MNIST Deep Learning

INTRODUCTION

In this MNIST Deep Learning Experiment, we will need to build a model that can determine a digit (0-9) from a handwritten image with 784 pixels, and we will develop a deep learning model training with a 70000 image dataset. The purpose of this experiment is to reach the higher accuracy possible through several rounds of tuning in hyperparameters. All works will be done under the TensorFlow framework (Keras) and Google colabotory, where python programming is necessary. In the preprocessing, each value will be scaled by dividing 255, getting a range between 0 and 1. This step will make our data suitable for training. And the data set will also be shuffled in a buffer size of 10000 for limited memory space and spread the data as randomly as possible. These preprocessing steps will make learning more efficient. Upon the preprocessing, we also need to define our input and output layers. The input layer contains 784 nodes, which equals the number of pixels in the picture. The output layer contains 10 nodes, and each node represents the possibility of a number. In order to make this work, we must need softmax activation to produce the possibility for our output layer. After these preparations, we can now start our experiment. The experiment will contain several steps, and each step will look for an optimal hyperparameter that can produce the best accuracy through GridSearch. The hyperparameters that we will work on are hidden layer depth, hidden layer width, activation functions, nodes distribution in the hidden layer, batches size, data split ratio, initializers, optimizer, and the number of epochs. Since there is a large number of possible combinations between these hyperparameters, it's impossible for this experiment to try every single combination for the best result. Instead, I will target a few hyperparameters for each step, and integrate the result with the next step of the experiment.

EXPERIMENT

Hidden Layers Depth, Width, Activation Functions, and Nodes Distribution (Model Complexity)

This is the first step to reaching optimal accuracy. We will be finding the best combination of Depth, Width, Activation Functions, and Nodes Distribution. First, we will try a different combination of width and depth with ReLU, and pick the one with the best accuracy. Then we use the best combination of width and depth in our next test to try a different combination of activation functions. We pick the one with the best accuracy among all combinations and try different combinations of node distribution in each layer. After these steps, we will have optimal combinations with good accuracy and complexity since we don't want too much model complexity or too low accuracy. More model complexity may not help the model do better, and the goal of these steps is to help find the optimal point. In addition, we also want to avoid overfitting the validation set, and a 1% difference will be considered too much.

Grid Search the number of Depth and Width for the Best Accuracy. For testing efficiency, we limited the range of nodes per layer between the output size and to input size. For the number of layers, we will keep adding more until there is no significant difference from the previous one in the accuracy of the test set. (at least 2-4 hidden layers will be tested)

Hidden Layers	Activation Function Layer	Activation Function Layer *	Nodes per layer	Accuracy in validation	Accuracy in test
2	ReLU	ReLU	50	97.42%	96.67%
2	ReLU	ReLU	100	98%	97.39%
2	ReLU	ReLU	200	98.47%	97.63%
2	ReLU	ReLU	400	98.73%	98.14%
2	ReLU	ReLU	500	99.05%	97.91%
3	ReLU	ReLU	50	97.33%	96.75%
3	ReLU	ReLU	100	98.45%	97.63%
3	ReLU	ReLU	200	98.40%	97.83%
3	ReLU	ReLU	400	98.35%	97.53%
4	ReLU	ReLU	50	97.60%	97.01%
4	ReLU	ReLU	100	97.87%	97.41%
4	ReLU	ReLU	200	98.60%	97.74%
4	ReLU	ReLU	400	98.53%	97.55%

According to the test among the combination of a various number of hidden layers and hidden layer size. The hidden layer depth of 2 and the width of 400 is a good combination for our model. 500 seems to be overfitting to our validation set, and 3 and 4 hidden layers don't have much improvement over 2. Therefore, adding more complexity over that may not be a good idea. Then we will be using the combination of a hidden layer depth of 2 and a width of 400 for our next stop, finding the better activation function for each layer.

Grid Search the combination of different activation functions in hidden layers. The 3 commonly used hidden layers activation functions are ReLU, Sigmoid, and TanH. We will be using these 3 functions to test out the best accuracy value in the test set.

Hidden Layers	Activation Function Layer 1	Activation Function Layer 2	Nodes per layer	Accuracy in validation	Accuracy in test
2	ReLU	ReLU	400	98.73%	98.14%
2	Sigmoid	Sigmoid	400	97.30%	96.93%
2	TanH	TanH	400	98.55%	97.54%
2	ReLU	Sigmoid	400	98.42%	97.78%
2	Sigmoid	ReLU	400	97.52%	96.81%
2	TanH	Sigmoid	400	98.15%	97.44%
2	Sigmoid	TanH	400	97.55%	97.30%
2	ReLU	TanH	400	98.95%	97.72%
2	Tanh	ReLU	400	98.33%	97.59%

According to the test above, we found that sigmoid is not very efficient to this model, which results in low accuracy. Relu has better results in accuracy than TanH. The combination with the highest accuracy to the test set is ReLU as Activation #1 and ReLU as Activation #2. And we will use this for our later test to try different distributions of nodes for each layer.

Grid Search different distribution of nodes in hidden layers, while the width is 400 per node. The approach will be +/- 100 for two layers opposite each time, shown in the table below. 7 major combinations will be tested for best accuracy in the test set.

Hidden Layers	Activation Function Layer 1	Activation Function Layer 2	Nodes for Activation Layer 1	Nodes for Activation Layer 2	Accuracy in validation	Accuracy in test
2	ReLU	ReLU	400	400	98.73%	98.14%
2	ReLU	ReLU	300	500	98.68%	97.90%
2	ReLU	ReLU	500	300	98.73%	97.91%
2	ReLU	ReLU	200	600	98.50%	97.88%
2	ReLU	ReLU	600	200	98.78%	98.15%

2	ReLU	ReLU	100	700	98.32%	97.58%
2	ReLU	ReLU	700	100	98.72%	98.02%

First ReLU with 600 nodes and Second ReLU with 200 nodes has better result in accuracy according to the above test. The distribution where decreasing the number of nodes in hidden layers toward output works better in the model overall. Combining everything together, now we get an optimal combination and complexity level for hidden layer depth, width, activation functions, and node distribution. These numbers will be used for the resting test.

Hidden Layers Depth: 2, Hidden Layers Width: 400
Activation Function Layer 1: ReLU
Activation Function Layer 2: ReLU
Nodes for Activation Layer 1: 600
Nodes for Activation Layer 2: 200

Batch Size

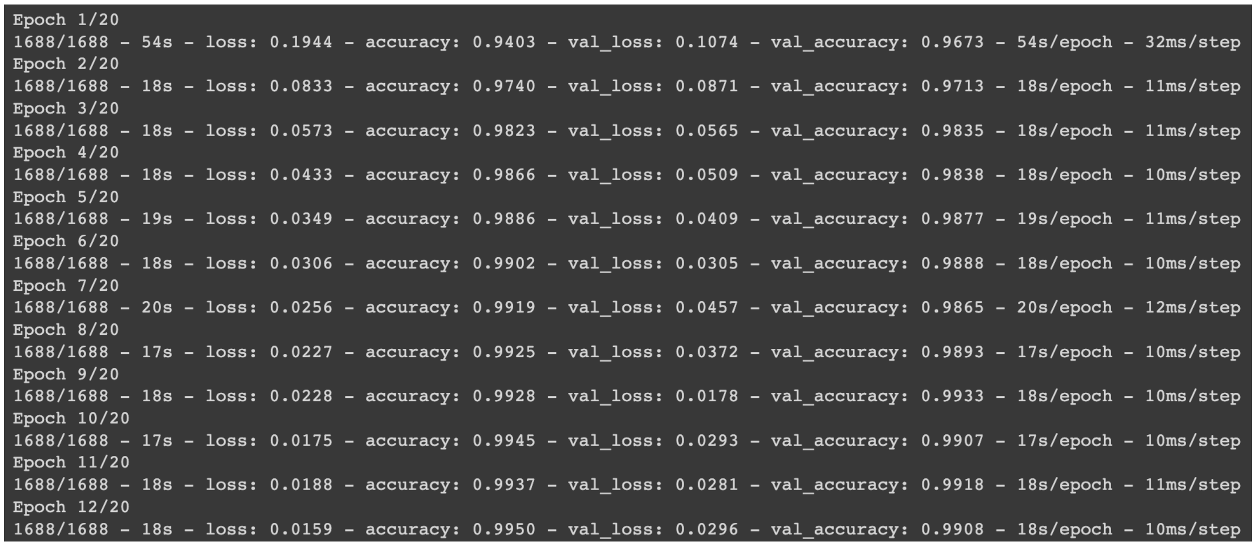
In deep learning, large batch size will learn and converge slowly but more accurate, and small batch size will learn and converge quickly but less accurate. Because there is a lower number of iterations (but more samples processed each iteration) per epoch for large batch size, and small batch size is the opposite. As we know that SGD and GD are two extremes in terms of speed and accuracy. SGD updates the model after every single sample, which can speed up a lot in the number training but lower accuracy. GD updates the model after all samples, which can get higher accuracy but a lower number of updates or training each epoch. In conclusion, SGD will lose the speedup benefit due to its low accuracy rate, and GD will also lose the accuracy due to its slow speed in a limited epoch number.

In this experiment, we will find the best accuracy for 5 different batch sizes (32, 64, 128, 256, 512) from different epochs. These numbers are chosen in the power of 2 for better performance and efficiency. We will choose the smaller possible batch sizes while maintaining a good accuracy result.

Batch Size	Best Accuracy in Test Set
32	97.88%
64	98.08%
128	97.74%
256	98.09%
512	98.14%

Grid Search for each batch size with different Accuracy and Epoch. Finding the best possible Accuracy for each batch size without overfitting. In order to avoid overfitting, the model will perform early stopping, where we stop when the validation loss starts increasing or the training loss becomes very small. (Below is the process of getting the above results)

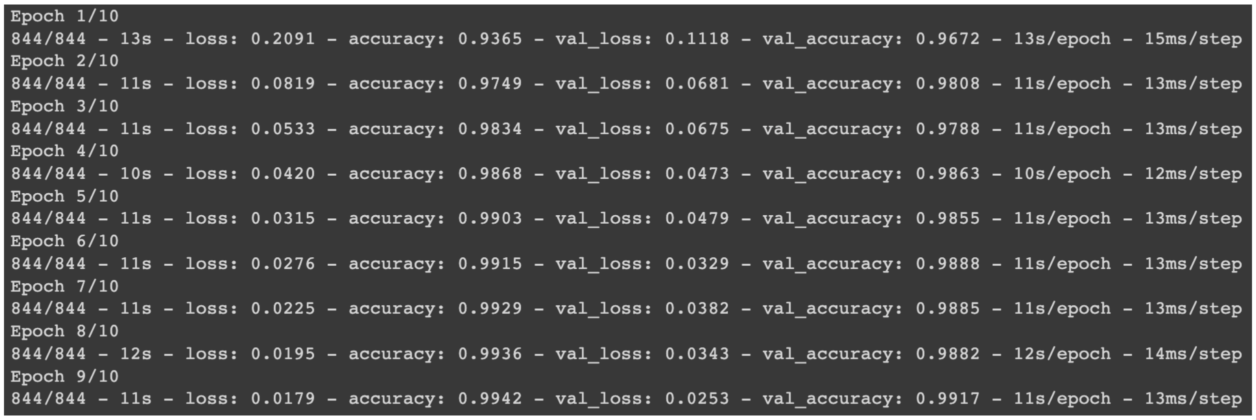
32 Batch Size



```
Epoch 1/20
1688/1688 - 54s - loss: 0.1944 - accuracy: 0.9403 - val_loss: 0.1074 - val_accuracy: 0.9673 - 54s/epoch - 32ms/step
Epoch 2/20
1688/1688 - 18s - loss: 0.0833 - accuracy: 0.9740 - val_loss: 0.0871 - val_accuracy: 0.9713 - 18s/epoch - 11ms/step
Epoch 3/20
1688/1688 - 18s - loss: 0.0573 - accuracy: 0.9823 - val_loss: 0.0565 - val_accuracy: 0.9835 - 18s/epoch - 11ms/step
Epoch 4/20
1688/1688 - 18s - loss: 0.0433 - accuracy: 0.9866 - val_loss: 0.0509 - val_accuracy: 0.9838 - 18s/epoch - 10ms/step
Epoch 5/20
1688/1688 - 19s - loss: 0.0349 - accuracy: 0.9886 - val_loss: 0.0409 - val_accuracy: 0.9877 - 19s/epoch - 11ms/step
Epoch 6/20
1688/1688 - 18s - loss: 0.0306 - accuracy: 0.9902 - val_loss: 0.0305 - val_accuracy: 0.9888 - 18s/epoch - 10ms/step
Epoch 7/20
1688/1688 - 20s - loss: 0.0256 - accuracy: 0.9919 - val_loss: 0.0457 - val_accuracy: 0.9865 - 20s/epoch - 12ms/step
Epoch 8/20
1688/1688 - 17s - loss: 0.0227 - accuracy: 0.9925 - val_loss: 0.0372 - val_accuracy: 0.9893 - 17s/epoch - 10ms/step
Epoch 9/20
1688/1688 - 18s - loss: 0.0228 - accuracy: 0.9928 - val_loss: 0.0178 - val_accuracy: 0.9933 - 18s/epoch - 10ms/step
Epoch 10/20
1688/1688 - 17s - loss: 0.0175 - accuracy: 0.9945 - val_loss: 0.0293 - val_accuracy: 0.9907 - 17s/epoch - 10ms/step
Epoch 11/20
1688/1688 - 18s - loss: 0.0188 - accuracy: 0.9937 - val_loss: 0.0281 - val_accuracy: 0.9918 - 18s/epoch - 11ms/step
Epoch 12/20
1688/1688 - 18s - loss: 0.0159 - accuracy: 0.9950 - val_loss: 0.0296 - val_accuracy: 0.9908 - 18s/epoch - 10ms/step
```

This is a portion of the model output with a 32-batch size. The model starts overfitting from around epoch 7, where validation loss starts increasing. And we will perform early stopping in epoch 7 for batch size 32.

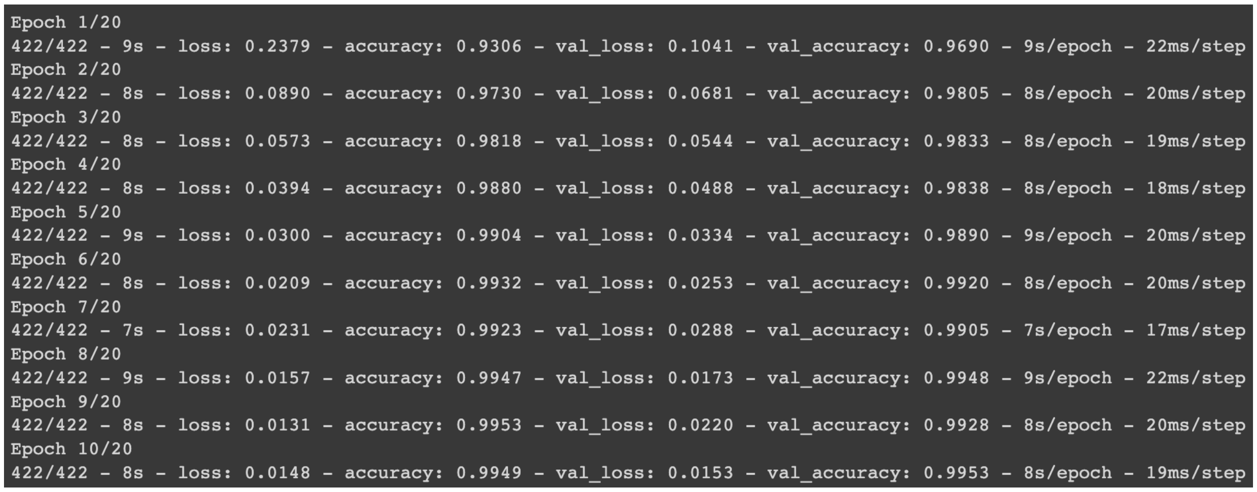
64 Batch Size



```
Epoch 1/10
844/844 - 13s - loss: 0.2091 - accuracy: 0.9365 - val_loss: 0.1118 - val_accuracy: 0.9672 - 13s/epoch - 15ms/step
Epoch 2/10
844/844 - 11s - loss: 0.0819 - accuracy: 0.9749 - val_loss: 0.0681 - val_accuracy: 0.9808 - 11s/epoch - 13ms/step
Epoch 3/10
844/844 - 11s - loss: 0.0533 - accuracy: 0.9834 - val_loss: 0.0675 - val_accuracy: 0.9788 - 11s/epoch - 13ms/step
Epoch 4/10
844/844 - 10s - loss: 0.0420 - accuracy: 0.9868 - val_loss: 0.0473 - val_accuracy: 0.9863 - 10s/epoch - 12ms/step
Epoch 5/10
844/844 - 11s - loss: 0.0315 - accuracy: 0.9903 - val_loss: 0.0479 - val_accuracy: 0.9855 - 11s/epoch - 13ms/step
Epoch 6/10
844/844 - 11s - loss: 0.0276 - accuracy: 0.9915 - val_loss: 0.0329 - val_accuracy: 0.9888 - 11s/epoch - 13ms/step
Epoch 7/10
844/844 - 11s - loss: 0.0225 - accuracy: 0.9929 - val_loss: 0.0382 - val_accuracy: 0.9885 - 11s/epoch - 13ms/step
Epoch 8/10
844/844 - 12s - loss: 0.0195 - accuracy: 0.9936 - val_loss: 0.0343 - val_accuracy: 0.9882 - 12s/epoch - 14ms/step
Epoch 9/10
844/844 - 11s - loss: 0.0179 - accuracy: 0.9942 - val_loss: 0.0253 - val_accuracy: 0.9917 - 11s/epoch - 13ms/step
```

This is a portion of the model output with a 64-batch size. The model starts overfitting from around epoch 7, where validation loss starts increasing. And we will perform early stopping in epoch 7 for batch size 64.

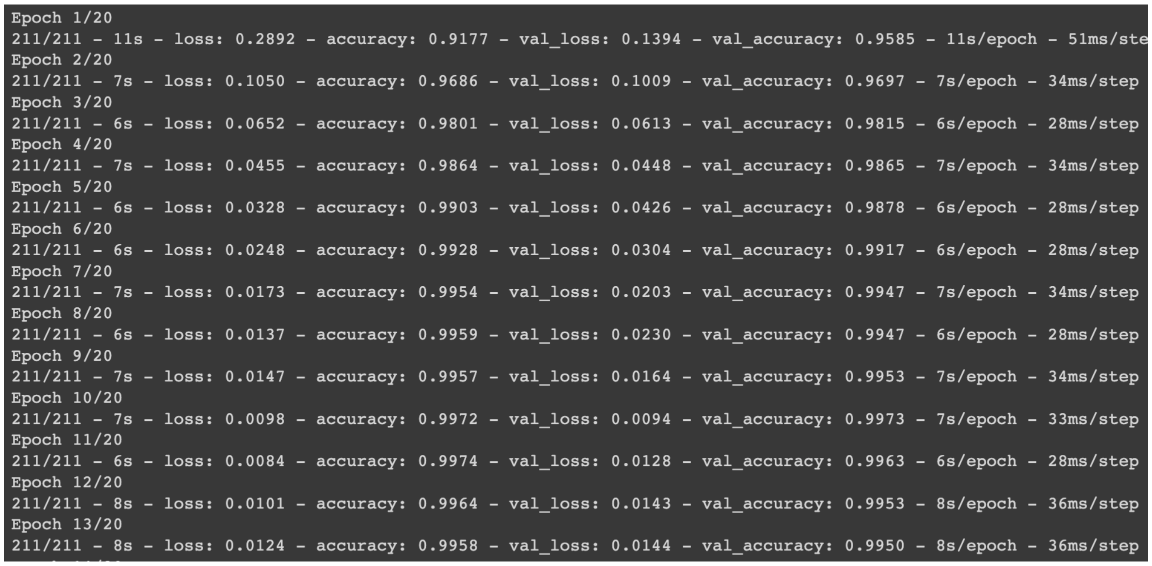
128 Batch Size



```
Epoch 1/20
422/422 - 9s - loss: 0.2379 - accuracy: 0.9306 - val_loss: 0.1041 - val_accuracy: 0.9690 - 9s/epoch - 22ms/step
Epoch 2/20
422/422 - 8s - loss: 0.0890 - accuracy: 0.9730 - val_loss: 0.0681 - val_accuracy: 0.9805 - 8s/epoch - 20ms/step
Epoch 3/20
422/422 - 8s - loss: 0.0573 - accuracy: 0.9818 - val_loss: 0.0544 - val_accuracy: 0.9833 - 8s/epoch - 19ms/step
Epoch 4/20
422/422 - 8s - loss: 0.0394 - accuracy: 0.9880 - val_loss: 0.0488 - val_accuracy: 0.9838 - 8s/epoch - 18ms/step
Epoch 5/20
422/422 - 9s - loss: 0.0300 - accuracy: 0.9904 - val_loss: 0.0334 - val_accuracy: 0.9890 - 9s/epoch - 20ms/step
Epoch 6/20
422/422 - 8s - loss: 0.0209 - accuracy: 0.9932 - val_loss: 0.0253 - val_accuracy: 0.9920 - 8s/epoch - 20ms/step
Epoch 7/20
422/422 - 7s - loss: 0.0231 - accuracy: 0.9923 - val_loss: 0.0288 - val_accuracy: 0.9905 - 7s/epoch - 17ms/step
Epoch 8/20
422/422 - 9s - loss: 0.0157 - accuracy: 0.9947 - val_loss: 0.0173 - val_accuracy: 0.9948 - 9s/epoch - 22ms/step
Epoch 9/20
422/422 - 8s - loss: 0.0131 - accuracy: 0.9953 - val_loss: 0.0220 - val_accuracy: 0.9928 - 8s/epoch - 20ms/step
Epoch 10/20
422/422 - 8s - loss: 0.0148 - accuracy: 0.9949 - val_loss: 0.0153 - val_accuracy: 0.9953 - 8s/epoch - 19ms/step
```

This is a portion of the model output with a 128-batch size. The model starts overfitting from around epoch 7, where validation loss starts increasing. And we will perform early stopping in epoch 7 for batch size 128.

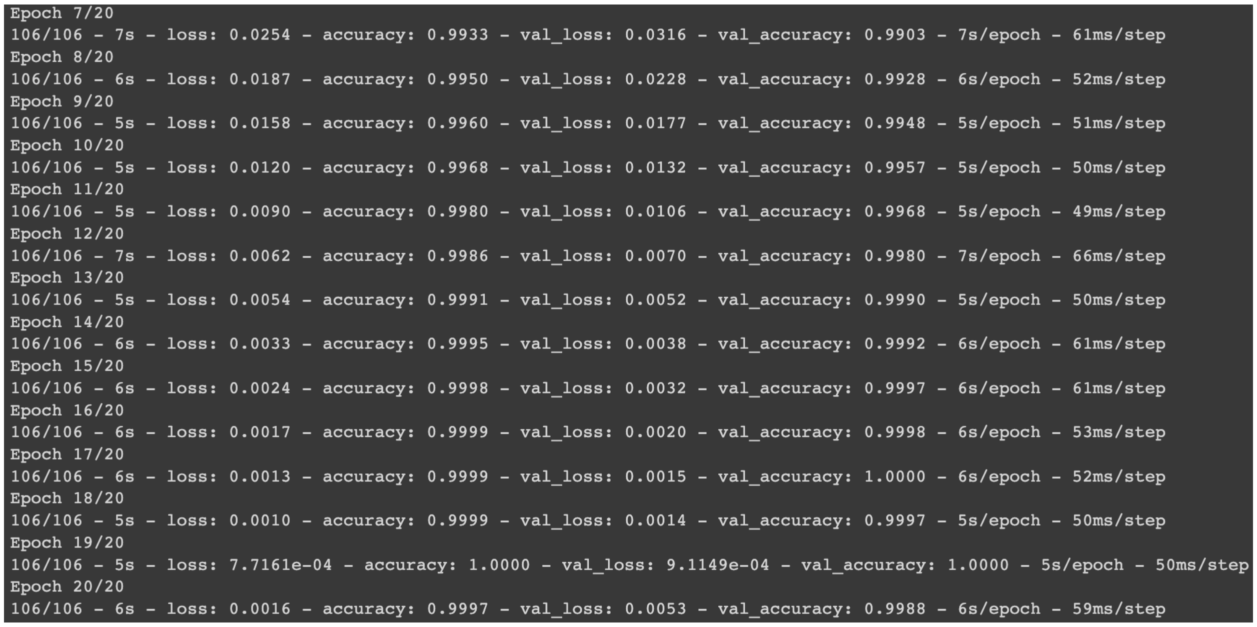
256 Batch Size



```
Epoch 1/20
211/211 - 11s - loss: 0.2892 - accuracy: 0.9177 - val_loss: 0.1394 - val_accuracy: 0.9585 - 11s/epoch - 51ms/ste
Epoch 2/20
211/211 - 7s - loss: 0.1050 - accuracy: 0.9686 - val_loss: 0.1009 - val_accuracy: 0.9697 - 7s/epoch - 34ms/step
Epoch 3/20
211/211 - 6s - loss: 0.0652 - accuracy: 0.9801 - val_loss: 0.0613 - val_accuracy: 0.9815 - 6s/epoch - 28ms/step
Epoch 4/20
211/211 - 7s - loss: 0.0455 - accuracy: 0.9864 - val_loss: 0.0448 - val_accuracy: 0.9865 - 7s/epoch - 34ms/step
Epoch 5/20
211/211 - 6s - loss: 0.0328 - accuracy: 0.9903 - val_loss: 0.0426 - val_accuracy: 0.9878 - 6s/epoch - 28ms/step
Epoch 6/20
211/211 - 6s - loss: 0.0248 - accuracy: 0.9928 - val_loss: 0.0304 - val_accuracy: 0.9917 - 6s/epoch - 28ms/step
Epoch 7/20
211/211 - 7s - loss: 0.0173 - accuracy: 0.9954 - val_loss: 0.0203 - val_accuracy: 0.9947 - 7s/epoch - 34ms/step
Epoch 8/20
211/211 - 6s - loss: 0.0137 - accuracy: 0.9959 - val_loss: 0.0230 - val_accuracy: 0.9947 - 6s/epoch - 28ms/step
Epoch 9/20
211/211 - 7s - loss: 0.0147 - accuracy: 0.9957 - val_loss: 0.0164 - val_accuracy: 0.9953 - 7s/epoch - 34ms/step
Epoch 10/20
211/211 - 7s - loss: 0.0098 - accuracy: 0.9972 - val_loss: 0.0094 - val_accuracy: 0.9973 - 7s/epoch - 33ms/step
Epoch 11/20
211/211 - 6s - loss: 0.0084 - accuracy: 0.9974 - val_loss: 0.0128 - val_accuracy: 0.9963 - 6s/epoch - 28ms/step
Epoch 12/20
211/211 - 8s - loss: 0.0101 - accuracy: 0.9964 - val_loss: 0.0143 - val_accuracy: 0.9953 - 8s/epoch - 36ms/step
Epoch 13/20
211/211 - 8s - loss: 0.0124 - accuracy: 0.9958 - val_loss: 0.0144 - val_accuracy: 0.9950 - 8s/epoch - 36ms/step
```

This is a portion of the model output with a 256-batch size. The model starts overfitting from around epoch 11, where validation loss starts increasing. And we will perform early stopping in epoch 11 for batch size 256.

512 Batch Size



```
Epoch 7/20
106/106 - 7s - loss: 0.0254 - accuracy: 0.9933 - val_loss: 0.0316 - val_accuracy: 0.9903 - 7s/epoch - 61ms/step
Epoch 8/20
106/106 - 6s - loss: 0.0187 - accuracy: 0.9950 - val_loss: 0.0228 - val_accuracy: 0.9928 - 6s/epoch - 52ms/step
Epoch 9/20
106/106 - 5s - loss: 0.0158 - accuracy: 0.9960 - val_loss: 0.0177 - val_accuracy: 0.9948 - 5s/epoch - 51ms/step
Epoch 10/20
106/106 - 5s - loss: 0.0120 - accuracy: 0.9968 - val_loss: 0.0132 - val_accuracy: 0.9957 - 5s/epoch - 50ms/step
Epoch 11/20
106/106 - 5s - loss: 0.0090 - accuracy: 0.9980 - val_loss: 0.0106 - val_accuracy: 0.9968 - 5s/epoch - 49ms/step
Epoch 12/20
106/106 - 7s - loss: 0.0062 - accuracy: 0.9986 - val_loss: 0.0070 - val_accuracy: 0.9980 - 7s/epoch - 66ms/step
Epoch 13/20
106/106 - 5s - loss: 0.0054 - accuracy: 0.9991 - val_loss: 0.0052 - val_accuracy: 0.9990 - 5s/epoch - 50ms/step
Epoch 14/20
106/106 - 6s - loss: 0.0033 - accuracy: 0.9995 - val_loss: 0.0038 - val_accuracy: 0.9992 - 6s/epoch - 61ms/step
Epoch 15/20
106/106 - 6s - loss: 0.0024 - accuracy: 0.9998 - val_loss: 0.0032 - val_accuracy: 0.9997 - 6s/epoch - 61ms/step
Epoch 16/20
106/106 - 6s - loss: 0.0017 - accuracy: 0.9999 - val_loss: 0.0020 - val_accuracy: 0.9998 - 6s/epoch - 53ms/step
Epoch 17/20
106/106 - 6s - loss: 0.0013 - accuracy: 0.9999 - val_loss: 0.0015 - val_accuracy: 1.0000 - 6s/epoch - 52ms/step
Epoch 18/20
106/106 - 5s - loss: 0.0010 - accuracy: 0.9999 - val_loss: 0.0014 - val_accuracy: 0.9997 - 5s/epoch - 50ms/step
Epoch 19/20
106/106 - 5s - loss: 7.7161e-04 - accuracy: 1.0000 - val_loss: 9.1149e-04 - val_accuracy: 1.0000 - 5s/epoch - 50ms/step
Epoch 20/20
106/106 - 6s - loss: 0.0016 - accuracy: 0.9997 - val_loss: 0.0053 - val_accuracy: 0.9988 - 6s/epoch - 59ms/step
```

This is a portion of the model output with a 512-batch size. The model starts overfitting from around epoch 20, where validation loss starts increasing. And we will perform early stopping in epoch 20 for batch size 512.

Summary:
According to several tests done above, the model has a very similar result across all different batches. However, the situations are all different. I notice that the more batch size we have, the more overfitting we are to our validation set, which is not good. Among all results, the **64 batch size** has better overall performance, with low overfitting and high accuracy. So we choose this batch size for our model. In addition, a lower batch size can make our model learn faster so.

Validation Data Size

MNIST data set: 70000 samples
MNIST train and validation set: 60000 samples
MNIST test set: 10000 samples

Validation Data Split Ratio (0.05 - 0.2)	Epoch (Early Stopping)	Accuracy in validation	Accuracy in test
0.05 (3000 samples)	6	99.17%	98.19%
0.075 (4500 samples)	7	99.11%	97.84%
0.1 (6000 samples)	7	98.85%	98.08%

0.125 (7500 samples)	6	98.59%	97.65%
0.15 (9000 samples)	6	98.38%	97.60%
0.175 (10500 samples)	6	98.53%	97.92%
0.2 (12000 samples)	6	98.72%	97.93%

According to the test above, the split ratio can somehow to impact the model performance a little bit. The more samples in the train set may push the validation accuracy higher, but overfitting seems to be worse toward the validation set. The fewer samples in the train set may have lower accuracy but we have less overfitting toward the validation set. I will choose the number in the middle, which is **0.1 ratio** for 6000 validation samples and 54000 train samples.

Initializer

It's been proved that HeNormal initializer is the best for ReLu, and GlorotUniform is the best for Softmax. We will compare if the model learns better and faster with these two initializers.

ReLu Initializer	Softmax Initializer	Accuracy	Epoch
None	None	98.16%	7
tf.keras.initializers. HeNormal()	tf.keras.initializers. GlorotUniform()	98.24%	8

There is no significant difference between with initializers and without initializers in this experiment. However, adding initializers should be a good idea, and is recommended because we want to avoid starting with a non-ideal range for our activation functions.

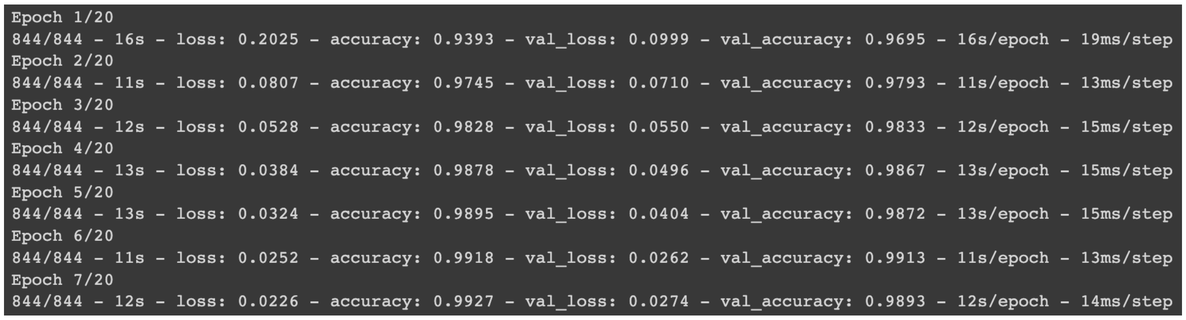
Optimizer

Optimizer	Accuracy
ADAM	98.08%
SGD	94.98%
AdaGrad	91.77%
RMSProp	97.60%

Adam is proven to be the best optimizer so far. According to our test, we can see that ADAM has better accuracy than other optimizers. RMSProp also did very well in the training but was not as efficient as ADAM. Overall, **ADAM** will be our choice for the training optimizer.

Epoch Test

This is the final step in this experiment. We already figure out an optimal value for the above hyperparameters through the test. This step will integrate all of the hyperparameters, and determine the epoch taken. The test will stop when there is overfitting in the validation set.



```
Epoch 1/20
844/844 - 16s - loss: 0.2025 - accuracy: 0.9393 - val_loss: 0.0999 - val_accuracy: 0.9695 - 16s/epoch - 19ms/step
Epoch 2/20
844/844 - 11s - loss: 0.0807 - accuracy: 0.9745 - val_loss: 0.0710 - val_accuracy: 0.9793 - 11s/epoch - 13ms/step
Epoch 3/20
844/844 - 12s - loss: 0.0528 - accuracy: 0.9828 - val_loss: 0.0550 - val_accuracy: 0.9833 - 12s/epoch - 15ms/step
Epoch 4/20
844/844 - 13s - loss: 0.0384 - accuracy: 0.9878 - val_loss: 0.0496 - val_accuracy: 0.9867 - 13s/epoch - 15ms/step
Epoch 5/20
844/844 - 13s - loss: 0.0324 - accuracy: 0.9895 - val_loss: 0.0404 - val_accuracy: 0.9872 - 13s/epoch - 15ms/step
Epoch 6/20
844/844 - 11s - loss: 0.0252 - accuracy: 0.9918 - val_loss: 0.0262 - val_accuracy: 0.9913 - 11s/epoch - 13ms/step
Epoch 7/20
844/844 - 12s - loss: 0.0226 - accuracy: 0.9927 - val_loss: 0.0274 - val_accuracy: 0.9893 - 12s/epoch - 14ms/step
```

Final Accuracy: 98.14%

CONCLUSION

In this experiment, we reached a final accuracy of 98.14% in our test set. We have 2 hidden layers and 400 nodes per layer for our model. This combination seems to be reasonable since we don't really need a lot of complexity on the MNIST problem and 400 nodes per layer is within the range of input size and output size. We also compare several combinations of activation functions and node distribution. ReLu works the best with better accuracy, and [600, 200] distribution for two layers has better performance over others.

We also try different batches sizes and train with each batches size until overfitting. We found out that the more batch size we gain, the more overfitting toward the validation set, where the test set seems to reach its limit of around 98% accuracy. If it's the case, we may prefer a smaller batch size, where the model can learn faster. 64 batch sizes seem to be the best so far.

For the data split between validation and train set, we found a good spot that gives us good accuracy with less overfitting. More samples in the train set will eventually make the data learn more, but it may become harder for us to detect overfitting while the validation set is small. Fewer samples in the train set will make the data learn less, but the overfitting may be easily detected. I prefer somewhere in the middle, where 0.1 is a good spot for our model.

In addition, setting up initializers for our model is recommended since it can avoid us from getting a non-ideal range for our activation function in the beginning. We also compared several optimizers in our model, and ADAM is undoubtedly the best one, which is also proven by many. ADAM performs both momentum and dynamic learning rate schedule in our training, which helps the model learn more smoothly. The momentum will make the model easier to escape the local minimum reaching a lower loss value and the dynamic learning rate schedule can greatly boost the speed of learning while maintaining good accuracy.

Finally, we integrate everything and perform a final test with 7 epochs under early stopping (avoiding overfitting). However, the result of this experiment is definitely not the most ideal or optimal, as we miss out on a lot of possible combinations. Maybe something like GridSearchCV or RandomSearchCV can help do this better. We can do this as an additional experiment for the future.