Saishnu Ramesh Kumar

300758706

Project 2 – Performance Assessment of Different Algorithms

Abstract:

The goal of this study was to assess the performance of algorithms for handwritten digits by creating a classifier that would allocate them. This was done by using the MNIST database that was downloaded online on Canvas which was provided to us. The dataset is split into two parts, train, and test datasets. Instead of using all the datasets that were included in the file, the program is intended to extract the first 500 examples for each of them. We would then do the following, implement a K Nearest Neighbor (KNN) and K-means algorithm from scratch, and assess their performances. By using the built-in functions of MATLAB, I then had to create a Multilayer Perceptron (MLP) and a Convolutional Neural Network (CNN).

Introduction:

In the K Nearest Neighbor, we had to use the distance between the neighbors to determine their class and to assess the performance of the algorithm with the following k values, k = 1, k = 5, and k = 10. As for K-Means, we had to use the closest example to the centroid to label the cluster and then label the test example with the label from the cluster that it originates from. To assess this version of K-means, it must adapt to supervised classification, where we can assess the following k values when, k = 10, k = 20, and k = 30. Regarding the Multilayer Perceptron, two hidden layer numbers were assigned, 50 and 100, which we have to assess. Finally, for the Convolutional Neural Network, we had to use the Deep Learning Toolbox to construct the algorithm. Do note that for all algorithms mentioned, we had to fulfill the following criteria:

- Obtain their accuracies.
- Get the confusion matrices.
- Calculate their True Positives (TP) and False Negatives (FN) for the confusion matrices.
- Obtain the graph (for the CNN algorithm only).

The results from the study conducted were interesting and well predicted considering the types of algorithms that we are testing here. But one of the key findings I came across was that in the K-Means algorithm, its accuracy got far better as its k values were incremented by 10 each time. Nonetheless, it can be agreed that K-Means has the worse accuracy due to it being an unsupervised algorithm that will group data into clusters depending on how similar they are. This then does not aim to minimize its classification errors. Regarding K Nearest Neighbor and the Multilayer Perceptron, they can be placed in the middle region regarding their accuracies. This is because KNN is simpler and tends to work better with smaller-sized datasets, but the performance of the algorithm is limited due to its dimensionality. MLP on the other hand is far more accurate and powerful than KNN but the accuracy is still limited due to the complexity of the problems and the varying number of hidden layers used. The most accurate among all is the Convolutional Neural Network as it uses multiple convolution and pooling operations to extract features used in classification which allows it to obtain higher accuracies in a wider range.

Method Descriptions:

Each algorithm was implemented in its respective file and was called into one main file that would output the required information needed. The main file also includes loading in the train and test datasets given and the number of examples that we are going to use for this project (500). Four functions have been implemented, KNN, KMEANS, MLP, and CNN respectively.

1. K-Nearest Neighbor (KNN):

The KNN function classifies data points into different classes, and it takes in four input arguments, “train_data”, “train_labels”, “test_data”, and “test_labels” which have been extracted from the train and test datasets located in the main file of this program. The first function initializes the set of k values to be assessed for the accuracy of KNN. The number of examples from the train and test datasets is then obtained using the size function. A for loop is created to iterate over the k values where for each value of k, the function initializes an empty column vector called “predicted_labels” of length “num_examples_test” to store the predicted labels for the text examples. The function then iterates through each test example using another for loop. For each test example, the Euclidean distance between the test example and all the training examples is calculated using the sqrt and sum functions. It then moves on to locating the indices of the KNN using the mink function. Using the mode function, the most common label is then found from the training example that is the closest to the current test example. The predicted label for the current test example is then assigned to the “predicted_labels” vector. Once it iterates through all the test examples the confusion matrix is then created to display a figure alongside its accuracy and will calculate the True Positives (TP) and False Negatives (FN).

2. K-Means:

The K-Means function groups a set of data points into the k clusters based on their similarity and the function here takes in three input arguments, “data matrix X”, “k number of clusters”, and “maximum number of iterations”. We first initialize the cluster index, centroids, and distance to each centroid to zero. Then it will randomly select k examples from X and set them as the initial centroid. The algorithm will then iteratively assign each example to the closest centroid possible, adjust its centroid value, and check if the cluster index for each example has been changed from the previous iteration. If there is no change, the algorithm will stop. The function will then finally return the cluster index for each example, the matrix that contains the centroid for each cluster, and the distance between the example to the different centroids. In the main file, I have implemented the K-Means to classify data points into different clusters and then evaluate the accuracy for each k value. We first set the script to a random seed of 0 to ensure that the results will be the same every time the algorithm is run. Then it defines the array of “k_values” that we are going assess and the maximum number of iterations are set. The script will then loop using a for loop to go through each "k_values" and runs the K-Means algorithm for both the "train_data" and "test_data" to obtain the cluster indices and centroids respectively. Then a label is assigned to each data point in the cluster based on the most common label found in each train and test cluster. We can then compute the accuracy of the algorithm and the confusion matrix is then created for each k value and is displayed along with each accuracy along with the TP and FN.

3. Multilayer Perceptron (MLP):

The MLP function classifies data points into different classes, and it takes five input arguments, "train_data", "train_labels", "test_data", "test_labels", and "hidden_size". The purpose of this function is to train a feedforward neural network containing one hidden layer. The labels for both datasets are taken and are then changed using the categorical function to make sure that the fitcnet function can use the data provided. The fitcnet function is then used to train the feedforward neural network on the "train_data" and "train_labels". Then, the parameter, "LayerSizes" specifies the number of neurons that are initialized for the hidden layers. After that, the predict function predicts the labels for the "test_data" by using the trained neural network, and the predicted and actual labels are used for the confusionmat function. The accuracy is then computed for MLP, and the confusion matrix is then created and displayed with its accuracy. In the main file, the hidden_size has been set to 50 and 100 and is assigned each time before calling the mlp function. It will then finally display the confusion matrix alongside its accuracy for each hidden layer and calculates the TP and FN.

4. Convolutional Neural Network (CNN):

The CNN function will classify the images of the digits into different classes and then evaluate the algorithm's accuracy. The function starts off by loading the digit sample data as an image datastore and then displays some of the images there. It will then calculate the number of images for each category and then divide the data into training and validation datasets. It will then continue to define the CNN layout by using a series of layers that include the convolutional layers, batch normalization layers, ReLU activation layers, max-pooling layers, fully connected layer, softmax layer, and lastly a classification layer. It will then display the network and define the options for training the network. The training set data is used to train the network and it will predict the labels for the validation data by using the trained neural network and it will then calculate the final validation accuracy. The function will display the confusion matrix, along with its accuracy, graph, and handwritten digits alongside calculating its TP and FN.

Results Analysis:

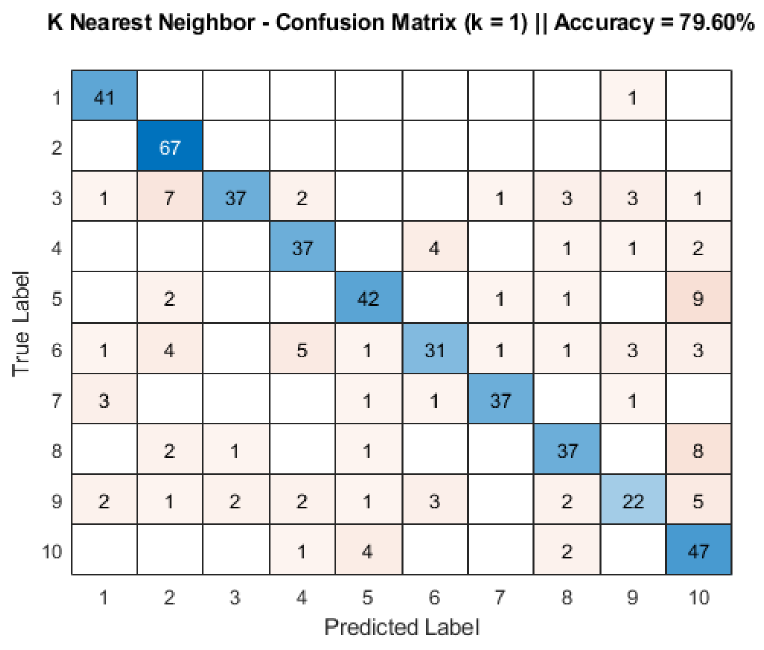
To further analyze the confusion matrices, we would need to calculate the true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each of the matrices shown. The formulas for calculating these are as follows:

- **True Positive (TP)** = sum(diag(C)). Takes the sum of the diagonal results from the confusion matrix.

- **True Negative (TN)** = sum(all elements in C) – TP – FP – FN. Takes the sum of all elements in C then subtracts it from TP, FP, and FN.
- **False Positive (FP)** = sum(columns) – TP. Takes the sum of all columns from the matrix then subtracts it with TP.
- **False Negative (FN)** = sum(rows) – TP. Takes the sum of all rows from the matrix then subtracts it with the TP.

Do note that considering the datasets we are using for this assessment, there are no False Positives and True Negatives. This is because FP only happens when the algorithm predicts the right label for the image but since the label does not belong to that specific image and the datasets used already have class labels assigned to them, FP will not happen. As for TN, this would only occur when there are several cases where the classifier can correctly identify something not belonging to the class. Therefore, this will not be the case for the data used as we are attempting to match the "test_data" to which every label set belongs to their specific class.

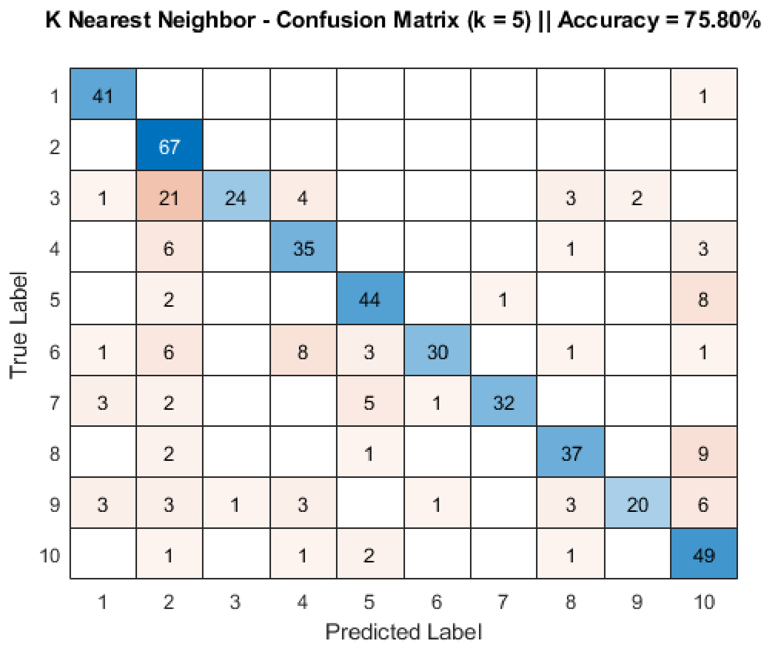
1. K-Nearest Neighbor (KNN):



K Nearest Neighbor - Confusion Matrix (k = 1) || Accuracy = 79.60%

True Label \ Predicted Label	1	2	3	4	5	6	7	8	9	10
1	41								1	
2		67								
3	1	7	37	2			1	3	3	1
4				37		4		1	1	2
5		2			42		1	1		9
6	1	4		5	1	31	1	1	3	3
7	3				1	1	37		1	
8		2	1		1			37		8
9	2	1	2	2	1	3		2	22	5
10				1	4			2		47

When k = 1, we can observe that the accuracy obtained is 79.60%. The confusion matrix shows us that the highest correctly classified diagonal for a particular class is 67 instances and the lowest is 22. The off-diagonal element that is the highest is 9 instances and the lowest is 1.



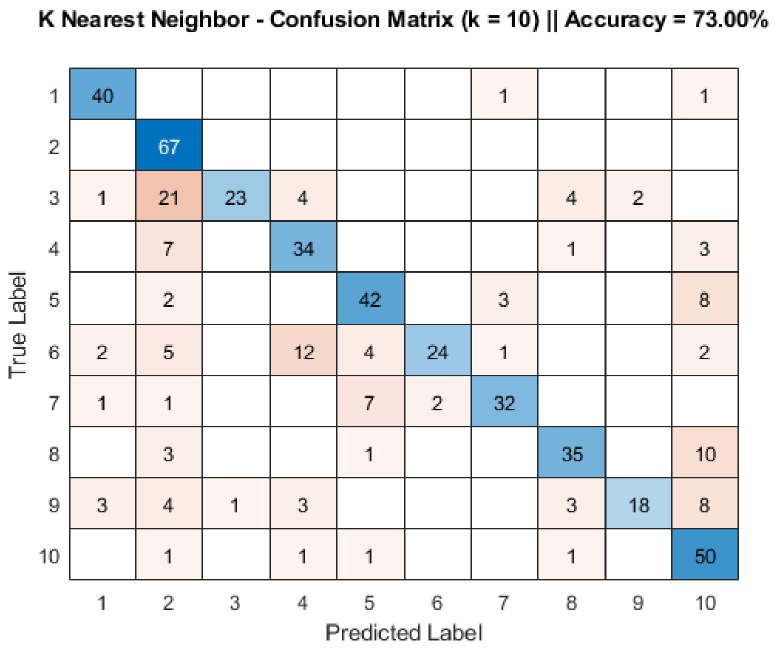
K Nearest Neighbor - Confusion Matrix (k = 5) || Accuracy = 75.80%

True Label

	1	2	3	4	5	6	7	8	9	10
1	41									1
2		67								
3	1	21	24	4				3	2	
4		6		35				1		3
5		2			44		1			8
6	1	6		8	3	30		1		1
7	3	2			5	1	32			
8		2			1			37		9
9	3	3	1	3		1		3	20	6
10		1		1	2			1		49

Predicted Label

When k = 5, we notice that the accuracy is 75.80%. The confusion matrix displays that the highest correctly classified diagonal for a particular class is once again 67 instances with the lowest being 20. The highest off-diagonal element displayed is 21 instances and the lowest shown is 1.



K Nearest Neighbor - Confusion Matrix (k = 10) || Accuracy = 73.00%

True Label

	1	2	3	4	5	6	7	8	9	10
1	40						1			1
2		67								
3	1	21	23	4				4	2	
4		7		34				1		3
5		2			42		3			8
6	2	5		12	4	24	1			2
7	1	1			7	2	32			
8		3			1			35		10
9	3	4	1	3				3	18	8
10		1		1	1			1		50

Predicted Label

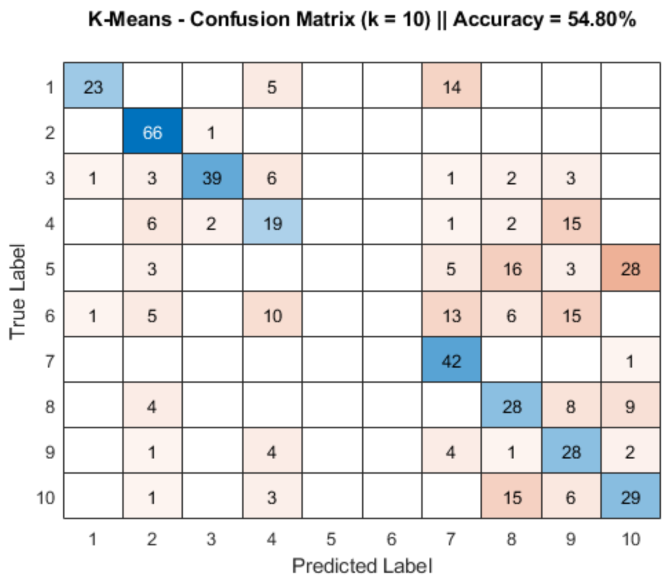
When k = 10, the accuracy gained is 73%. The confusion matrix tells us that the highest correctly classified diagonal is 67 instances and the lowest is 18. Once again, the highest off-diagonal displayed is 21 instances and the lowest is 1.

Overall: From the accuracy obtained for the KNN algorithm, we can see that the accuracy of KNN falls between 70% to 80%. In each increment, the accuracy starts to reduce the higher the k value is set, this is due to the bias-variance trade-off which refers to the balance between the model's ability to fit the training data and its ability to generalize new, unseen data. Therefore, the smaller the k value is set, the higher the accuracy of KNN will be. The average accuracy for the three k values is about 76.13%. The overall confusion matrix from all three outcomes shows that the highest correctly classified diagonal is in class 2 with 67 instances, and it was in the same class for all outcomes. The lowest for that ranges anywhere below 25 instances. The highest off-diagonals are 9, 21, and 21 instances respectively in order of the figures displayed. We consider the diagonal elements in blue as the true positives because they belong to a particular class and are correctly classified into the respective classes. The off-diagonal elements are instances that have been misclassified for each class and are known as the false positives or false negatives, this depends on if the instances are classified as being set to the wrong class or not being set to the correct class respectively. We can refer to the table below that represents the results obtained for TP and FN using the formulas mentioned above:

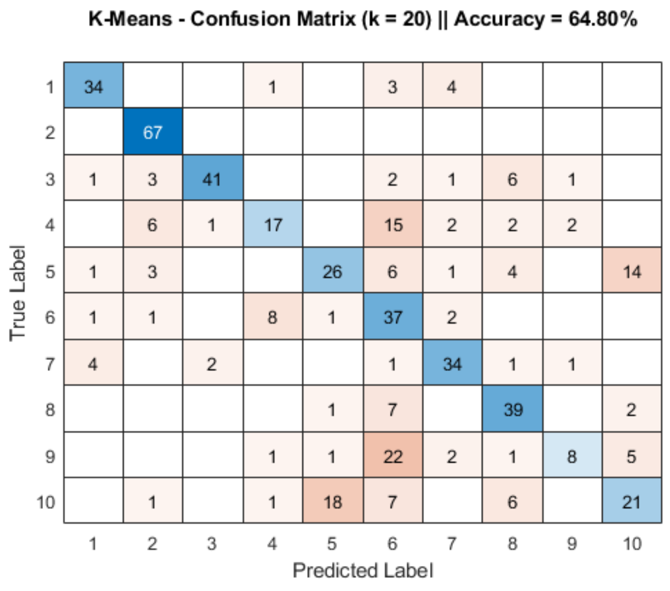
K Values	**1**	**5**	**10**
True Positives (TP)	398	379	365
False Negatives (FN)	102	121	135

We can see that when k = 1 the TP is 398 with the FN being 102. When k = 5, the TP is 379 and the FN is 121. Lastly when k = 10, the TP is 365 with FN calculated to be 135.

2. K-Means:



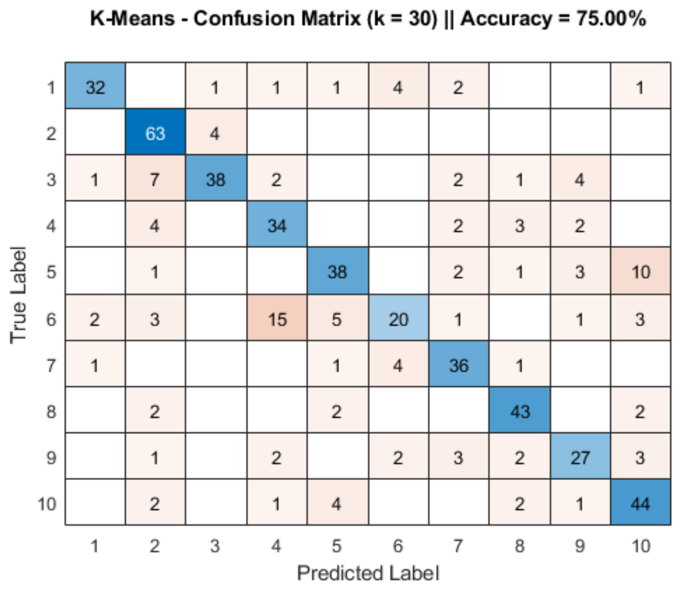
When k = 10, the accuracy obtained is around 54.80%. The confusion matrix shows that the highest number of instances is 66 and the lowest is 19. The highest off-diagonal instance is 28 and the lowest is 1.



K-Means - Confusion Matrix (k = 20) || Accuracy = 64.80%

True Label \ Predicted Label	1	2	3	4	5	6	7	8	9	10
1	34			1		3	4			
2		67								
3	1	3	41			2	1	6	1	
4		6	1	17		15	2	2	2	
5	1	3			26	6	1	4		14
6	1	1		8	1	37	2			
7	4		2			1	34	1	1	
8					1	7		39		2
9				1	1	22	2	1	8	5
10		1		1	18	7		6		21

When k = 20, the accuracy obtained is 64.80%. The confusion matrix on the other hand shows the highest instance of true positives is 67 with the lowest being 8. The highest off-diagonal instance is 22 and the lowest is 1.



K-Means - Confusion Matrix (k = 30) || Accuracy = 75.00%

True Label \ Predicted Label	1	2	3	4	5	6	7	8	9	10
1	32		1	1	1	4	2			1
2		63	4							
3	1	7	38	2			2	1	4	
4		4		34			2	3	2	
5		1			38		2	1	3	10
6	2	3		15	5	20	1		1	3
7	1				1	4	36	1		
8		2			2			43		2
9		1		2		2	3	2	27	3
10		2		1	4			2	1	44

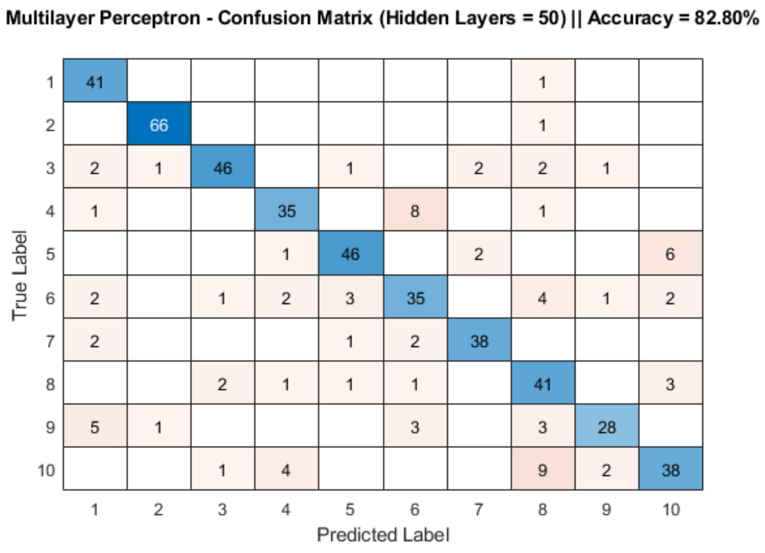
When k = 30, the accuracy shown is 75%. The confusion matrix tells us that the highest instance is 63 with the lowest of true positives being 20. The highest off-diagonal instance is 15 and with the lowest being 1.

Overall: From the accuracy obtained for K-Means, ranges from around 50 to 80%. The results do seem interesting because since this algorithm is unsupervised, the higher the k values, the less accurate it will be because it decreases the clustering accuracy. With the increment of the k values, it leads to overfitting and the algorithm would end up creating too many clusters and assigning too few data points for each of them. But in this case, since we are attempting to use the k-means to adapt to supervised learning, we essentially turn this algorithm into a classification algorithm where we intend to predict the class labels of new data points that are based on the labeled data provided. Depending on the quality of the clustering and accuracy of the class labels assigned for each cluster, and if the clustering is accurate along with the class labels being correctly assigned, we would then be able to get higher accuracies as seen from the data provided above. Therefore, for this instance of K-Means, the higher the k value set, the higher the accuracy is. The average accuracy for the three values that we tested is 64.87%. As for the confusion matrices, the three outcomes have shown that the highest correctly classified diagonal is in class 2 with the instances being 66, 67, and 63 respectively. The lowest for them ranges below 30 instances. The highest off-diagonal instances are 28, 22, and 15 respectively. One unique difference K-Means had regarding the confusion matrix when k = 10, is that in columns 5 and 6, they were empty. We have to consider that K-Means is still an unsupervised learning algorithm whereby it does not have any predefined labels or classes for the data which could suggest that the two columns have no instances assigned to them, especially if the number of clusters is set too high or if there is an unbalance of data. We can refer to the table below that represents the results obtained for TP and FN using the formulas mentioned above:

K Values	**10**	**20**	**30**
True Positives (TP)	274	324	375
False Negatives (FN)	226	176	125

We can see that the TP for k = 10 is 274 with the FN being 226. When k=20, the TP is 324 with FN calculated to be 176. Finally, when k = 30, the TP is 375 with FN resulting to be 125.

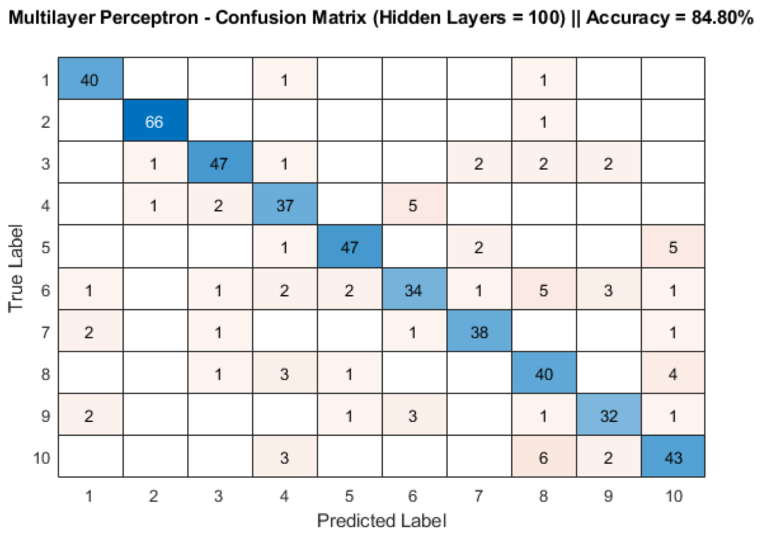
3. Multilayer Perceptron (MLP):



Multilayer Perceptron - Confusion Matrix (Hidden Layers = 50) || Accuracy = 82.80%
True Label
Predicted Label

	1	2	3	4	5	6	7	8	9	10
1	41							1		
2		66						1		
3	2	1	46		1		2	2	1	
4	1			35		8		1		
5				1	46		2			6
6	2		1	2	3	35		4	1	2
7	2				1	2	38			
8			2	1	1	1		41		3
9	5	1				3		3	28	
10			1	4				9	2	38

When hidden layers = 50, the accuracy obtained is 82.80%. The confusion matrix shows us that the highest number of correctly classified instances is 66. The lowest for this would be 28 instances. The highest off-diagonal instance is 9 with the lowest being 1.



Multilayer Perceptron - Confusion Matrix (Hidden Layers = 100) || Accuracy = 84.80%
True Label
Predicted Label

	1	2	3	4	5	6	7	8	9	10
1	40			1				1		
2		66						1		
3		1	47	1			2	2	2	
4		1	2	37		5				
5				1	47		2			5
6	1		1	2	2	34	1	5	3	1
7	2		1			1	38			1
8			1	3	1			40		4
9	2				1	3		1	32	1
10				3				6	2	43

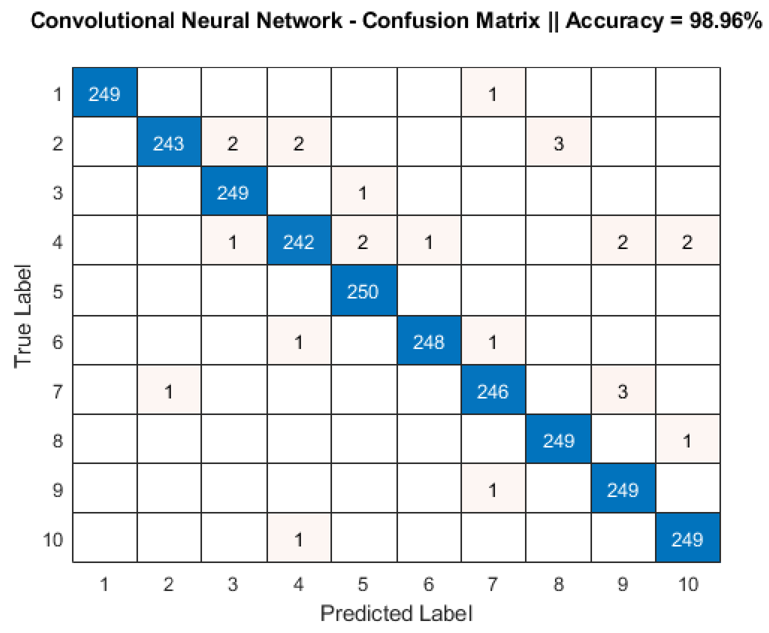
When hidden layers = 100, the accuracy is 84.80%. The confusion matrix tells us that the highest number of correctly classified instances is 66 as well. The lowest is 32 instances. The highest off-diagonal instance is 6 with the lowest as 1.

Overall: From the accuracy obtained for the MLP, the accuracy does not fall below 80%. This does depend on the number of hidden layers we are using as increasing the size of the hidden layers can improve the accuracy up to a certain point. This is because, after further increments, the size of the hidden layer may not have significant improvements in its accuracy and could potentially lead to overfitting. The reason why MLP does this is that increasing the size of the hidden layers increases the capacity of the network to learn complex patterns in the data. But if the hidden layer size is too big, it may start memorizing the data instead of learning the patterns and this leads to overfitting and poor generalization to new data. Therefore, it can be said that there is a limit to the number of hidden layers we can assign until the accuracy starts to not show significant increments. As for the confusion matrices, the highest correctly classified instances for both hidden layers are in class 2, which is 66. The lowest falls below 35 instances. As for the highest off-diagonal instances, they would be 9 and 6 respectively. It does look interesting how similar the confusion matrices for both these hidden layers are, despite their layers being doubled. This would depend on the complexity of the datasets used with the specific problem, hence resulting in some similarities. We can refer to the table below that represents the results obtained for TP and FN using the formulas mentioned above:

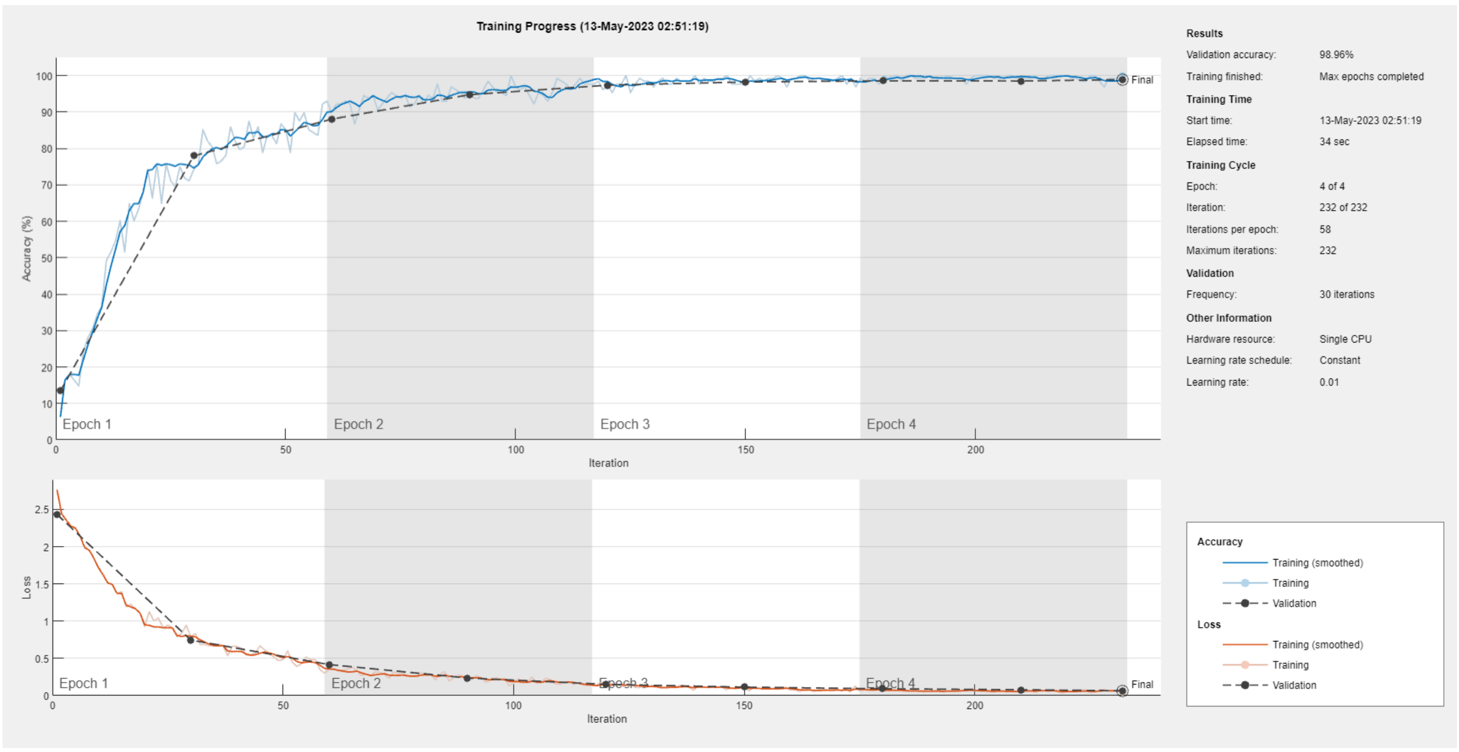
Hidden Sizes	**50**	**100**
True Positives (TP)	414	424
False Negatives (FN)	86	76

We can see that the TP for Hidden Size 50 is 414 with the FN being 86 and when Hidden Size 100, the TP is 424 with FN calculated to be 76.

4. Convolutional Neural Network (CNN):



With CNN, we got an accuracy of 98.96%. The confusion matrix shows that the highest number of correctly classified instances is 250 and the lowest is 243. The highest off-diagonal instance is 3 with the lowest being 1.



Results

Validation accuracy:	98.96%
Training finished:	Max epochs completed

Training Time

Start time:	13-May-2023 02:51:19
Elapsed time:	34 sec

Training Cycle

Epoch:	4 of 4
Iteration:	232 of 232
Iterations per epoch:	58
Maximum iterations:	232

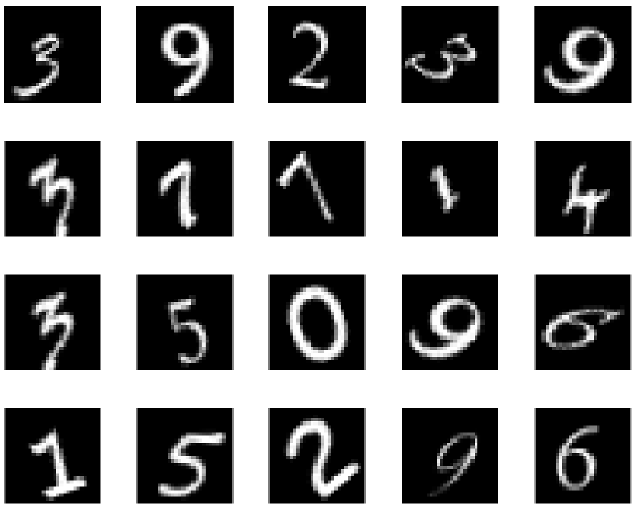
Validation

Frequency:	30 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.01

The graph obtained shows us how during each iteration the accuracy of the data increases and then slowly starts to even out at the very end of the graph. Initially, we can see that it was gradually learning up till around the 60th iteration when it starts to slow down and make the graph flatter. On the loss graph, it shows a drastic decrease from iteration 0 to about 30 and after the 60th iteration, it slowly starts decreasing until it reaches the end about a number very close to 0.



The handwritten digits obtained show us the 20 digits that have been created by the algorithm.

Overall: With the implementation of CNN, we can see how accurate this algorithm is by using deep learning as it produces an accuracy of 98.96%. The confusion matrix shows us that almost all the instances for each class are filled despite there being some mismatched instances for each class. It was also able to display 20 of the handwritten digits it was able to learn from training. We can refer to the table below that represents the results obtained for TP and FN using the formulas mentioned above:

CNN	
True Positives (TP)	2474
False Negatives (FN)	26

We can see that the TP for CNN is 2,474 and the FN calculated is 26.

<u>Conclusion:</u>

From the results I have obtained from this study, we can see which algorithms perform the worst and the best. Overall, it can be concluded that K-Means would still result in having the worst accuracies despite that in this study it is set to be a classification algorithm hence why the results are a little different than anticipated. One of the confusion matrices for K-Means was also interesting but knowing that we are attempting to make the algorithm adapt to a supervised learning environment could have been a cause for that result. The K Nearest Neighbor and Multilayer Perceptron algorithms worked as intended and displayed the expected results as they both seem to fit the middle criteria of being a somewhat balanced and better working algorithm than K-Means. It was interesting to see how KNN reacted when the k values were increased as the accuracy started to drop. As for the MLP, I found that even with a doubled hidden layer size used, the results did not differ or drift away as much as I had initially thought they would. Regarding CNN, it can be said to be the most efficient among all due to its very high accuracy.

When it came to the confusion matrices obtained from the algorithms, we are then able to better identify the elements of the various instances that are correctly classified diagonally as compared to the off-diagonal elements that are misclassified. When we analyze the off-diagonal elements, we can identify which classes are confused with each other and where errors are being made by the algorithm itself. Most of the highest true positives obtained were in class 2 for almost every confusion matrix. For CNN the highest true positives were found in class 5 and there were minimal off-diagonal elements that were misclassified.

Some possible improvements I believe could have been made was to possibly grab a bigger dataset instead of just grabbing the first 500 examples to likely get a better reading all around. We could have also assessed the performances for both K Nearest Neighbor and K-Means better by having more k values to test. As for the Multilayer Perceptron, we could have possibly also added more hidden layers to see where the accuracies may stop to differ as much. For the Convolutional Neural Network, we could have allowed it to go through more iterations and notice the graph changes over time.