      0.97      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

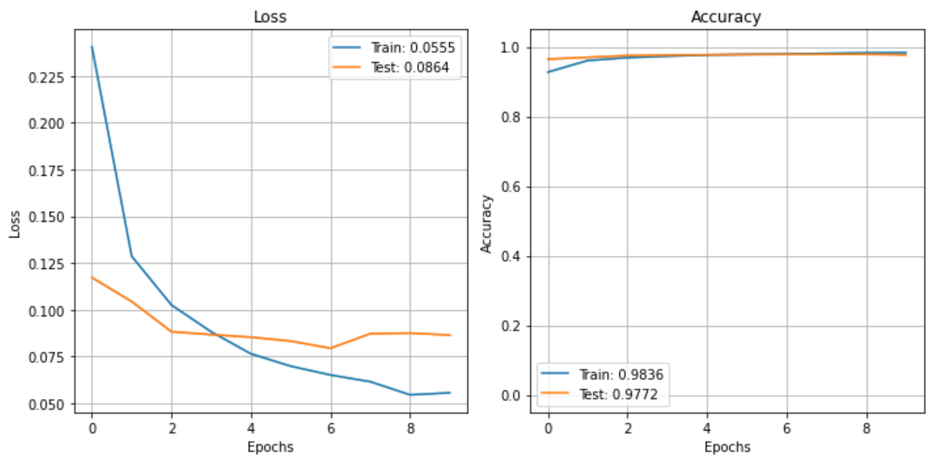
Model 3: 1 layer model with 320 Nodes

The model input layer has 784 nodes that are connected to one hidden layer of 320 nodes, which utilize the Relu activation function and apply batch normalization and dropout (0.2). This is connected to 10 output nodes which apply the SoftMax activation function.

Time to process: 1min 21s

Train : `loss: 0.0318 - accuracy: 0.9907`

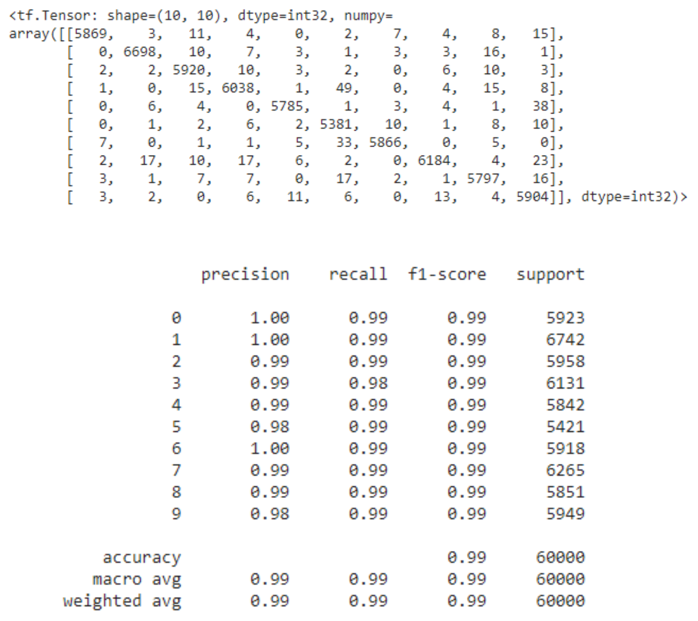
Test: `loss: 0.0766 - accuracy: 0.9797`



The confusion matrix show that the model performs well on train and test data. The rows of the confusion matrix represent the actual label starting from 0 to 9 and the columns represent the predicted label. Each column label is from 0-9.

Based on the test data, the hand-written 9, 2 and 1 are harder to classify and get misclassified as 4 ,7 and 7 respectively. This makes intuitive sense based on the images.

Train



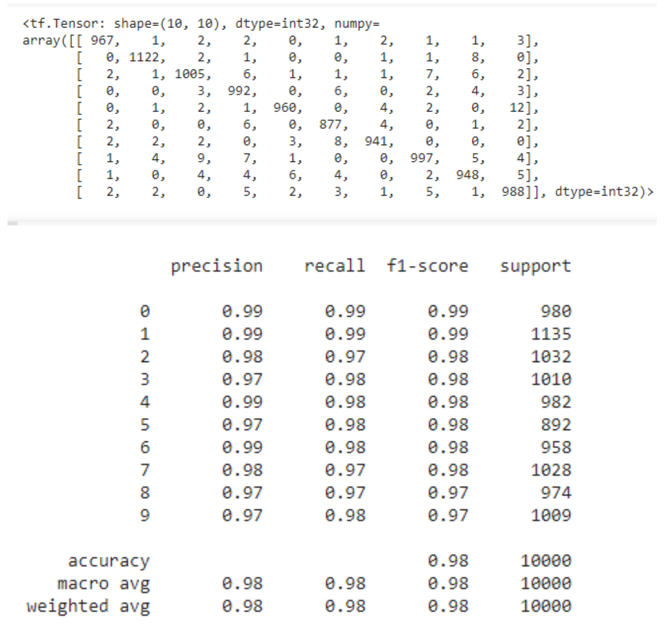
```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[5869,    3,   11,    4,    0,    2,    7,    4,    8,   15],
       [   0, 6698,   10,    7,    3,    1,    3,    3,   16,    1],
       [   2,    2, 5920,   10,    3,    2,    0,    6,   10,    3],
       [   1,    0,   15, 6038,    1,   49,    0,    4,   15,    8],
       [   0,    6,    4,    0, 5785,    1,    3,    4,    1,   38],
       [   0,    1,    2,    6,    2, 5381,   10,    1,    8,   10],
       [   7,    0,    1,    1,    5,   33, 5866,    0,    5,    0],
       [   2,   17,   10,   17,    6,    2,    0, 6184,    4,   23],
       [   3,    1,    7,    7,    0,   17,    2,    1, 5797,   16],
       [   3,    2,    0,    6,   11,    6,    0,   13,    4, 5904]], dtype=int32)>
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      5923
           1       1.00      0.99      0.99      6742
           2       0.99      0.99      0.99      5958
           3       0.99      0.98      0.99      6131
           4       0.99      0.99      0.99      5842
           5       0.98      0.99      0.99      5421
           6       1.00      0.99      0.99      5918
           7       0.99      0.99      0.99      6265
           8       0.99      0.99      0.99      5851
           9       0.98      0.99      0.99      5949

    accuracy                           0.99     60000
   macro avg       0.99      0.99      0.99     60000
weighted avg       0.99      0.99      0.99     60000
```

Test



```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 967,    1,    2,    2,    0,    1,    2,    1,    1,    3],
       [   0, 1122,    2,    1,    0,    0,    1,    1,    8,    0],
       [   2,    1, 1005,    6,    1,    1,    1,    7,    6,    2],
       [   0,    0,    3,  992,    0,    6,    0,    2,    4,    3],
       [   0,    1,    2,    1,  960,    0,    4,    2,    0,   12],
       [   2,    0,    0,    6,    0,  877,    4,    0,    1,    2],
       [   2,    2,    2,    0,    3,    8,  941,    0,    0,    0],
       [   1,    4,    9,    7,    1,    0,    0,  997,    5,    4],
       [   1,    0,    4,    4,    6,    4,    0,    2,  948,    5],
       [   2,    2,    0,    5,    2,    3,    1,    5,    1,  988]], dtype=int32)>
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.97      0.98      1032
           3       0.97      0.98      0.98      1010
           4       0.99      0.98      0.98       982
           5       0.97      0.98      0.98       892
           6       0.99      0.98      0.98       958
           7       0.98      0.97      0.98      1028
           8       0.97      0.97      0.97       974
           9       0.97      0.98      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

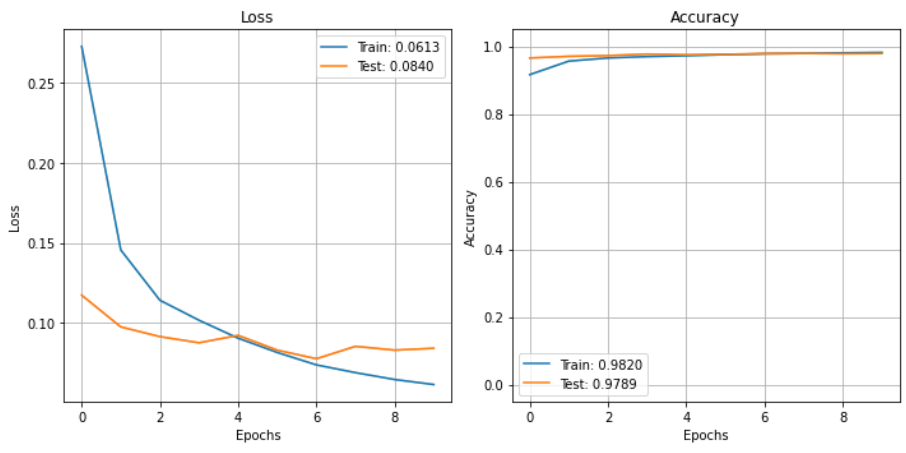
Model 4: 2 layer model with 320 Nodes each

The model input layer has 784 nodes that are connected to two hidden layers of 320 nodes, which utilize the Relu activation function and apply batch normalization and dropout (0.2) to both of the layers. This is then connected to 10 output nodes which apply the SoftMax activation function.

Time to process: 1min 56s

Train: `loss: 0.0307 - accuracy: 0.9915`

Test: `loss: 0.0685 - accuracy: 0.9824`



The confusion matrix shows that the model performs well on train and test data. The rows of the confusion matrix represent the actual label starting from 0 to 9 and the columns represent the predicted label. Each column label is from 0-9.

Based on the test data, hand-written 9, 5 and 2 are harder for the model to classify and get misclassified as 4, 3 and 7 respectively. This makes intuitive sense based on the images.

Train

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[5900,    0,    4,    1,    0,    1,    6,    2,    6,    3],
       [   0, 6702,    5,    2,    2,    0,    6,    5,   17,    3],
       [   6,    9, 5917,    2,    2,    0,    1,    7,   14,    0],
       [   2,    0,   28, 6043,    0,    7,    0,    6,   32,   13],
       [   1,    4,    3,    0, 5763,    1,    4,    1,    2,   63],
       [   1,    0,    7,   31,    3, 5337,   14,    2,   17,    9],
       [   3,    1,    2,    0,    4,    7, 5891,    0,   10,    0],
       [   1,   10,   11,    4,    7,    0,    0, 6214,    2,   16],
       [   3,    2,    4,    4,    1,    4,    6,    4, 5821,    2],
       [   5,    1,    1,    6,   10,    4,    1,   12,    9, 5900]], dtype=int32)>
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5923
           1       1.00      0.99      1.00      6742
           2       0.99      0.99      0.99      5958
           3       0.99      0.99      0.99      6131
           4       0.99      0.99      0.99      5842
           5       1.00      0.98      0.99      5421
           6       0.99      1.00      0.99      5918
           7       0.99      0.99      0.99      6265
           8       0.98      0.99      0.99      5851
           9       0.98      0.99      0.99      5949

    accuracy                           0.99     60000
   macro avg       0.99      0.99      0.99     60000
weighted avg       0.99      0.99      0.99     60000
```

Test

```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 972,    0,    1,    0,    1,    0,    2,    1,    3,    0],
       [   0, 1126,    3,    1,    0,    0,    3,    0,    2,    0],
       [   0,    0, 1022,    0,    1,    0,    0,    6,    2,    1],
       [   0,    0,    2,  994,    0,    3,    0,    3,    4,    4],
       [   3,    0,    0,    0,  948,    0,    5,    3,    2,   21],
       [   2,    0,    0,    8,    1,  866,    3,    1,    4,    7],
       [   3,    1,    2,    0,    1,    5,  944,    0,    2,    0],
       [   1,    5,    7,    2,    1,    0,    0, 1004,    1,    7],
       [   0,    0,    2,    3,    3,    2,    0,    2,  960,    2],
       [   2,    2,    1,    5,    3,    0,    1,    5,    2,  988]], dtype=int32)>
```

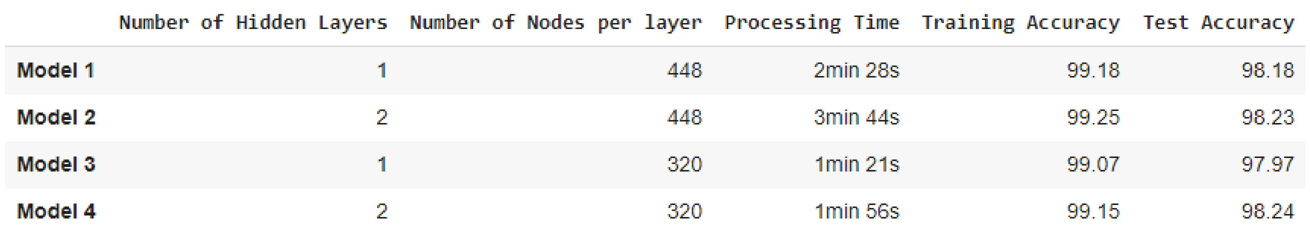
```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.98      0.99      0.99      1032
           3       0.98      0.98      0.98      1010
           4       0.99      0.97      0.98       982
           5       0.99      0.97      0.98       892
           6       0.99      0.99      0.99       958
           7       0.98      0.98      0.98      1028
           8       0.98      0.99      0.98       974
           9       0.96      0.98      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

Summary of the 4 models' data

All of the models perform well and are very similar in performance, but model 4 has the best test accuracy. Based on the loss and accuracy plot and difference between test and train accuracy, model 4 does not appear to overfit and that suggests that it is not memorizing the training data.



	Number of Hidden Layers	Number of Nodes per layer	Processing Time	Training Accuracy	Test Accuracy
Model 1	1	448	2min 28s	99.18	98.18
Model 2	2	448	3min 44s	99.25	98.23
Model 3	1	320	1min 21s	99.07	97.97
Model 4	2	320	1min 56s	99.15	98.24

Implementation and programming

This assignment was done using google colab utilizing Tensoflow Keras packages, NumPy, scikit-learn and pandas.

I loaded the MINST target Y train and test arrays using the Keras dataset package.

I used NumPy and pandas for exploratory data analysis.

I then created a one-hot encoded Y train and test NumPy array.

Then I unrolled the 28 x 28 NumPy array into a flat 784 array and normalized it by dividing it by the max value that is 255. That scales the values from 0 to 1.

Using the model method from Keras, I created all 4 model objects, which included the model architecture. In that architecture, I used batch normalization (default) and dropout (0.2) regularization method on the hidden layer/layers from Keras package.

Then I used the compile method to select the algorithm and optimizer and metric for the model.

I then fitted the model using 10 epochs and validation test of 0.2. The model fit was saved in object (history).

I used the model evaluate method to get the accuracy and loss on training and test data.

I used Keras plot history function on the history of the model to plot the loss and accuracy plots for each epoch on the train and test/validated data.

I used confusion matrix function from scikit-learn and created a confusion matrix for test and train data for all 4 models.

I used the scikit-learn classification report to get the accuracy, recall and precision statistics for test and train data for all 4 models

Exposition, problem description, and management recommendations

Based on my analysis above, I would recommend Model 4, that has 2 hidden layer with 320 nodes each. This model performed better based on test accuracy and has the smallest difference between test and train accuracy, which along with the loss and accuracy graph confirm that the model does not overfit data. It is also the 2nd most time-efficient after model 3.

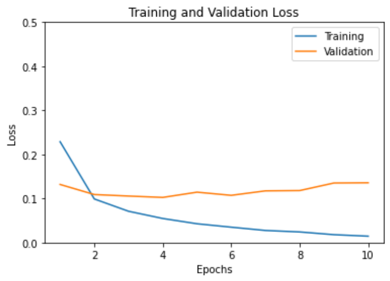
At the end, all models performed really well and the performances are very close to each other. Model 4 produced the highest accuracy and model 3 took the least amount of time. If time to train is priority , then Model 3 with one hidden layers of 320 nodes should be chosen.

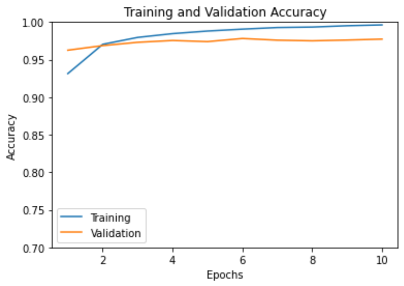
All the models performed at 98% accuracy on the test data. Misclassification occurred because hand-written images, 9 and 4 for example, are very similar and are hard to distinguish. To achieve better accuracy, we will need a more complex model architecture.

Appendix

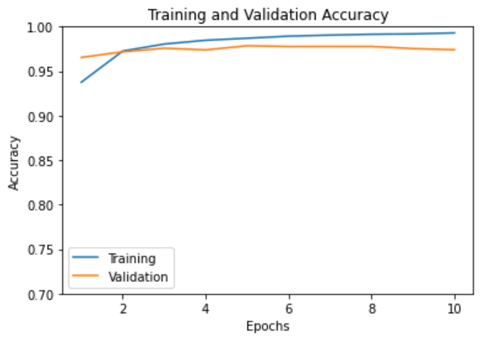
Without regularization

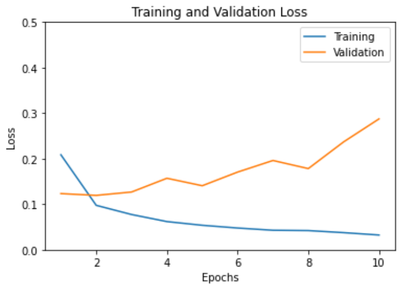
Model 1





Model 2



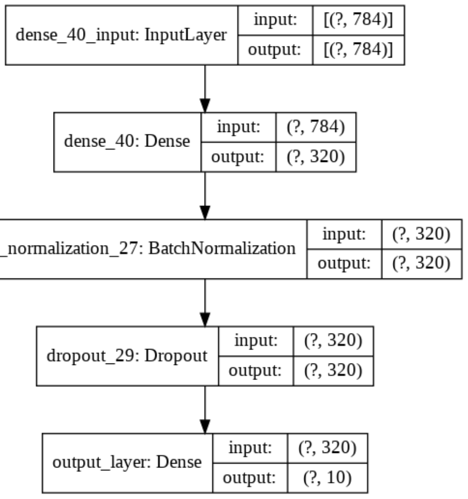


Model 2 architecture

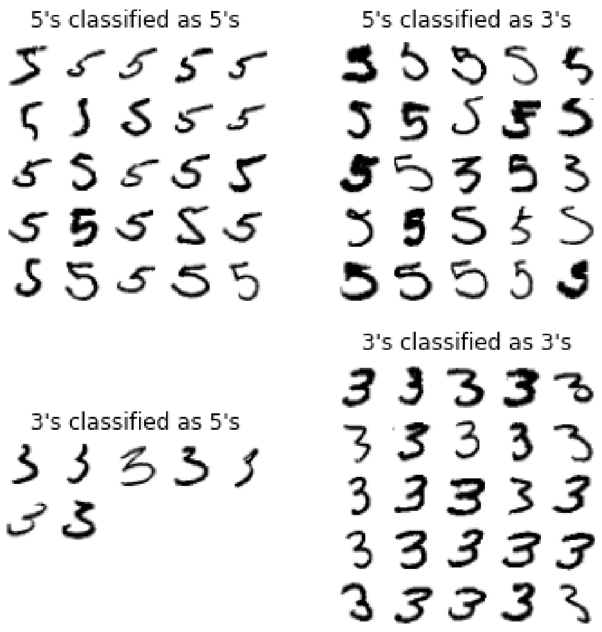
```
[ ]  model.summary()
```

```
Model: "sequential_25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_40 (Dense)             (None, 320)               251200
_________________________________________________________________
batch_normalization_27 (Batc (None, 320)               1280
_________________________________________________________________
dropout_29 (Dropout)         (None, 320)               0
_________________________________________________________________
output_layer (Dense)         (None, 10)                3210
=================================================================
Total params: 255,690
Trainable params: 255,050
Non-trainable params: 640
_________________________________________________________________
```

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



Sample error in model 4


5's classified as 5's
5's classified as 3's
3's classified as 3's
3's classified as 5's