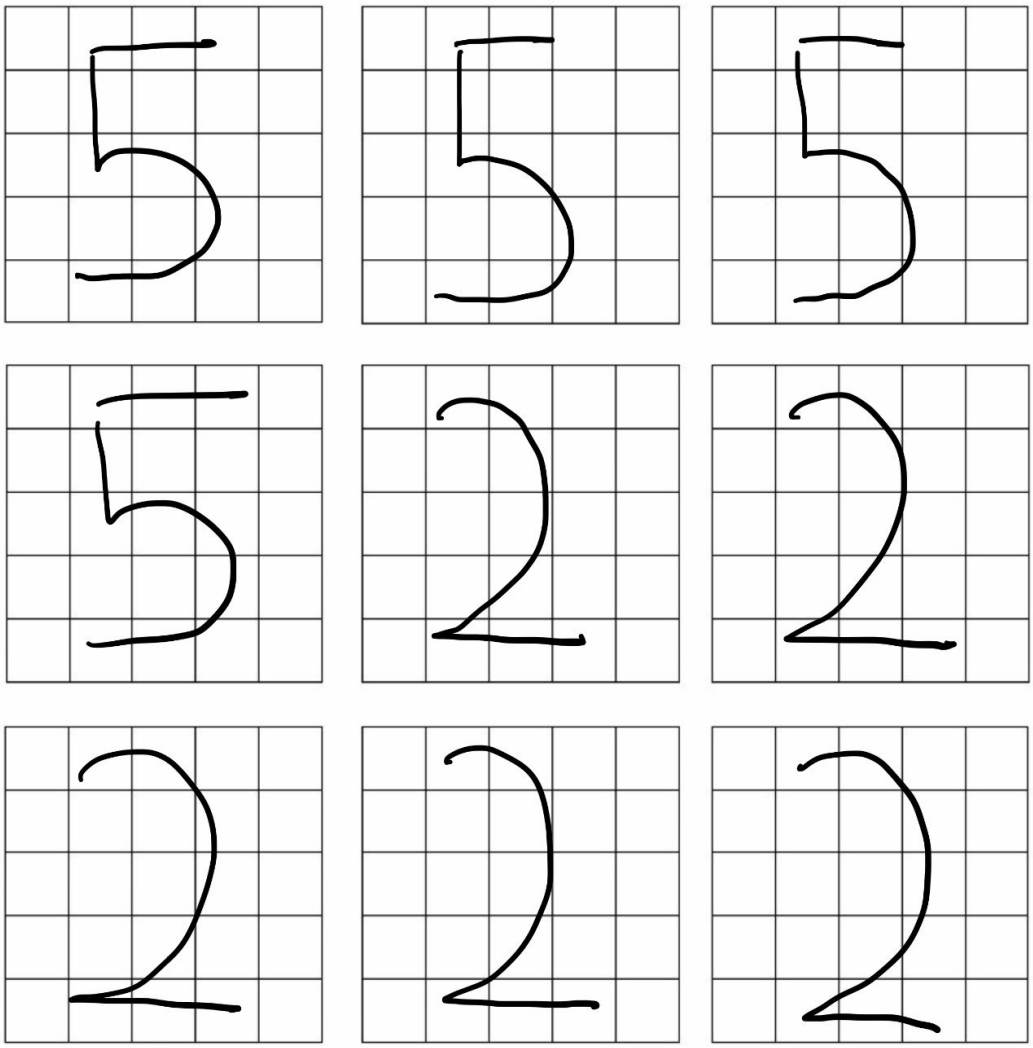
1.

**APPENDIX 1 : Data Collection Grids**



2.

a. Trial 1:7/8 --87.5%
   Trial 2: 7/8 --87.5%
   Trial 3: 6/8 --75%
   Trial 4: 8/8 --100%
   Trial 5: 7/8 -- 87.5%
b. The hopfield network is generally making errors when classifying twos. This makes sense because there is a bit of variation/noise when 2 was recorded in our test. Sometimes, the written number will cross certain parts of the grid and other times it won't which accounts for the 1's in some places where there were 0's or vice versa. Since the

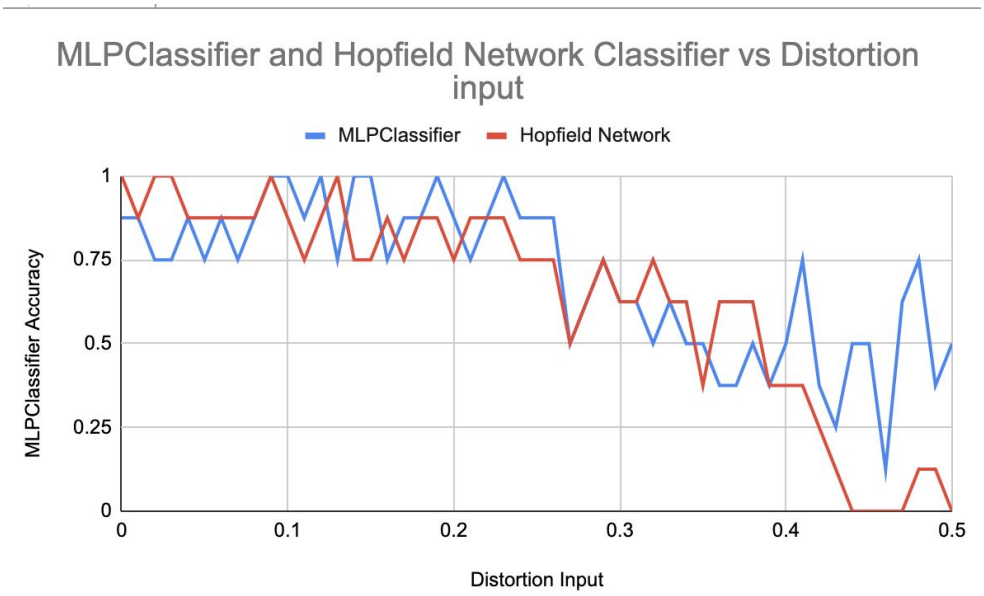
two that I wrote were written with some variation in each test case, our hopfield network will sometimes not be able to recognize if we have a two or not. This is because our perfect classification or 'training data' only has one specific instance for two that will work and even if we do run our retrieve and classify functions we will still not be able to account for too much noise and ultimately converge to the correct classifier. After looking through my data I noticed that my 2 did indeed have some variation/noise across the written set. The 5 on the other hand always classifies correctly because looking back at when I wrote it, there isn't much variation in the grid parts it crosses when I wrote it. In other words, the 1 and 0 pattern is similar with not too much noise to both each other in the test data and in comparison to the training data or perfect instance of 5. A hopfield network converges by doing semi-random updating which is as close as we can get to synchronous updating because it is unrealistic to having all neurons updating at the same rate. We keep on doing this until we converge(don't have any changing nodes) for an arbitrary sequence. This convergence method in itself also accounts for our classification not being perfect since we attempt to the closest sequence-- five or two. If there are more matches to the pattern of 5 we continue converging in that direction even if our classification was supposed to be a two. Too much noise as explained above in our test cases can cause issues like mis-classifying a two as a five or vice versa. A hopfield network cannot adapt to variance as well as other methods since we continue converging to whichever pattern is closest to our trained or perfect case. Despite, the hopfield network not being able to adapt to variance very well, I would like to explain why my classification accuracy score is still very high. Even though I had some variance, it was not enough to mis-classify several of the test cases. To my understanding, in office hours the general trend was that the hopfield network would perform worse than the MLPclassifier. Since my writing didn't have too large of a variance, however, my hopfield network was still able to perform just as well as the MLP.

3.

a. Trial 1: 8/8 --100%
   Trial 2: 6/8 --75%
   Trial 3: 8/8 --100%
   Trial 4: 7/8 --87.5%
   Trial 5: 7/8 --87.5
b. Again in the case of the MLPClassifier, we misclassify the 2's sometimes. This again is accounted for with the variance in the set with respect to the 1's and 0's. Parts of the grids are crossed in some of the test cases and in others they are not. The key difference from the previous explanation on hopfield networks, however, is that the mechanism of convergence for the MLPClassifier is different than that of a hopfield network. Regardless, we still get similar convergence since the test data does not have too much variation(differences in 1's and 0's) to make a significant difference in classification accuracy. In general, however, the MLPClassifier should perform far better than the hopfield network. It is due to the low amount of variance between the data in the

test set that we were able to get similar accuracy in the MLPClassifier and Hopfield Network(since the hopfield network otherwise doesn't do well with high variance among test sets). The MLPClassifier works by using a feedforward artificial neural network which maps input data(features) onto outputs(labels). It is made up of more than one perceptron which allows us to have multiple layers/features. Each feature would be considered a layer and in each layer we have a number to neurons that fire when we consider the impact of a feature on the label(prediction). Rather than just have one perceptron(feature) make the prediction we use several each with their own set of neurons just like a neural network. We use backpropagation and error calculation to vary our input/hidden/output weights until we are able to get a perfect classification. Our goal is to minimize the error so the output can mimic the training cases just as well and even handle variation on them. An MLPClasssifier can handle variation extremely well because we use backpropagation and minimization of error(unwanted/misclassifications of 1's and 0's) to work robustly in a variety of test cases. Since backpropagation is based on a least mean squares algorithm, we have a larger room for error in the misclassification of 1's and 0's to still classify a five or two correctly. It can be compared to how in a support vector machine we have a larger hyper-plane giving us more variance to fit the data except in this case it actually helps us classify our data even better especially when there is variance in the writing of numbers on a grid.

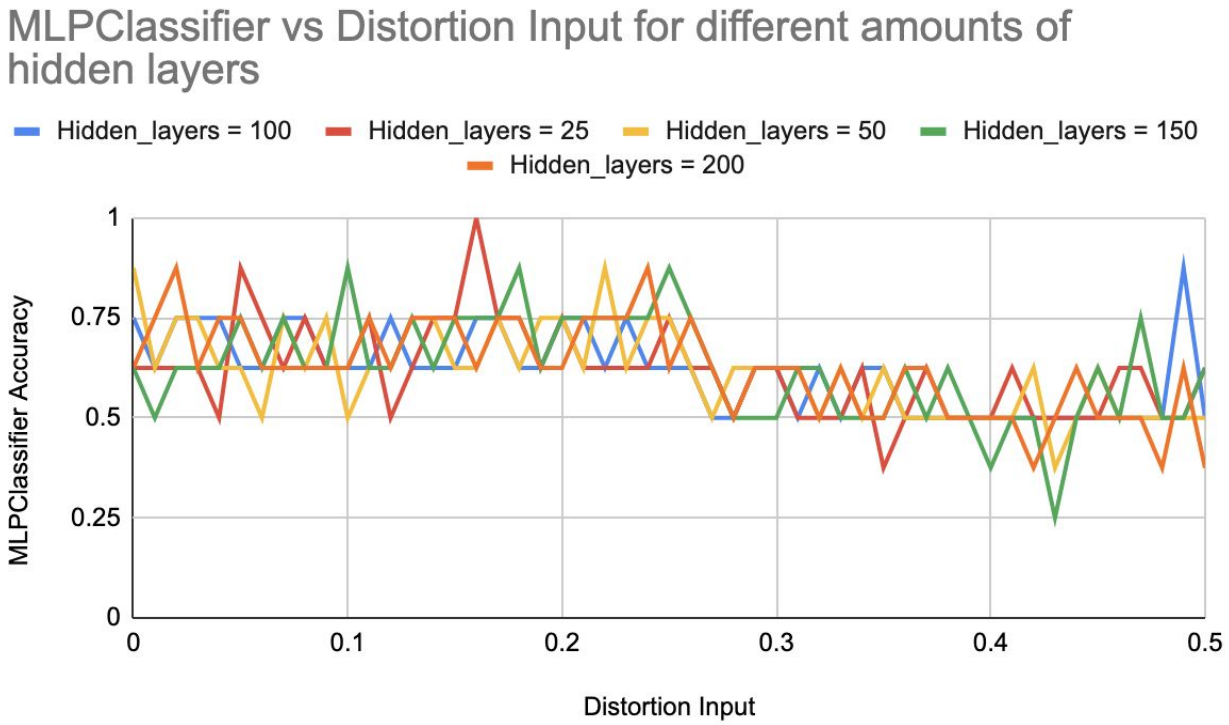
4.




a.

b. In both of these cases distortion completely destroys the classification accuracy as we continue to increase the distortion input to 0.5. The MLPClassifier holds up better than the Hopfield network with increasing distortion inputs which makes sense because Hopfield Networks are not very good at handling variation(as explained in part 2 and 3) whereas MLPClassifiers are better at this.

5.




a.

It appears that increasing the number of hidden layers in an MLP classifier does not make much of a difference when it comes to varying distortion input. First off, it is worth discussing that now we are using both our own identikey input data in addition to some new input that was provided as testing data. This accounts for why our MLPClassifier accuracy is a bit lower than previously with just our own identikey data since there is even more distorted input data being classified lowering the overall classification accuracy. Looking very closely, we can see that as we go to extremes(highs or lows in the number of hidden layers ie 25 and 200) there are more spikes or random extreme variations in the data. This makes sense because as we increase or decrease the number of features(hidden layers) we may overfit or underfit the data. When we overfit the data we may get an extremely high accuracy where there shouldn't be one with the general convergence of the algorithm or an extremely low accuracy if we overfit so much in the wrong direction that we completely ignore any variation whatsoever. The same applies with underfitting where being classified as a 2 or a 5 would be equally as likely because there is no precision or narrowing our search down. We also still notice a general downward trend in the classification accuracy independent of the number of hidden layers. Hidden layer size tend to be optimal when its between the size of the input layer and output layer but this won't matter as much when we have an increasingly distorted input. Overall, hidden layers don't make much of a difference when we are increasing the number of distorted input,however, I am sure without distortion it would be a completely different story. The distortion just causes data points to be all over the place.