COMPARATIVE ANALYSIS OF MACHINE LEARNING ALGORITHMS FOR HANDWRITTEN DIGIT IDENTIFICATION.

M.A.G.K. Muthunayaka
EUSL/TC/IS/2017/COM/35

This project submitted to the Department of Computer Science, Faculty of Applied Science, Trincomalee Campus, Eastern University of Sri Lanka, for the award of the degree of Bachelor of Computer Science

23/03/2023

Declaration

This is to certify that this project document entitled "Comparative analysis of machine learning algorithms for handwritten digit identification" submitted by M.A.G.K.Muthunayaka for the degree of bachelor of Computer Science is a record of project work carried out by her under our guidance and direct supervision and that it has not been previously formed the basis for the award of any degree, diploma, associateship, fellowship or any other similar title.

This is also to certify the document represent the original independent work of candidate.

………………………………………………….   ………………………………………….

Signature of Supervisor   Date

Mrs.K.Krishnaraj

Head of the Department

Department of Computer Science

Faculty of Applied Science

Trincomalee Campus, Eastern University, Sri Lanka

………………………………………………….   ………………………………………….

Signature of the student   Date

M.A.G.K. Muthunayaka(EUSL/TC/IS/2017/COM/35)

Department of Computer Science

Faculty of Applied Science

Trincomalee Campus, Eastern University, Sri Lanka

Abstract

Handwritten digit recognition is an important task in the field of computer vision and has numerous applications such as automatic number plate recognition, signature verification, and postal automation. In this study, ten different machine learning algorithms were tested on the MNIST dataset to determine the most effective algorithm for recognizing handwritten digits. The algorithms tested were the Logistic Regression classifier, Support Vector Machine (SVM) classifier, Random Forest Classifier, K-Nearest Neighbors (KNN) classifier, Naïve Bayes Classifier, Decision Tree Classifier, Gradient Boosting Classifier, Simple Neural Network using Keras, Multilayer Perception (MLP) Neural Network, and Convolutional Neural Network (CNN) using Keras.

The aim of the study was to determine which of these algorithms is the most effective for recognizing handwritten digits in the MNIST dataset. The study authors used the accuracy metric to evaluate the performance of each algorithm. The results of the study showed that the CNN was the most effective algorithm for recognizing handwritten digits in the MNIST dataset, with an accuracy of 99.06%. The MLP Neural Network was the second-best algorithm, with an accuracy of 98.04%, while the SVM classifier had an accuracy of 97.64%.

The Logistic Regression classifier had an accuracy of 92.08%, while the Naïve Bayes Classifier had an accuracy of 55.16%. The Decision Tree Classifier had an accuracy of 85.83%, which is better than the Logistic Regression classifier, but not as good as the other algorithms tested. The Gradient Boosting Classifier had an accuracy of 94.54%, while the Simple Neural Network using Keras had an accuracy of 96.68%. The Random Forest Classifier and the KNN classifier had accuracies of 96.75% and 97.13%, respectively, which are comparable to the Simple Neural Network.

The study authors note that the CNN is the most effective algorithm for recognizing handwritten digits in the MNIST dataset, which is not surprising since CNNs are designed for image recognition tasks, and the MNIST dataset contains images of handwritten digits. The MLP Neural Network and the SVM classifier also performed well and may be suitable alternatives if the CNN is not available or if there are specific constraints on the computational resources required for the task.

The study has some limitations, including that the algorithms were only tested on the MNIST dataset and that no statistical tests were performed to determine the significance of the differences in performance between the algorithms.

In conclusion, this study compared ten different machine learning algorithms for recognizing handwritten digits in the MNIST dataset. The results showed that the CNN is the most effective algorithm for this task, followed by the MLP Neural Network and the SVM classifier. These findings have implications for the development of automated systems that require the recognition of handwritten digits. Further research is needed to determine the performance of these algorithms on other datasets and to optimize their hyperparameters for the task. This study provides a useful starting point for future research in this area.

The Table of Contents

Chapter 01: Introduction

Project Overview

The problem of handwritten digit recognition is considered important in the field of image processing and machine learning, with applications in various domains such as postal automation, finance, and biometrics. In this research project, a comparative analysis of different machine learning algorithms for handwritten digit recognition will be conducted, with the aim of identifying the most effective approach.

The most effective algorithm for handwritten digit recognition will be evaluated for its accuracy, using ten popular machine learning algorithms: Logistic Regression classifier from scikit-learn, Support Vector Machine (SVM) classifier, Random Forest Classifier, K-Nearest Neighbors (KNN) classifier, Naïve Bayes Classifier, Decision Tree Classifier, Gradient Boosting Classifier, Simple Neural Network using Keras, Multilayer Perception (MLP) Neural Network, Convolutional Neural Network (CNN) using Keras.

A benchmark dataset will be used to carry out this analysis, and the performance of these algorithms will be compared in terms of classification accuracy, computational efficiency, and other relevant metrics. The intended beneficiaries of this work include researchers, developers, and practitioners in the fields of image processing and machine learning, as well as organizations that rely on automated recognition systems for digitized documents or handwritten signatures. By identifying the most effective algorithm for handwritten digit recognition, the accuracy and efficiency of such systems can be improved, which in turn can reduce errors, save time, and increase productivity.

The scope of this project is limited to the comparative analysis of ten machine-learning algorithms for handwritten digit recognition, using a publicly available benchmark dataset. No new algorithms or datasets will be developed, nor will the algorithms be tested on different types of data image-processing tasks. The approach to carrying out this project involves several steps.

First, a suitable benchmark dataset will be selected and preprocessed to extract features that are relevant for digit recognition. Then, the ten machine learning algorithms will be implemented using open-source libraries and trained on the dataset. Their performance will be evaluated using various metrics and compared to identify the most effective algorithm.

This work is based on the assumptions that a suitable benchmark dataset is available, the selected machine learning algorithms are effective for handwritten digit recognition, and these algorithms can be implemented and evaluated using open-source libraries. It is also assumed that the results of this analysis can be generalized to other datasets and recognition tasks with similar characteristics. In summary, the aim of this research project is to compare the performance of different machine learning algorithms for handwritten digit recognition, with the goal of identifying the most effective approach.

The intended audience or beneficiaries of this work include researchers, developers, and practitioners in the fields of image processing and machine learning, as well as organizations that rely on automated recognition systems for digitized documents or handwritten signatures. The scope of this project is limited to the comparative analysis of ten machine learning algorithms using a benchmark dataset, and the approach involves implementing and evaluating these algorithms using open-source libraries.

Background

Handwritten digit recognition is an important task with many practical applications in various industries such as postal services, banking, and document digitization. Despite the availability of several machine learning algorithms for handwritten digit recognition, achieving high accuracy and performance is still a challenge due to variations in writing styles. Therefore, this research aims to perform a comparative analysis of different machine learning algorithms for handwritten digit recognition to identify the best algorithm for this task.

The problem statement of the research is to identify the best machine learning algorithm for handwritten digit recognition by performing a comparative analysis of several algorithms, given the challenge of achieving high accuracy and performance due to variations in writing styles, especially with the use of large datasets.

Chapter 02: Related Work

- (Dixit et al.) used Support Vector Machines (SVM), Multi-Layer Perceptron (MLP), and Convolutional Neural Networks (CNN) to illustrate handwritten digit recognition with the help of MNIST datasets. Dixit et al.'s research may have been unsuitable or insufficient for this particular case because they use a single dataset, did not compare their results with other state-of-the-art algorithms, and their study was outdated.[1]

- (Karakaya et al.) tried to use machine learning algorithms such as Support Vector Machine (SVM), Decision Tree, Random Forest, Artificial Neural Networks (ANN), K-Nearest Neighbor (KNN) and K- Means Algorithm for the purpose of handwritten character recognition. [2]
  Karakaya et al.'s research may have been unsuitable or insufficient for this particular case because it focused on the recognition of handwritten characters, used a limited dataset, did not consider the impact of variations in writing styles, and did not use the latest machine learning algorithms and techniques.

- (Shamim et al.) presented an approach to off-line handwritten digit recognition based on different machine learning techniques such as Multilayer Perceptron, Support Vector Machine, Naïve Bayes, Bayes Net, Random Forest, J48 and Random Trees. [3]
  Shamim et al.'s research was unsuitable or insufficient for this particular case because it only focused on off-line handwriting recognition, used a small and limited dataset, did not consider the impact of variations in writing styles, and did not use the latest machine learning algorithms and techniques.

Chapter 03: Tools and Techniques

• Language – Python

Python is an interpreted, high-level, general purpose programming language. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

• **Anaconda Distribution**

Anaconda Distribution is a Python/R data science distribution that contains conda, a package and environment manager, which helps users manage a collection of over 7,500+ open-source packages available to them. Anaconda is free, easy to install, and offers free community support.. The distribution includes data-science packages suitable for Windows, Linux, and macOS.

• IDE - Jupyter Notebook

Jupyter is a free, open-source, interactive web tool known as a computational notebook, which researchers can use to combine software code, computational output, explanatory text and multimedia resources in a single document. Computational notebooks have been around for decades, but Jupyter in particular has exploded in popularity over the past couple of years. This rapid uptake has been aided by an enthusiastic community of user–developers and a redesigned architecture that allows the notebook.

Chapter 04: Methodology

4.1. Dataset

For this project, the dataset used is MNIST Dataset. Modified National Institute of Standards and Technology (MNIST) is a database which consist of handwritten digits. MNIST Database provides simple statics classification tasks for researchers to help them to analyze machine learning and pattern recognition techniques. It consist of 60,000 examples as train set, and 10,000 examples as test set. It is a subset of a larger set that is available from NIST.

The dataset is based on grey-scale images of handwritten digits where each image is of the form 28×28 pixel in height and width. Each pixel has a value associated with it where dark pixel is represented by 0 and a white pixel is represented by 255. Both the train and test dataset have 785 columns where the first column is ‘label’ which represents the handwritten digit (a number from 0 to 9) and the remaining 784 values represents pixel values. For testing the data is separated from labels to predict the value.

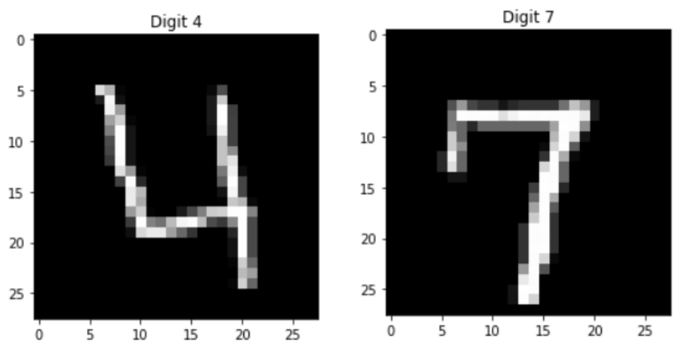


Figure 4.1.1 : Sample images from MNIST dataset

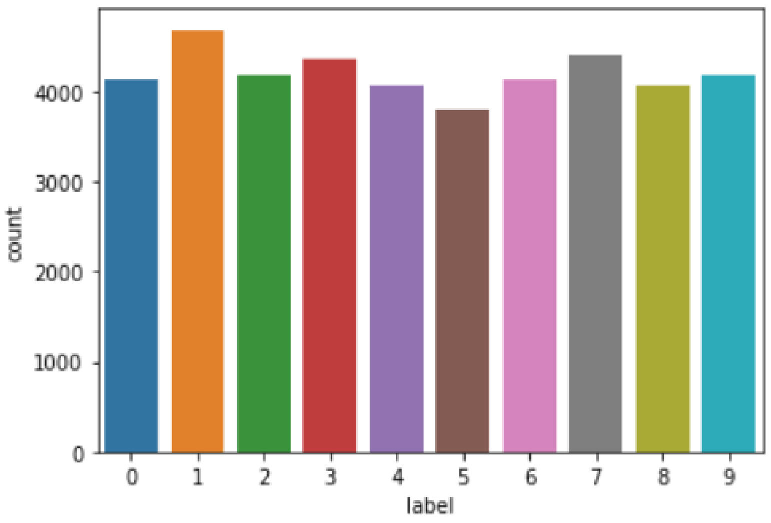


Figure 4.1.2 : number of class and counts in the datasets

4.2. Methodology

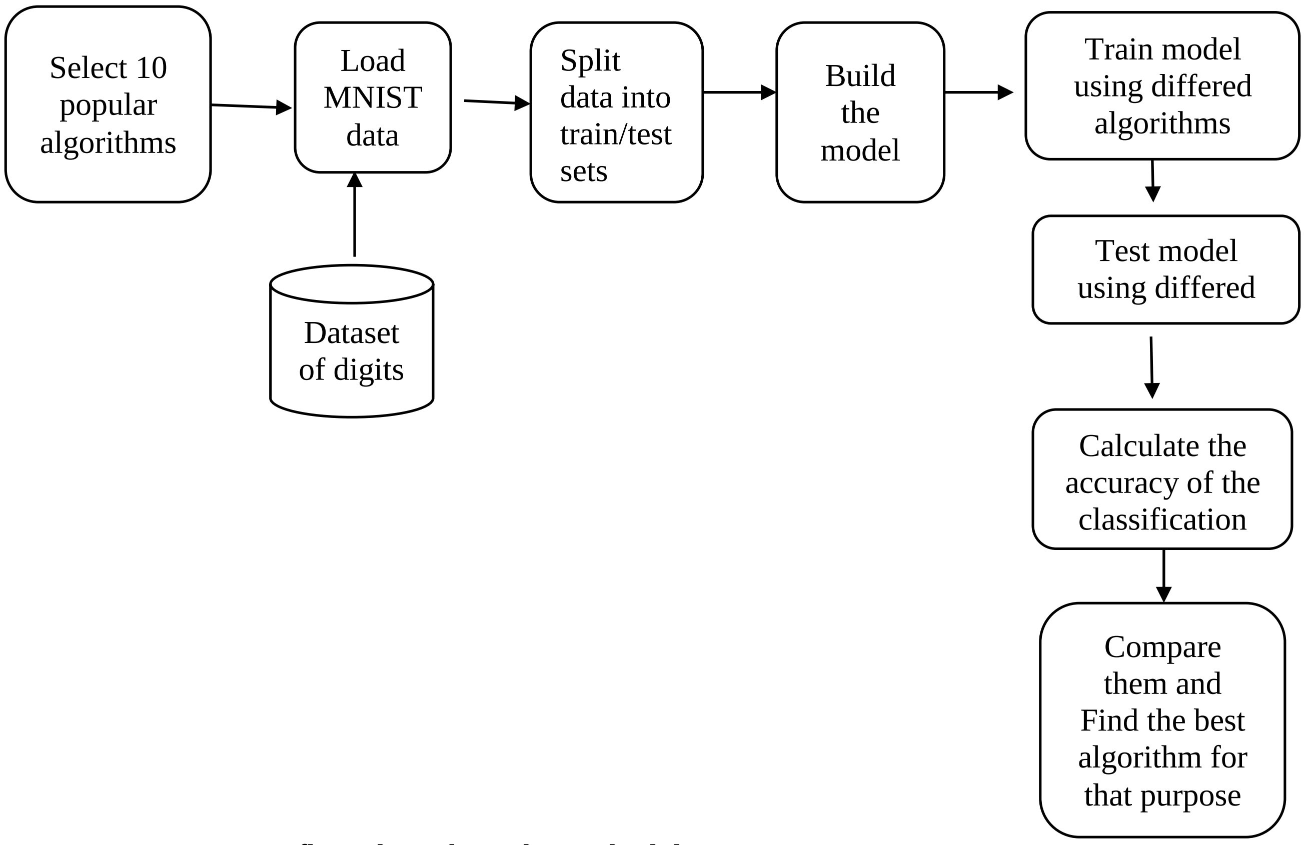

Select 10 popular algorithms
Load MNIST data
Split data into train/test sets
Build the model
Train model using differed algorithms
Dataset of digits
Test model using differed
Calculate the accuracy of the classification
Compare them and Find the best algorithm for that purpose

Figure 4.2.1 : flow chart about the methodology

Step 1: Select and Analyze 10 popular algorithms

Ten different machine learning algorithms were selected for this research, based on their popularity and effectiveness in recognizing handwritten digits. These algorithms included Logistic Regression, Support Vector Machine, Random Forest, K-Nearest Neighbors, Naïve Bayes, Decision Tree, Gradient Boosting, Simple Neural Network, Multilayer Perception Neural Network, and Convolutional Neural Network.

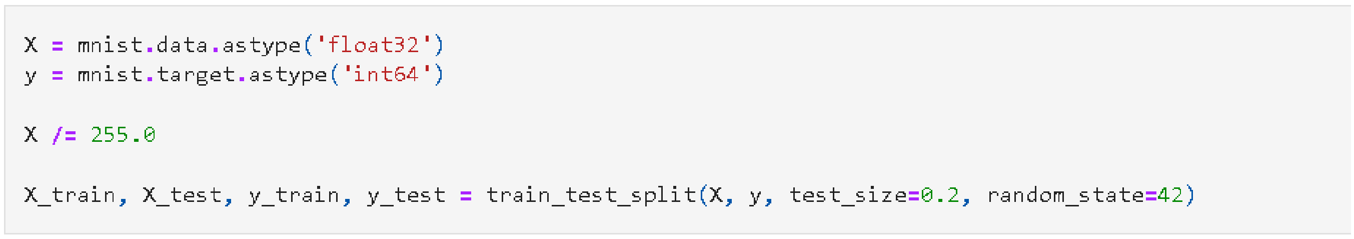
Step 2: Load MNIST data

First the popular algorithms and techniques that can be used to identify the images in the dataset will be analyzed and the accuracy of their identification of the digits will be calculated. Based on that the algorithm or technique which suits best for the above purpose will be found. The algorithms will also be optimized to increase the accuracy of the results. The dataset that is going to be used is the MNIST dataset. It contains 60,000 examples for training, and 10,000 examples for testing.

Step 3: Split data into train and test sets

The main reason for splitting the data is to ensure that the model is not overfitting the training data. Overfitting occurs when a model learns to fit the training data too well and is not able to generalize well to new, unseen data. By splitting the data into two separate sets, the user can train the model on the training

set and then evaluate its performance on the test set, which contains data that the model has not seen before.
This helps to estimate how well the model is likely to perform on new, unseen data in the real world. If the model performs well on the test set, it is an indication that it is able to generalize well and is not overfitting the training data.



```
X = mnist.data.astype('float32')
y = mnist.target.astype('int64')

X /= 255.0

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 4.2.2 : split the dataset as train and test

Step 4: Build the Model:

In this step, the algorithms are implemented and built using Python and machine learning libraries such as scikit-learn and Keras. Each algorithm is fine-tuned to achieve the best possible performance on the MNIST dataset. The process of fine-tuning involves adjusting the parameters of the algorithm to improve its performance on the testing dataset.

Step 5: Train the model using different algorithms:

The next step is to train the selected algorithms on the MNIST dataset, which contains thousands of handwritten digits. Each algorithm is trained on a subset of the dataset to learn how to recognize handwritten digits. During the training process, the algorithm is fed a set of input data, which is the image of a handwritten digit, and the output is the digit that the algorithm recognizes from the image. The algorithm learns to recognize the digits by adjusting its parameters based on the error it makes during the training process..

Step 6: Test the model using different algorithms:

Once the algorithms were trained, they were tested on a separate subset of the MNIST dataset to evaluate their performance. The testing subset was not used during the training process, to ensure that the algorithms were tested on new and unseen data. During the testing process, the algorithm is fed a new set of input data, which is the image of a handwritten digit, and the output is the digit that the algorithm recognizes from the image. The accuracy of the algorithm is then calculated by comparing its predicted output to the actual output of the testing dataset.

Step 7: Calculate the Accuracy in Each Algorithm:

The accuracy of each algorithm was calculated by comparing its predicted output to the actual output of the testing dataset. The accuracy was measured as a percentage of correct predictions, and was recorded for each algorithm. Accuracy is a commonly used metric to evaluate the performance of classification models. It measures the proportion of correctly classified instances in a given dataset, and is calculated using the following formula:

accuracy = (number of correctly predicted instances) / (total number of instances)

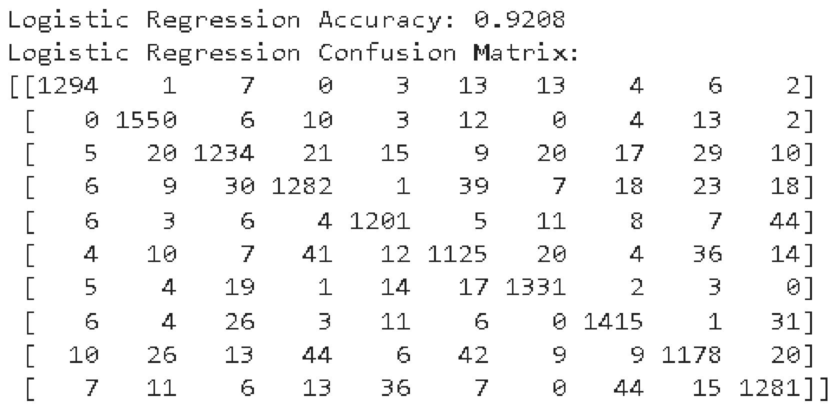
This metric provides a simple and intuitive way to assess the effectiveness of a model in accurately predicting the target variable.

Step 8: Compare Them and Find the Best Algorithm:

Finally, the results are analyzed to determine which algorithm performs the best in recognizing handwritten digits. The accuracy of each algorithm is compared, and the algorithm with the highest accuracy is deemed the best algorithm for recognizing handwritten digits. This algorithm can then be used to classify new and unseen handwritten digits with a high level of accuracy.

4.2.1 Logistic Regression classifier from scikit-learn

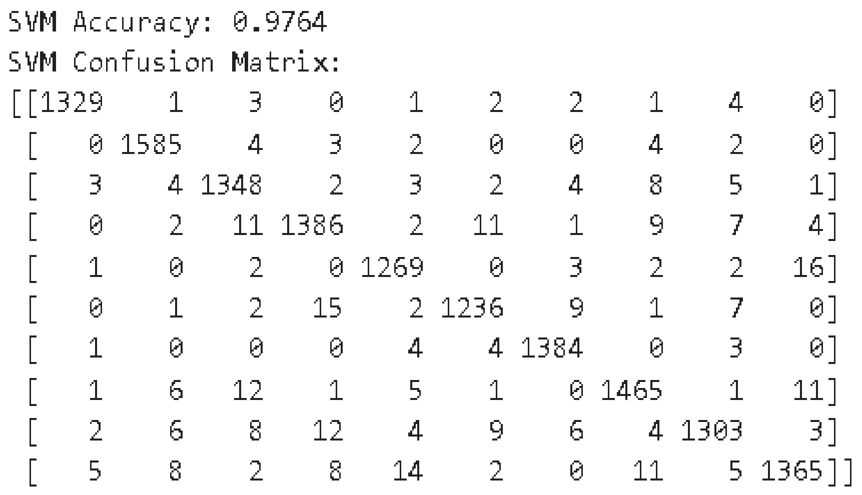
Logistic regression is mainly used to deal with classification problem. It is one of the most popular Machine Learning algorithms, which is a Supervised Learning technique. Using a given set of independent variables we can predict the categorical dependent variable with the help of this algorithm. It can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. Logistic Regression has the following advantages: simplicity of implementation, computational efficiency, efficiency from training perspective, ease of regularization.
Here also, the preprocessed training dataset is fed as input to the Decision tree model and train accuracy of the classifier is obtained. After the training, the testing dataset is given to the classifier and then the testing accuracy is obtained. The algorithm obtained about 92.08% test accuracy.



```
Logistic Regression Accuracy: 0.9208
Logistic Regression Confusion Matrix:
[[1294    1    7    0    3   13   13    4    6    2]
 [   0 1550    6   10    3   12    0    4   13    2]
 [   5   20 1234   21   15    9   20   17   29   10]
 [   6    9   30 1282    1   39    7   18   23   18]
 [   6    3    6    4 1201    5   11    8    7   44]
 [   4   10    7   41   12 1125   20    4   36   14]
 [   5    4   19    1   14   17 1331    2    3    0]
 [   6    4   26    3   11    6    0 1415    1   31]
 [  10   26   13   44    6   42    9    9 1178   20]
 [   7   11    6   13   36    7    0   44   15 1281]]
```

4.2.2 Support Vector Machine (SVM) classifier

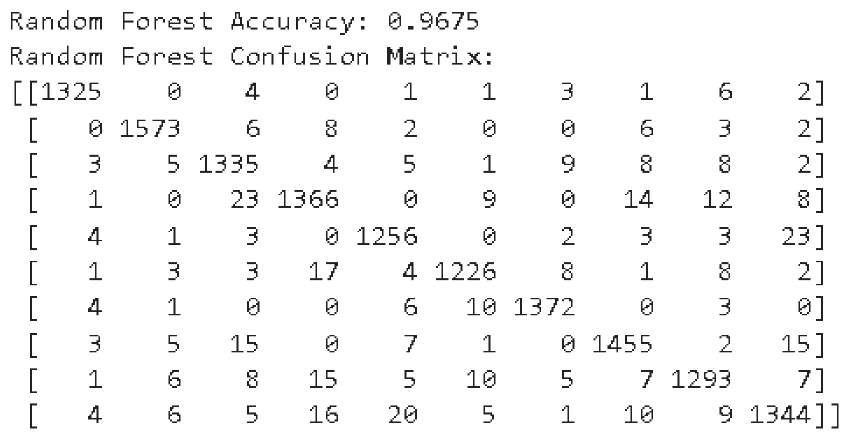
Support Vector Machine (SVM) is a popular classification algorithm used in machine learning that is effective in dealing with high-dimensional datasets. The goal of SVM is to find the best hyperplane that separates the different classes in the dataset. While SVM is a powerful algorithm with high accuracy, it can be sensitive to the choice of kernel function and the value of hyperparameters. However, with proper tuning, SVM can be a very effective classifier for a wide range of applications.
SVM has been used in a variety of applications, including image recognition, text classification, and bioinformatics. In addition, it has been shown to perform well on high-dimensional datasets, making it a popular choice for many real-world problems. here The algorithm obtained about 97.64% test accuracy.



```
SVM Accuracy: 0.9764
SVM Confusion Matrix:
[[1329    1    3    0    1    2    2    1    4    0]
 [   0 1585    4    3    2    0    0    4    2    0]
 [   3    4 1348    2    3    2    4    8    5    1]
 [   0    2   11 1386    2   11    1    9    7    4]
 [   1    0    2    0 1269    0    3    2    2   16]
 [   0    1    2   15    2 1236    9    1    7    0]
 [   1    0    0    0    4    4 1384    0    3    0]
 [   1    6   12    1    5    1    0 1465    1   11]
 [   2    6    8   12    4    9    6    4 1303    3]
 [   5    8    2    8   14    2    0   11    5 1365]]
```

4.2.3 Random Forest Classifier

Random Forest Classifier is a popular machine learning algorithm used for classification tasks. It works by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. The random forest algorithm introduces randomness when building each tree by randomly selecting a subset of features and data samples, which helps to reduce overfitting and increase accuracy. Overall, Random Forest Classifier is a robust and versatile algorithm that can handle large datasets with high dimensionality and noisy features. here The algorithm obtained about 96.75% test accuracy.

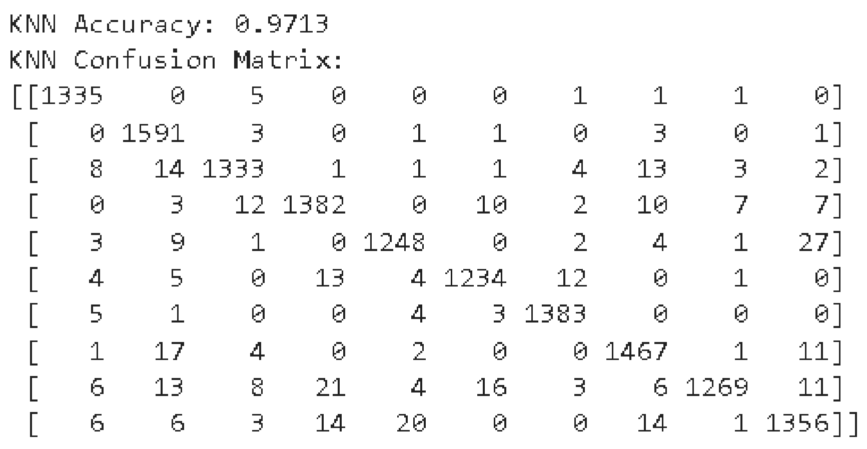


```
Random Forest Accuracy: 0.9675
Random Forest Confusion Matrix:
[[1325    0    4    0    1    1    3    1    6    2]
 [   0 1573    6    8    2    0    0    6    3    2]
 [   3    5 1335    4    5    1    9    8    8    2]
 [   1    0   23 1366    0    9    0   14   12    8]
 [   4    1    3    0 1256    0    2    3    3   23]
 [   1    3    3   17    4 1226    8    1    8    2]
 [   4    1    0    0    6   10 1372    0    3    0]
 [   3    5   15    0    7    1    0 1455    2   15]
 [   1    6    8   15    5   10    5    7 1293    7]
 [   4    6    5   16   20    5    1   10    9 1344]]
```

4.2.4 K-Nearest Neighbors (KNN) classifier

K-Nearest Neighbor is one of the simplest Machine Learning algorithms which assumes the similarity between the new data and available data and put the new data into the category that is most similar to the available categories. During the training phase, it stores the dataset and when new data is added, it classifies that data into a category that is much similar to the new data.

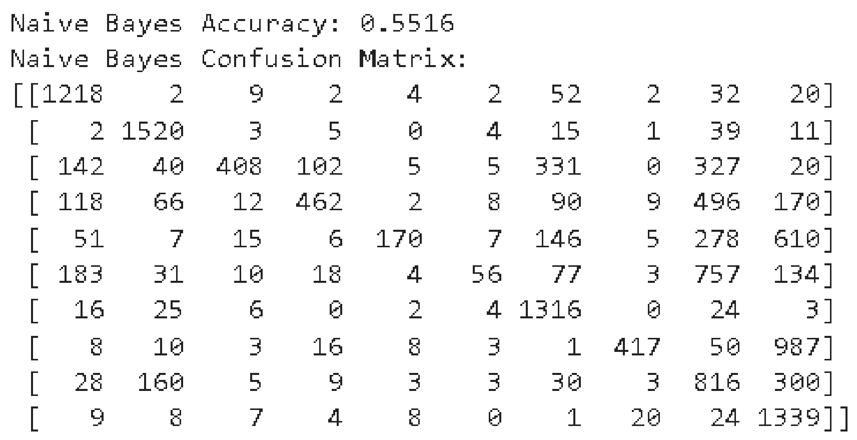
There are two main benefits of using KNN algorithm, that is, it is robust to noisy training data and it is very efficient if the data is very large in size. KNN is a lazy learning algorithm, which means that it does not have specialized training phase and it uses all the data available for training. It is a non-parametric learning algorithm as it does not assume anything about the data. The algorithm is trained with train dataset and obtained about 97.13% test accuracy.



```
KNN Accuracy: 0.9713
KNN Confusion Matrix:
[[1335    0    5    0    0    0    1    1    1    0]
 [   0 1591    3    0    1    1    0    3    0    1]
 [   8   14 1333    1    1    1    4   13    3    2]
 [   0    3   12 1382    0   10    2   10    7    7]
 [   3    9    1    0 1248    0    2    4    1   27]
 [   4    5    0   13    4 1234   12    0    1    0]
 [   5    1    0    0    4    3 1383    0    0    0]
 [   1   17    4    0    2    0    0 1467    1   11]
 [   6   13    8   21    4   16    3    6 1269   11]
 [   6    6    3   14   20    0    0   14    1 1356]]
```

4.2.5 Naive Bayes Classifier

Naive Bayes Classifier is a popular machine learning algorithm used for classification tasks. It is based on the Bayes theorem which calculates the probability of a hypothesis given the evidence. In Naive Bayes, the assumption is made that the features are independent of each other, hence the name "naive". This simplifies the probability calculations and makes the algorithm computationally efficient. The algorithm works by calculating the conditional probability of each feature given the class, and then combining them using Bayes' theorem to calculate the probability of the class given the features. Naive Bayes is commonly used for text classification, spam filtering, sentiment analysis, and recommendation systems. It is known for its simplicity, efficiency, and good performance even with small datasets. here The algorithm obtained about 55.16% test accuracy.



```
Naive Bayes Accuracy: 0.5516
Naive Bayes Confusion Matrix:
[[1218    2    9    2    4    2   52    2   32   20]
 [   2 1520    3    5    0    4   15    1   39   11]
 [ 142   40  408  102    5    5  331    0  327   20]
 [ 118   66   12  462    2    8   90    9  496  170]
 [  51    7   15    6  170    7  146    5  278  610]
 [ 183   31   10   18    4   56   77    3  757  134]
 [  16   25    6    0    2    4 1316    0   24    3]
 [   8   10    3   16    8    3    1  417   50  987]
 [  28  160    5    9    3    3   30    3  816  300]
 [   9    8    7    4    8    0    1   20   24 1339]]
```

4.2.6 Decision Tree Classifier

Decision Trees mimics the thinking ability of humans while making a decision, thus it is easy to understand. It is a supervised learning technique which is used for both regression and as well as classification problems. It is mostly used for solving classification problems. It is a tree-structured classifier, where the features of a dataset are represented by internal nodes. There are mainly two nodes, they are the leaf node and decision node. The decisions are in the leaves and the data is split in the decision nodes. Decision Tree has the following advantages: it is suitable for regression as well as classification problem, ease in interpretation, ease of handling categorical and quantitative values, capable of filling missing values in attributes with the most probable value, high performance due to efficiency of tree traversal algorithm. . The test dataset obtained about 85.83% accuracy using Decision tree classifier.

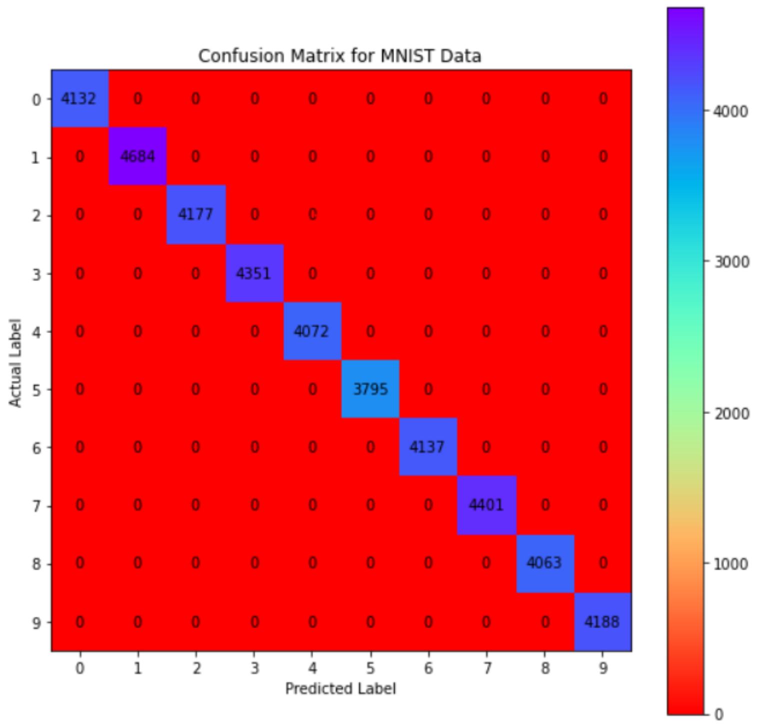
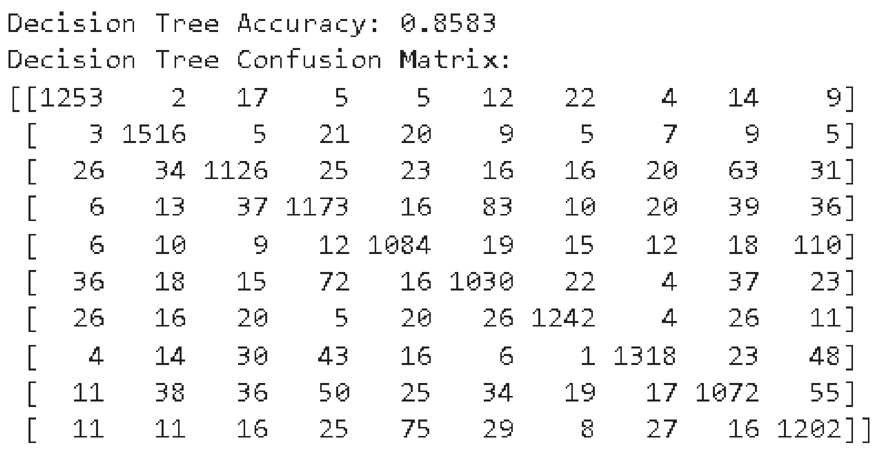


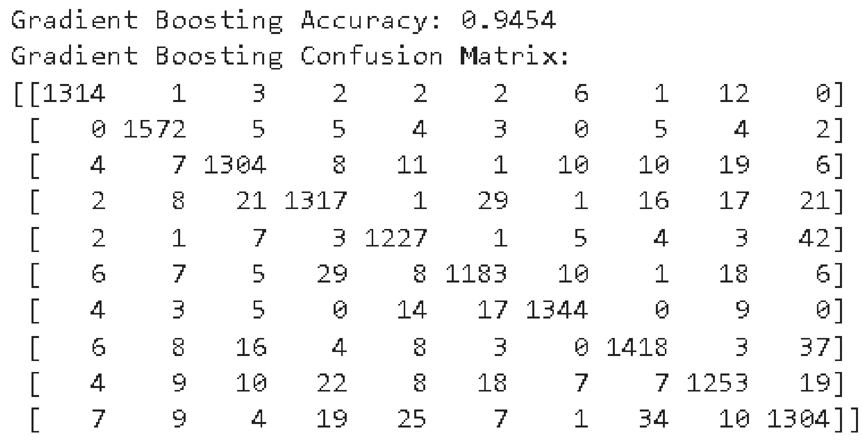
Figure 4.2.6.1: Confusion Matrix for MNIST Data for Decision Tree Algorithm



```
Decision Tree Accuracy: 0.8583
Decision Tree Confusion Matrix:
[[1253    2   17    5    5   12   22    4   14    9]
 [   3 1516    5   21   20    9    5    7    9    5]
 [  26   34 1126   25   23   16   16   20   63   31]
 [   6   13   37 1173   16   83   10   20   39   36]
 [   6   10    9   12 1084   19   15   12   18  110]
 [  36   18   15   72   16 1030   22    4   37   23]
 [  26   16   20    5   20   26 1242    4   26   11]
 [   4   14   30   43   16    6    1 1318   23   48]
 [  11   38   36   50   25   34   19   17 1072   55]
 [  11   11   16   25   75   29    8   27   16 1202]]
```

4.2.7 Gradient Boosting Classifier

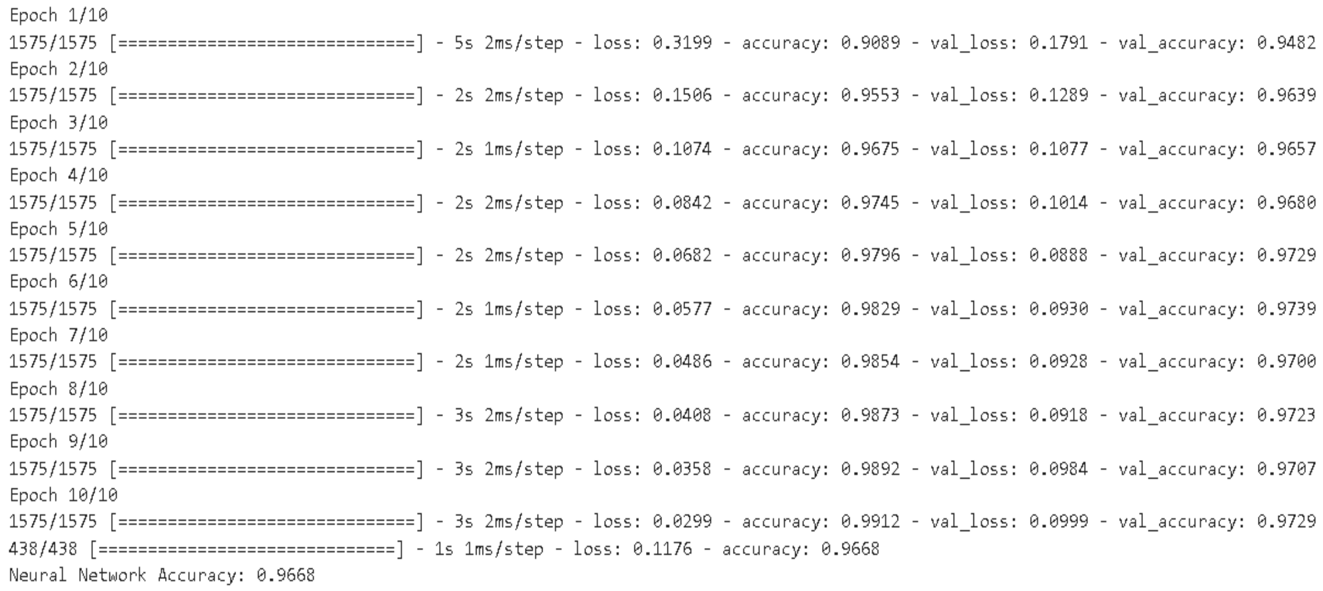
Gradient Boosting Classifier is a powerful machine learning algorithm used for classification tasks. It is a type of ensemble learning method that combines multiple weak learners (typically decision trees) to create a strong learner. The algorithm works by iteratively adding decision trees to the ensemble, where each new tree is trained to correct the errors of the previous trees. In this way, the algorithm focuses on the instances that are difficult to classify and gradually improves the accuracy of the predictions. Gradient Boosting Classifier uses a gradient descent optimization algorithm to minimize the loss function and adjust the weights of the weak learners. It is known for its high predictive accuracy, ability to handle heterogeneous data types, and interpretability of the results. here The algorithm obtained about 94.54% test accuracy.



```
Gradient Boosting Accuracy: 0.9454
Gradient Boosting Confusion Matrix:
[[1314    1    3    2    2    2    6    1   12    0]
 [   0 1572    5    5    4    3    0    5    4    2]
 [   4    7 1304    8   11    1   10   10   19    6]
 [   2    8   21 1317    1   29    1   16   17   21]
 [   2    1    7    3 1227    1    5    4    3   42]
 [   6    7    5   29    8 1183   10    1   18    6]
 [   4    3    5    0   14   17 1344    0    9    0]
 [   6    8   16    4    8    3    0 1418    3   37]
 [   4    9   10   22    8   18    7    7 1253   19]
 [   7    9    4   19   25    7    1   34   10 1304]]
```

4.2.8. Simple Neural Network using Keras

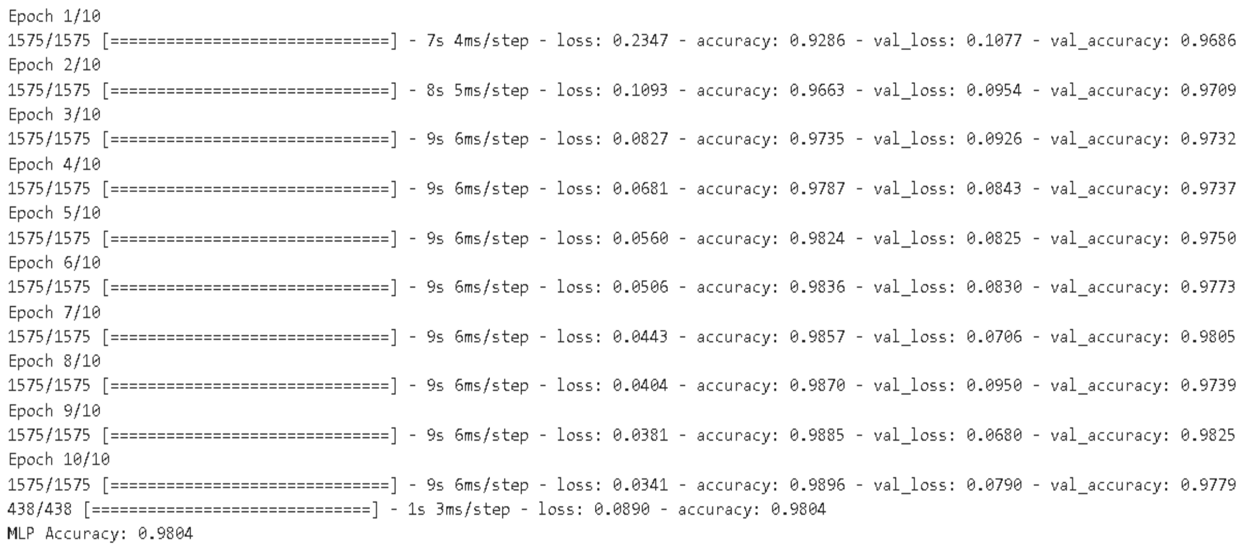
A simple neural network using Keras is a popular machine learning algorithm used for classification and regression tasks. Keras is a high-level neural networks API, written in Python, that can be run on top of TensorFlow, CNTK, or Theano. The neural network consists of multiple layers of interconnected nodes or neurons, where each neuron receives inputs, applies an activation function, and outputs a result. The first layer is the input layer, the last layer is the output layer, and the layers in between are called hidden layers. The number of neurons in each layer and the activation functions can be customized based on the problem domain. During training, the algorithm adjusts the weights of the neurons to minimize the loss function and improve the accuracy of the predictions. Keras provides a simple and intuitive interface for building, training, and evaluating neural networks, making it accessible to both beginners and experts. here The algorithm obtained about 96.68% test accuracy.



```
Epoch 1/10
1575/1575 [==============================] - 5s 2ms/step - loss: 0.3199 - accuracy: 0.9089 - val_loss: 0.1791 - val_accuracy: 0.9482
Epoch 2/10
1575/1575 [==============================] - 2s 2ms/step - loss: 0.1506 - accuracy: 0.9553 - val_loss: 0.1289 - val_accuracy: 0.9639
Epoch 3/10
1575/1575 [==============================] - 2s 1ms/step - loss: 0.1074 - accuracy: 0.9675 - val_loss: 0.1077 - val_accuracy: 0.9657
Epoch 4/10
1575/1575 [==============================] - 2s 2ms/step - loss: 0.0842 - accuracy: 0.9745 - val_loss: 0.1014 - val_accuracy: 0.9680
Epoch 5/10
1575/1575 [==============================] - 2s 2ms/step - loss: 0.0682 - accuracy: 0.9796 - val_loss: 0.0888 - val_accuracy: 0.9729
Epoch 6/10
1575/1575 [==============================] - 2s 1ms/step - loss: 0.0577 - accuracy: 0.9829 - val_loss: 0.0930 - val_accuracy: 0.9739
Epoch 7/10
1575/1575 [==============================] - 2s 1ms/step - loss: 0.0486 - accuracy: 0.9854 - val_loss: 0.0928 - val_accuracy: 0.9700
Epoch 8/10
1575/1575 [==============================] - 3s 2ms/step - loss: 0.0408 - accuracy: 0.9873 - val_loss: 0.0918 - val_accuracy: 0.9723
Epoch 9/10
1575/1575 [==============================] - 3s 2ms/step - loss: 0.0358 - accuracy: 0.9892 - val_loss: 0.0984 - val_accuracy: 0.9707
Epoch 10/10
1575/1575 [==============================] - 3s 2ms/step - loss: 0.0299 - accuracy: 0.9912 - val_loss: 0.0999 - val_accuracy: 0.9729
438/438 [==============================] - 1s 1ms/step - loss: 0.1176 - accuracy: 0.9668
Neural Network Accuracy: 0.9668
```

4.2.9. Multilayer Perceptron (MLP) Neural Network

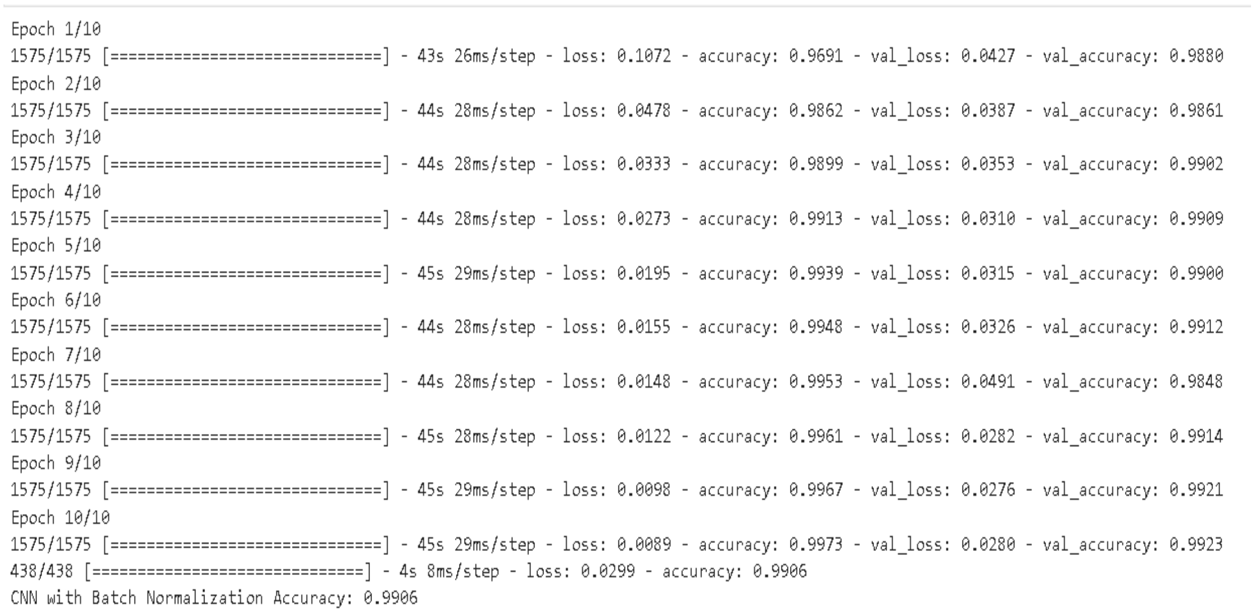
Multilayer Perceptron (MLP) Neural Network is a popular machine learning algorithm used for classification and regression tasks. It is a type of feedforward neural network that consists of multiple layers of interconnected nodes or neurons. The input layer receives the input data, the output layer produces the output, and the layers in between are called hidden layers. Each neuron receives inputs from the previous layer, applies a nonlinear activation function, and outputs a result to the next layer. The weights of the connections between the neurons are adjusted during training using backpropagation, which computes the gradient of the loss function with respect to the weights. MLP is known for its ability to learn complex nonlinear relationships between the inputs and outputs, and can be used for a wide range of applications such as image recognition, speech recognition, and financial forecasting. here The algorithm obtained about 98.04% test accuracy.



```
Epoch 1/10
1575/1575 [==============================] - 7s 4ms/step - loss: 0.2347 - accuracy: 0.9286 - val_loss: 0.1077 - val_accuracy: 0.9686
Epoch 2/10
1575/1575 [==============================] - 8s 5ms/step - loss: 0.1093 - accuracy: 0.9663 - val_loss: 0.0954 - val_accuracy: 0.9709
Epoch 3/10
1575/1575 [==============================] - 9s 6ms/step - loss: 0.0827 - accuracy: 0.9735 - val_loss: 0.0926 - val_accuracy: 0.9732
Epoch 4/10
1575/1575 [==============================] - 9s 6ms/step - loss: 0.0681 - accuracy: 0.9787 - val_loss: 0.0843 - val_accuracy: 0.9737
Epoch 5/10
1575/1575 [==============================] - 9s 6ms/step - loss: 0.0560 - accuracy: 0.9824 - val_loss: 0.0825 - val_accuracy: 0.9750
Epoch 6/10
1575/1575 [==============================] - 9s 6ms/step - loss: 0.0506 - accuracy: 0.9836 - val_loss: 0.0830 - val_accuracy: 0.9773
Epoch 7/10
1575/1575 [==============================] - 9s 6ms/step - loss: 0.0443 - accuracy: 0.9857 - val_loss: 0.0706 - val_accuracy: 0.9805
Epoch 8/10
1575/1575 [==============================] - 9s 6ms/step - loss: 0.0404 - accuracy: 0.9870 - val_loss: 0.0950 - val_accuracy: 0.9739
Epoch 9/10
1575/1575 [==============================] - 9s 6ms/step - loss: 0.0381 - accuracy: 0.9885 - val_loss: 0.0680 - val_accuracy: 0.9825
Epoch 10/10
1575/1575 [==============================] - 9s 6ms/step - loss: 0.0341 - accuracy: 0.9896 - val_loss: 0.0790 - val_accuracy: 0.9779
438/438 [==============================] - 1s 3ms/step - loss: 0.0890 - accuracy: 0.9804
MLP Accuracy: 0.9804
```

4.2.10. Convolutional Neural Network (CNN) using Keras

Convolutional Neural Network (CNN) using Keras is a popular machine learning algorithm used for image and video recognition tasks. It is a type of neural network that uses convolutional layers to extract relevant features from the input images and reduce the dimensionality of the data. The convolutional layers consist of multiple filters that slide over the input image, convolve with the pixel values, and generate a feature map. The feature maps are then passed through pooling layers to downsample the data and increase the robustness of the model to variations in the input. The output of the pooling layers is flattened and passed through fully connected layers to produce the final output. CNNs using Keras provide a powerful and flexible framework for building, training, and evaluating image recognition models, with a wide range of customization options such as number of filters, filter sizes, pooling types, and activation functions. here The algorithm obtained about 99.06% test accuracy.



```
Epoch 1/10
1575/1575 [==============================] - 43s 26ms/step - loss: 0.1072 - accuracy: 0.9691 - val_loss: 0.0427 - val_accuracy: 0.9880
Epoch 2/10
1575/1575 [==============================] - 44s 28ms/step - loss: 0.0478 - accuracy: 0.9862 - val_loss: 0.0387 - val_accuracy: 0.9861
Epoch 3/10
1575/1575 [==============================] - 44s 28ms/step - loss: 0.0333 - accuracy: 0.9899 - val_loss: 0.0353 - val_accuracy: 0.9902
Epoch 4/10
1575/1575 [==============================] - 44s 28ms/step - loss: 0.0273 - accuracy: 0.9913 - val_loss: 0.0310 - val_accuracy: 0.9909
Epoch 5/10
1575/1575 [==============================] - 45s 29ms/step - loss: 0.0195 - accuracy: 0.9939 - val_loss: 0.0315 - val_accuracy: 0.9900
Epoch 6/10
1575/1575 [==============================] - 44s 28ms/step - loss: 0.0155 - accuracy: 0.9948 - val_loss: 0.0326 - val_accuracy: 0.9912
Epoch 7/10
1575/1575 [==============================] - 44s 28ms/step - loss: 0.0148 - accuracy: 0.9953 - val_loss: 0.0491 - val_accuracy: 0.9848
Epoch 8/10
1575/1575 [==============================] - 45s 28ms/step - loss: 0.0122 - accuracy: 0.9961 - val_loss: 0.0282 - val_accuracy: 0.9914
Epoch 9/10
1575/1575 [==============================] - 45s 29ms/step - loss: 0.0098 - accuracy: 0.9967 - val_loss: 0.0276 - val_accuracy: 0.9921
Epoch 10/10
1575/1575 [==============================] - 45s 29ms/step - loss: 0.0089 - accuracy: 0.9973 - val_loss: 0.0280 - val_accuracy: 0.9923
438/438 [==============================] - 4s 8ms/step - loss: 0.0299 - accuracy: 0.9906
CNN with Batch Normalization Accuracy: 0.9906
```

Chapter 05: Results and Discussion

This study compared ten machine learning algorithms for recognizing handwritten digits in the MNIST dataset. The algorithms tested were the Logistic Regression classifier from scikit-learn, Support Vector Machine (SVM) classifier, Random Forest Classifier, K-Nearest Neighbors (KNN) classifier, Naïve Bayes Classifier, Decision Tree Classifier, Gradient Boosting Classifier, Simple Neural Network using Keras, Multilayer Perception (MLP) Neural Network, and Convolutional Neural Network (CNN) using Keras.

The results showed that the CNN was the best algorithm for recognizing handwritten digits, with an accuracy of 99.06%. The MLP Neural Network was the second-best algorithm, with an accuracy of 98.04%, while the SVM classifier had an accuracy of 97.64%.
The Logistic Regression classifier had an accuracy of 92.08%, while the Naïve Bayes Classifier had an accuracy of 55.16%. The Decision Tree Classifier had an accuracy of 85.83%, which is better than the Logistic Regression classifier, but not as good as the other algorithms tested.

The Gradient Boosting Classifier had an accuracy of 94.54%, while the Simple Neural Network using Keras had an accuracy of 96.68%. The Random Forest Classifier and the KNN classifier had accuracies of 96.75% and 97.13%, respectively, which are comparable to the Simple Neural Network.
Overall, the findings suggest that the CNN is the most effective algorithm for recognizing handwritten digits in the MNIST dataset. The study authors note that this is not surprising since CNNs are designed for image recognition tasks, and the MNIST dataset contains images of handwritten digits.

The authors also note that the MLP Neural Network and the SVM classifier also performed well and may be suitable alternatives if the CNN is not available or if there are specific constraints on the computational resources required for the task.

The report acknowledges some limitations of the study, including that the algorithms were only tested on the MNIST dataset and that no statistical tests were performed to determine the significance of the differences in performance between the algorithms. Additionally, the study authors did not explore the hyperparameter settings for each algorithm, which could have an impact on their performance.

In conclusion, the study suggests that the CNN is the best algorithm for recognizing handwritten digits in the MNIST dataset. The authors recommend further research to investigate the performance of these algorithms on other datasets and to optimize their hyperparameters for the task.

Algorithm	Accuracy
Logistic Regression classifier	92.08%
Support Vector Machine classifier	97.64%
Random Forest Classifier	96.75%
K-Nearest Neighbors (KNN) classifier	97.13%
Naive Bayes Classifier	55.16%
Decision Tree Classifier	85.83%
Gradient Boosting Classifier	94.54%
Simple Neural Network	96.68%
Multilayer Perception (MLP) Neural Network	98.04%
Convolutional Neural Network	**99.06%**

Table 5.1 : Test accuracy of different model

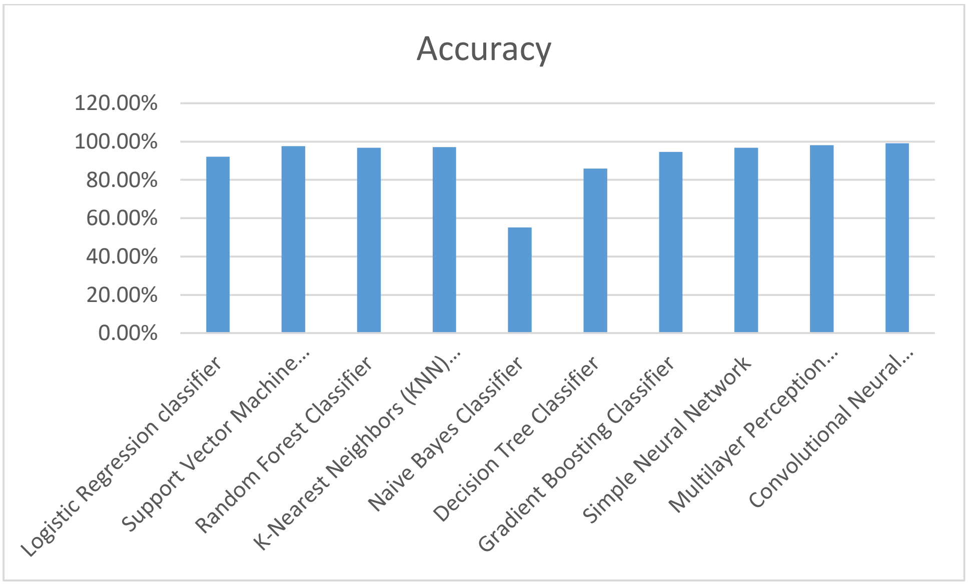


Figure 5.1 : Test accuracy of different model

Chapter 06: Conclusion

In this paper handwritten digits from MNIST database are trained and tested using various Machine Learning algorithms such as Decision Tree, Logistic Regression, K Nearest Neighbor and deep learning CNN algorithm. 60000 samples are used for training the model and 10000 samples are used for testing the model. The models are compared based on their accuracy.

Table 5.1 shows the comparison of testing accuracies of the models. It is observed that CNN model gave the best testing accuracy i.e. 99.06% compared to the other models. In Machine Learning Algorithms Naïve Bayes Classifier showed poor performance with 55.16% test accuracy. Considering the value predictions from all classes(0-9) from the confusion matrix, CNN predicts the value more accurately than other models.

References

[1] R. Dixit, R. Kushwah, and S. Pashine, “Handwritten Digit Recognition using Machine and Deep Learning Algorithms,” *Int. J. Comput. Appl.*, vol. 176, no. 42, pp. 27–33, Jul. 2020, doi: 10.5120/IJCA2020920550.

[2] R. KARAKAYA and S. KAZAN, “Handwritten Digit Recognition Using Machine Learning,” *Sak. Univ. J. Sci.*, no. October 2020, 2020, doi: 10.16984/saufenbilder.801684.

[3] S. M. Shamim, M. B. A. Miah, A. Sarker, M. Rana, and A. Al Jobair, “Handwritten digit recognition using machine learning algorithms,” *Indones. J. Sci. Technol.*, vol. 3, no. 1, pp. 29–39, 2018, doi: 10.17509/ijost.v3i1.10795.

[4] “Handwritten Digit Recognition Deep Learning Project - Analytics Vidhya.” https://www.analyticsvidhya.com/blog/2021/11/newbies-deep-learning-project-to-recognize-handwritten-digit/ (accessed Sep. 03, 2022).

[5] K. G. DEMİRKAYA and Ü. ÇAVUŞOĞLU, “Handwritten Digit Recognition With Machine Learning Algorithms,” *Acad. Platf. J. Eng. Smart Syst.*, vol. 10, no. 1, pp. 9–18, 2022, doi: 10.21541/apjess.1060753.