HANDWRITTEN DIGIT RECOGNITION

Minor project report submitted in partial fulfilment of the requirement for the degree of Bachelor of Technology

in

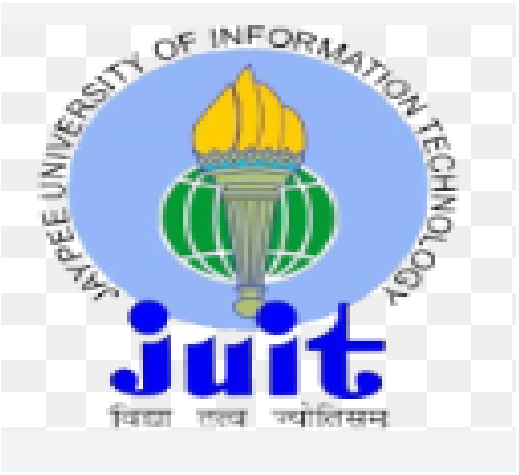
Computer Science and Engineering

By

ROHIT MISHRA (201465)

UNDER THE SUPERVISION OF

Dr HARI SINGH RAWAT



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, 173234, Himachal Pradesh, INDIA

TABLE OF CONTENT

DECLARATION

We hereby declare that this project has been done by me under the supervision of Dr Hari Singh Rawat**,** Jaypee University of Information Technology. We also declare that neither this project nor any part of this project has been submitted elsewhere for the award of any degree or diploma.

Supervised by:

(Dr. Hari Singh Rawat)

Designation - Assistant Professor (SG)

Department of Computer Science & Engineering

Jaypee University of Information Technology

Submitted by:

Chirag Jain - (201267)

Rohit Mishra-(201465)

Sachin Mishra-(201466)

Computer Science & Engineering Department

Jaypee University of Information Technology

CERTIFICATE

This is to certify that the work which is being presented in the project report titled “Handwritten Digit Recognition” in partial fulfilment of the requirements for the award of the degree of B.Tech in Computer Science And Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by “Chirag Jain(201267), Rohit Mishra (201465), Sachin Mishra(201466).” during the period from January 2023 to May 2023 under the supervision of Dr Hari Singh, Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

Chirag Jain - (201267)
Rohit Mishra-(201465)
Sachin Mishra-(201466)

The above statement made is correct to the best of my knowledge.

Dr Hari Singh Rawat
Designation - Assistant Professor (SG)
Computer Science & Engineering and Information Technology
Jaypee University of Information Technology, Waknaghat,

ACKNOWLEDGEMENT

Firstly, We express our heartiest thanks and gratefulness to almighty God for His divine blessing making it possible for us to complete the project work successfully.

We are grateful and wish my profound indebtedness to Supervisor **Dr Hari Singh**, Department of CSE Jaypee University of Information Technology, Wakhnaghat. Deep Knowledge & keen interest of my supervisor in the field of "**Research Area**" to carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, and reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

We would like to express my heartiest gratitude to **Dr Hari Singh,** Department of CSE, for his kind help to finish my project.

We would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, we would like to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, We must acknowledge with due respect the constant support and patients of our parents.

Chirag Jain - (201267)
Rohit Mishra-(201465)
Sachin Mishra-(201466)

ABSTRACT

Handwritten digit recognition is a popular task in the field of computer vision and machine learning. The goal is to correctly identify digits from handwritten images. This problem is challenging due to the variability in writing styles and the complexity of digit shapes. Various techniques have been developed to address this problem, including neural networks, decision trees, and support vector machines. Deep learning methods, such as convolutional neural networks, have achieved state-of-the-art results on this task. These methods involve training a model on a large dataset of labelled images, which allows it to learn the features that are most relevant for classification. Handwritten digit recognition has numerous practical applications, including recognizing zip codes on mail, reading bank checks, and recognizing handwritten signatures.

In recent years, deep learning methods, particularly convolutional neural networks (CNNs), have been widely used in digit recognition tasks and have achieved state-of-the-art performance. CNNs are neural networks that are specifically designed to process grid-like data, such as images. They consist of multiple layers of neurons, including convolutional layers, pooling layers, and fully connected layers. CNNs are capable of automatically learning the relevant features from the input images, which allows them to achieve high accuracy in digit recognition tasks.

The most popular dataset used for training and testing digit recognition models is the MNIST dataset. This dataset contains 60,000 training images and 10,000 test images of handwritten digits. Many researchers and developers have used this dataset to benchmark their models and compare their performance with other models.

Handwritten digit recognition has many practical applications. One of the most common applications is recognizing zip codes on mail, which is used by the postal service to sort mail more efficiently. Another application is reading bank checks, where digit recognition is used to extract the amount of money and other information from the handwritten text on the check. Handwritten digit recognition can also be used in recognizing handwritten signatures, which is important in fields such as banking, legal, and governmental sectors.

CHAPTER-1. INTRODUCTION

1.1 INTRODUCTION

Handwritten digit recognition is the task of recognizing the numerical digits from handwritten images. It is a widely studied problem in the field of pattern recognition and computer vision. The ability to recognize handwritten digits has many practical applications, such as recognizing zip codes on mail, reading bank checks, and recognizing handwritten signatures. Accurate recognition of handwritten digits is a challenging problem due to the variability in handwriting styles, the different shapes of digits, and the presence of noise and artefacts in the images.

Over the years, various machine learning techniques have been developed to address this problem, such as decision trees, support vector machines, and deep learning methods. Deep learning, particularly convolutional neural networks (CNNs), has recently emerged as a powerful tool for recognizing handwritten digits, achieving state-of-the-art performance on many benchmark datasets.

The most popular dataset used for training and testing digit recognition models is the MNIST dataset, which contains a large number of labelled images of handwritten digits. Researchers and developers use this dataset to benchmark their models and compare their performance with other models.

In this context, this paper provides an overview of the problem of handwritten digit recognition. We will explore the various techniques used for recognizing handwritten digits, including traditional machine learning methods and deep learning methods. We will also discuss the challenges associated with digit recognition and the practical applications of this technology.

1.2 OBJECTIVE

- The objective of handwritten digit recognition is to develop an accurate and reliable system that can automatically recognize and classify the numerical digits from handwritten images. The goal is to design a system that can correctly identify the digit represented in the image, regardless of variations in the writing style of different individuals, the different shapes of digits, and the presence of noise and artefacts in the images.
- The primary objective of handwritten digit recognition is to improve the efficiency and accuracy of tasks that involve recognizing handwritten digits, such as sorting mail, reading bank checks, and recognizing signatures. Accurate recognition of handwritten digits can significantly reduce errors and improve the speed of these tasks, which can have a significant impact on productivity and cost savings.
- In addition, handwritten digit recognition serves as a benchmark problem for evaluating the performance of various machine learning and deep learning techniques. The development of accurate and efficient handwritten digit recognition systems has important implications for the field of pattern recognition and computer vision, as it can provide valuable insights into the capabilities and limitations of different algorithms and architectures.

1.3 MOTIVATION

The motivation for developing handwritten digit recognition systems stems from the need to automate and improve the accuracy of tasks that involve recognizing handwritten digits. Traditional methods of digit recognition, such as manual sorting of mail, can be time-consuming and error-prone, particularly when dealing with large volumes of data. By automating the process of digit recognition, we can improve efficiency, reduce errors, and increase productivity.

1.4 LANGUAGE USED - PYTHON

- **Python IDLE**

```
import pandas as pd
from sklearn.model_selection import train_test_split
from google.colab import files
from scipy import stats
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from math import sqrt
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
```

Figure-1 Library Used

1.5 Technical Requirements (Hardware)

- Keras: Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. You can install Keras using pip,the Python package installer, by running the following command in your terminal: `pip install keras`.
- TensorFlow: TensorFlow is an open-source machine learning framework developed by Google. It provides a set of tools for building and training machine learning models, including deep neural networks. You can install TensorFlow using pip by running the following command in your terminal: `pip install tensorflow`

1.6 Features of Dataset

The MNIST dataset consists of images of handwritten digits, and each image is a 28x28 grayscale image. Therefore, the total number of features in the MNIST dataset is 28 x 28 = 784. Each pixel in the image is considered as a feature, and the value of each pixel represents the intensity of the grayscale colour, ranging from 0 (black) to 255 (white). Therefore, each image in the dataset is represented by a vector of 784 features, which can be used as input to train a machine learning model for handwritten digit recognition.

1.7 Deliverables/Outcomes

Model building through Machine Learning and Data Mining is a complex task. Considering a large dataset and numerous features, parameters of the training algorithms and testing it takes a while and a lot of effort to build a model. After creating the model we will be able to predict handwritten digit recognition

Chapter 02
Feasibility Study, Research Survey, Requirements Analysis, and Design

2.1 Feasibility Survey-

Handwritten digit recognition would involve assessing the technical and economic viability of developing a system that can accurately recognize handwritten digits**.**

2.2 Research Survey

Handwritten digit recognition would involve reviewing existing literature on the topic to understand the current state-of-the-art techniques and their applications. Here are some key areas that would be included in such a survey:

1. Dataset: Handwritten digit recognition algorithms are typically trained on a dataset of labelled images. The choice of dataset can have a significant impact on the performance of the algorithm. A survey would typically review the most commonly used datasets for this task, such as MNIST, EMNIST, and SVHN.
2. Feature extraction: One of the challenges in handwritten digit recognition is extracting meaningful features from the raw image data. A survey would review the most commonly used techniques for feature extraction, such as HOG, SIFT, and LBP, as well as more recent deep learning approaches like convolutional neural networks (CNNs).
3. Classification techniques: Once features have been extracted, a classifier is used to predict the label of the digit. A survey would review the most commonly used classification techniques for this

task, such as k-Nearest Neighbors, Support Vector Machines, and Random Forests.

4. Deep learning approaches: In recent years, deep learning techniques have emerged as state-of-the-art for handwritten digit recognition. A survey would review the most commonly used deep learning architectures for this task, such as LeNet, AlexNet, and ResNet.
5. Applications: Handwritten digit recognition has a wide range of applications, including in image processing, handwriting recognition, and optical character recognition (OCR). A survey would review the most common applications of this technology and the challenges faced in each application.

   A research survey of handwritten digit recognition would provide an overview of the current state-of-the-art techniques and their applications, as well as insights into the challenges and opportunities for further research in this area.

LITERATURE REVIEW

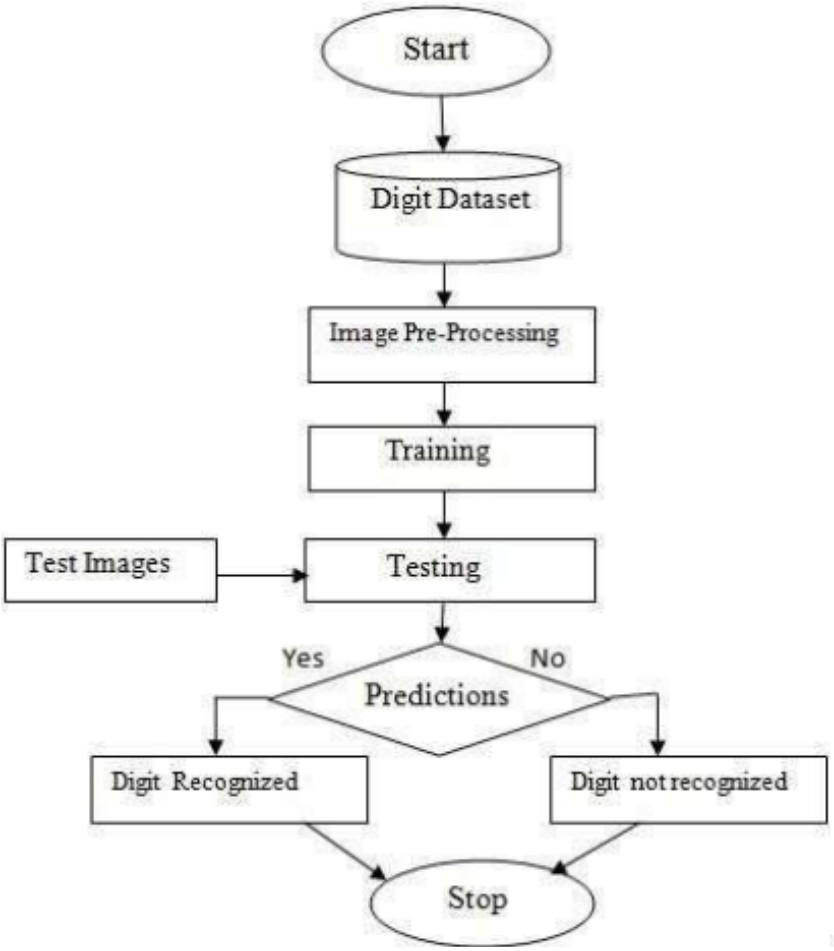
TITLE	AUTHOR	TOOLS	PERFORMANCE
Handwritten Digit recognition using convolutional neutral network	Yann Lecun ,Yoshua Bengio ,Patrik Haffner ,Leon Bottou	Data augmentation and dropout	99.1%
CNN approach for recognizing Handwritten Digits	Amna Khan ,Muhammad Sharif ,Hina Ayesha , Tayyaba Naseem and Attique Dawood	Dropout regulation and batch normalization to prevent over fitting	99%
Handwritten Digit recognition using machine learning	Fathma Siddique1# , Shadman Sakib2* , Md. Abu Bakr Siddique2$	Tensorflow MNIST DATASET CNN	CNN-99.11%
Deep learning to recognize handwritten digits	Akkireddy Challa	Water Reservoir Concept	CNN is 71%, SVM is 39%, and ANN is 37%
Handwritten Digit recognition using convolutional neutral network	S M Shamim, Mohammad Badrul Alam Miah, Angona Sarker, Masud Rana & Abdullah Al Jobair	Recognizing postal codes ,bank check processing and recognizing handwritten numbers	CNN -90%

2.3 Requirements Analysis

Google collaboratory

Libraries like: Tensorflow

2.4 Design



Chapter-3: Implementation

3.1 Machine Learning algorithms

3.1.1 Decision Tree Model:

DT is a supervised learning ML algorithm used for both classification and regression problems. But they are commonly used to solve classification problems . DT is based on a tree-structure classifier in which the root node represents the entire samples or population, and branches represent the rules, internal nodes. The results represent the features of the dataset defined by each leaf node .

```
from sklearn.datasets import load_digits
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

Load the dataset

```
digits = load_digits()
```

Split the data into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.3, random_state=100)
```

Define the decision tree classifier

```
clf = DecisionTreeClassifier(random_state=42)
```

Define the parameter grid

```
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2, 4, 6, 8],
    'min_samples_split': [2, 4, 6, 8],
    'min_samples_leaf': [1, 2, 3, 4]
}
```

Perform grid search using cross-validation

```
grid_search = GridSearchCV(clf, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

Get the best decision tree classifier

```
best_clf = grid_search.best_estimator_
```

Make predictions on the testing set

```
y_pred = best_clf.predict(X_test)
```

Compute performance measures

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')
```

Print the performance measures

```
print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1 Score: {:.2f}".format(f1))
```

- **Performance Measures**

```
Accuracy: 0.86
Precision: 0.87
Recall: 0.86
F1 Score: 0.86
```

3.1.2 Random Forest Tree

RF is the most widely used algorithm that comes under the supervised learning category. This algorithm is based upon the concept of ensemble learning, further classified into multiple classifiers that are combined for efficient predictions. The combination of various classifiers is used to solve complex problems by increasing the model's performance. This algorithm is used for both regression and classification problems. RF consists of many DTs in the form of subsets of the provided dataset and takes an average of the subsets to improvise the accuracy of the dataset.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_digits
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split
```

Load the dataset

```
digits = load_digits()
```

Split the data into training and testing sets

```
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.3, random_state=42)
```

Define the Random forest classifier

```
rfc = RandomForestClassifier()

parameters = {'n_estimators': [50, 250, 350], 'max_depth': [5, 10, 15], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 5]}
```

Define the parameter grid

```
 grid_search = GridSearchCV(rfc, parameters, cv=5)
```

```
grid_search.fit(X_train, y_train)
```

Make predictions on the testing set

```
y_pred = grid_search.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro')
recall = recall_score(y_test, y_pred, average='macro')
f1 = f1_score(y_test, y_pred, average='macro')
```

Print the performance measures

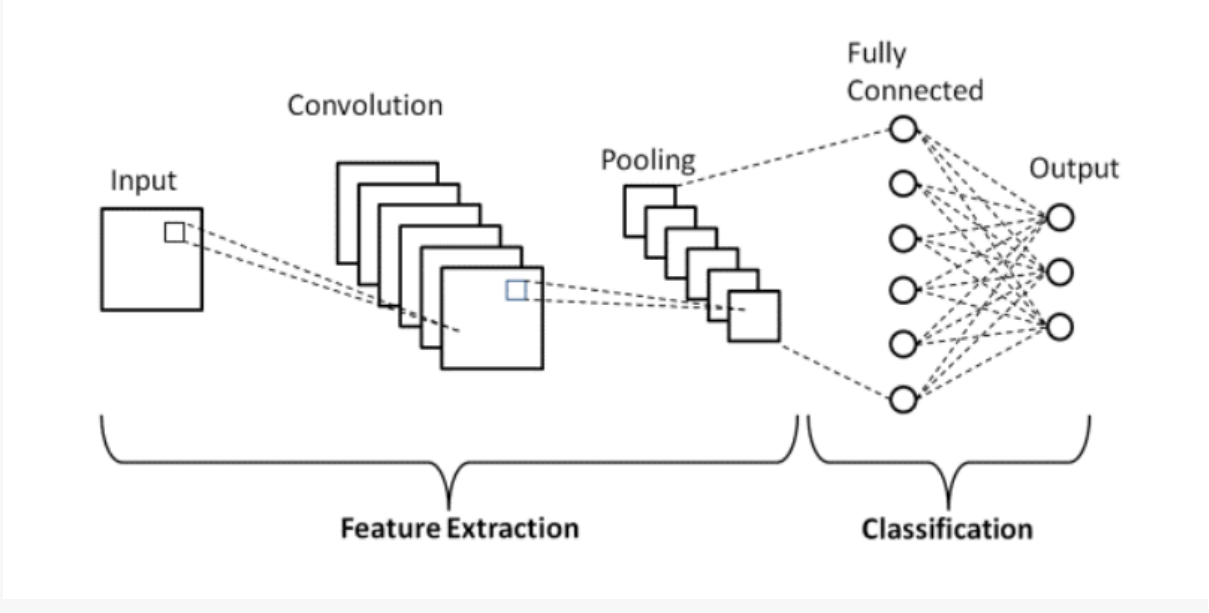
```
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

- **Calculating Performance measures**

```
Accuracy: 0.975925925925926
Precision: 0.9765095615983517
Recall: 0.9764231664735898
F1-score: 0.9763694068986943
```

3.1.3 Convolutional Neural Network

A convolutional neural network (CNN) is a type of artificial neural network commonly used for image recognition and classification tasks. It consists of several layers that work together to extract features from images and classify them. Here is a high-level overview of the key layers in a CNN:



1. **Convolutional Layer**: This layer performs convolution operations on the input image using a set of learnable filters. Each filter extracts a specific feature from the image by sliding over it and computing dot products. The output of this layer is a set of feature maps that represent the presence of different features in the input image.
2. **Activation Layer**: This layer applies an activation function to the output of the convolutional layer. The activation function adds non-linearity to the model, enabling it to learn complex patterns in the data. Common activation functions include ReLU, sigmoid, and tanh.
3. **Pooling Layer**: This layer downsamples the output of the activation layer by reducing its spatial dimensions. It achieves this by taking the maximum, average, or sum of a set of values in a given region of the feature map. This helps to reduce the number of parameters in the model and prevent overfitting.
4. **Fully Connected Layer**: This layer flattens the output of the previous layers and connects every neuron in the layer to every neuron in the next layer. It is essentially a traditional neural network layer that takes in the flattened features and outputs the class scores.

1. Importing Necessary Libraries

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.datasets import mnist
from sklearn.metrics import precision_score, recall_score, f1_score
import time
```

2. Load and split dataset into training and testing sets

```
# Load the MNIST dataset and split it into training and testing sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

3. Normalisation and reshape

```
# Normalize the pixel values of the images to the range [0, 1]
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

# Reshape the input images to have a single color channel
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1))
```

4. Building a model

```
# Define the CNN architecture
model = keras.Sequential(
    [
        layers.Conv2D(32,kernel_size= (3, 3), activation='relu',input_shape=(28, 28, 1)),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="tanh", input_shape=(28, 28, 1)),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="tanh", input_shape=(28, 28, 1)),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dense(64, activation="tanh"),
        layers.Dense(10, activation="softmax"),
    ]
)
```

5. **Compile model (**note - tuning parameters are specific in screenshot**)**

```
# Compile the model with the specified loss function, optimizer, and evaluation metrics
model.compile(
    loss="sparse_categorical_crossentropy",
    optimizer="adam",
    metrics=["accuracy"],
)
```

6. **Training the model**

```
# Train the model on the training dataset
start_time = time.time()
model.fit(x_train, y_train, epochs=5)
end_time = time.time()
```

7. **Evaluating the testing accuracy of model**

```
# Evaluate the model on the testing dataset
test_loss, test_acc = model.evaluate(x_test, y_test)
```

8. **Making prediction**

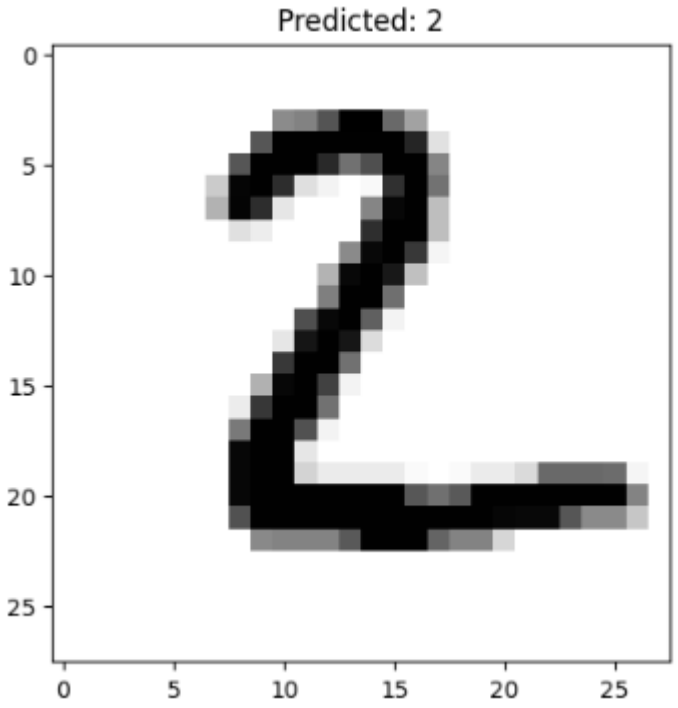
```
# Make predictions on the testing dataset
y_pred = model.predict(x_test)

# Convert the predicted probabilities to class labels
y_pred = tf.argmax(y_pred, axis=1)
```

9. **Calculating performance parameters**

```
# Calculate the precision, recall, and F1 score of the model
precision = precision_score(y_test, y_pred, average="macro")
recall = recall_score(y_test, y_pred, average="macro")
f1 = f1_score(y_test, y_pred, average="macro")
```

10.Printing the result

```
Epoch 1/5
1875/1875 [==============================] - 40s 21ms/step - loss: 0.1923 - accuracy: 0.9440
Epoch 2/5
1875/1875 [==============================] - 40s 21ms/step - loss: 0.0693 - accuracy: 0.9791
Epoch 3/5
1875/1875 [==============================] - 40s 22ms/step - loss: 0.0522 - accuracy: 0.9835
Epoch 4/5
1875/1875 [==============================] - 39s 21ms/step - loss: 0.0406 - accuracy: 0.9876
Epoch 5/5
1875/1875 [==============================] - 41s 22ms/step - loss: 0.0334 - accuracy: 0.9894
313/313 [==============================] - 3s 9ms/step - loss: 0.0512 - accuracy: 0.9851
313/313 [==============================] - 2s 7ms/step
Test loss: 0.0511716865003109
Test accuracy: 0.9850999712944031
Precision: 0.9850470773448992
Recall: 0.9851073282754841
F1 score: 0.985038393430359
Execution time: 203.10846853256226
```



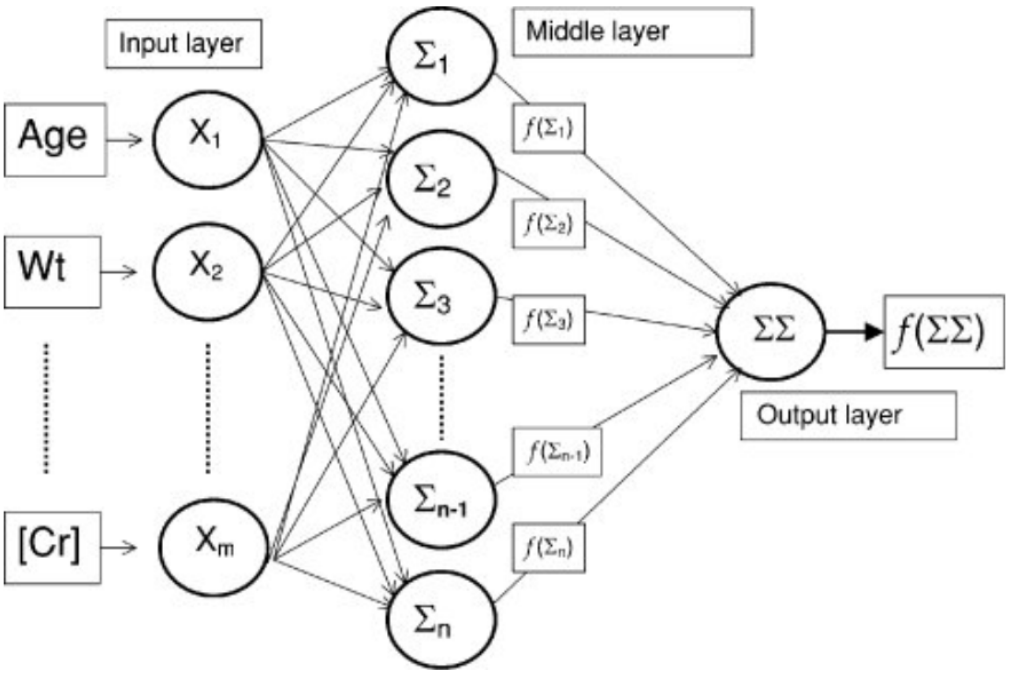
3.1.4 Artificial Neural Network

Artificial Neural Networks (ANN) are a type of machine learning model inspired by the structure and function of the human brain. They consist of a series of interconnected nodes, or neurons, which process and transmit information.

In an ANN, input data is fed into an input layer, which passes the information to one or more hidden layers consisting of neurons. The neurons in each layer receive weighted input from the neurons in the previous layer, and apply a nonlinear activation function to the weighted input in order to produce an output. The output from the last layer is then used to make a prediction or decision.

ANNs are trained using a supervised learning approach, where the network is presented with a set of labeled training data and adjusts its weights and biases to minimize the difference between its predicted output and the actual output. This process is called backpropagation.

Artificial Neural Networks (ANNs) consist of several layers, each having a specific purpose in the overall process of learning and prediction. The main layers in an ANN are:



- Input layer: This layer receives input data, which is usually a vector of features that describe the input data. The size of the input layer is determined by the number of features in the input data.
- Hidden layers: These layers perform computations on the input data and apply transformations to generate new features. The number of hidden layers and the number of neurons in each layer are hyperparameters that need to be tuned during model training.
- Output layer: This layer produces the final prediction based on the computations performed in the previous layers. The size of the output layer depends on the number of classes in the classification problem or the number of values to be predicted in the regression problem.

1. Import Necessary Libraries

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from keras.datasets import mnist
from keras.utils import to_categorical
import time
import matplotlib.pyplot as plt
```

2. Load MNIST dataset

```
# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

3. Preprocess by normalisation and reshape

```
# Preprocess the data
test_images = x_test / 255.0
x_train = x_train.reshape(-1, 28*28).astype('float32') / 255.0
x_test = x_test.reshape(-1, 28*28).astype('float32') / 255.0
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

4. Building a model and compile

```
# Build the model
model = Sequential()
model.add(Dense(128, activation='tanh', input_shape=(28*28,),use_bias=True))
model.add(Dense(10, activation='softmax',use_bias=True))
model.compile(optimizer='RMSprop', loss='categorical_crossentropy', metrics=['accuracy'])
```

//here activation and optimizers are being alters many time to get the best result

5. Training the model

```
# Train the model
start_time = time.time()
model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))
end_time = time.time()
print('Execution time:', end_time - start_time, 'seconds')
```

6. Evaluate the model

```
# Evaluate the model
loss, accuracy = model.evaluate(x_test, y_test)
print('Test loss:', loss)
print('Test accuracy:', accuracy)
```

7. Predicting the output from the model

```
# Predict on test data
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
```

8. Calculating Performance measures

```
# Calculate precision, recall, and f1 score
from sklearn.metrics import precision_recall_fscore_support
precision, recall, f1_score, _ = precision_recall_fscore_support(np.argmax(y_test, axis=1), y_pred_classes, average='weighted')
print('Precision:', precision)
print('Recall:', recall)
print('F1 score:', f1_score)
```

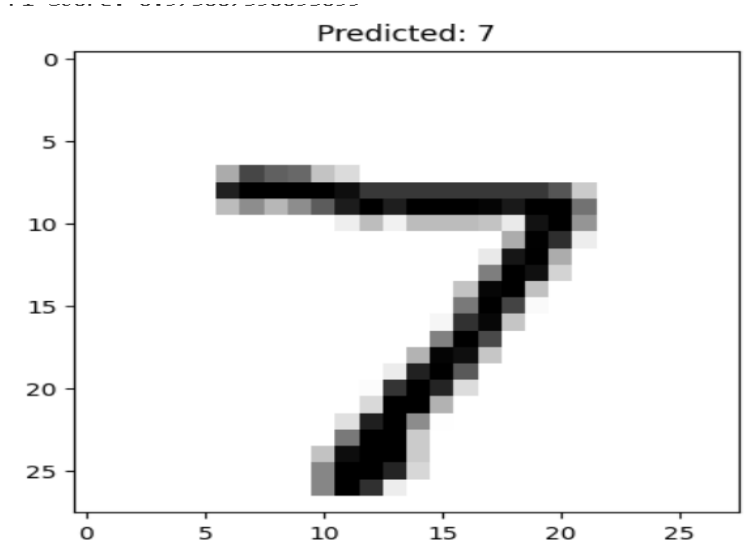
9. Test the model

```
for i in range(10):
    plt.imshow(test_images[i], cmap=plt.cm.binary)
    plt.title(f"Predicted: {np.argmax(y_pred[i])}")
    plt.show()
```

10. Run Time epoch and result (note-this output is for a particular tuning parameters)

```
Epoch 1/10
1875/1875 [==============================] - 13s 7ms/step - loss: 0.2802 - accuracy: 0.9188 - val_loss: 0.1780 - val_accuracy: 0.9464
Epoch 2/10
1875/1875 [==============================] - 10s 5ms/step - loss: 0.1426 - accuracy: 0.9585 - val_loss: 0.1145 - val_accuracy: 0.9652
Epoch 3/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1001 - accuracy: 0.9710 - val_loss: 0.0937 - val_accuracy: 0.9722
Epoch 4/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0773 - accuracy: 0.9773 - val_loss: 0.0847 - val_accuracy: 0.9736
Epoch 5/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0619 - accuracy: 0.9820 - val_loss: 0.0831 - val_accuracy: 0.9735
Epoch 6/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0507 - accuracy: 0.9852 - val_loss: 0.0755 - val_accuracy: 0.9769
Epoch 7/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0429 - accuracy: 0.9874 - val_loss: 0.0729 - val_accuracy: 0.9763
Epoch 8/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0359 - accuracy: 0.9899 - val_loss: 0.0712 - val_accuracy: 0.9770
Epoch 9/10
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0309 - accuracy: 0.9910 - val_loss: 0.0734 - val_accuracy: 0.9775
Epoch 10/10
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0260 - accuracy: 0.9931 - val_loss: 0.0769 - val_accuracy: 0.9756
Execution time: 83.18198132514954 seconds
313/313 [==============================] - 1s 2ms/step - loss: 0.0769 - accuracy: 0.9756
```

```
Execution time: 83.18198132514954 seconds
313/313 [==============================] - 1s 2ms/step - loss: 0.0769 - accuracy: 0.9756
Test loss: 0.0769491121172905
Test accuracy: 0.9756000004196167
313/313 [==============================] - 1s 2ms/step
Precision: 0.9757810831759587
Recall: 0.9756
F1 score: 0.975607590695699
```



3.1.5 Support Vector Machine

Supervised Machine Learning Algorithm used for classification and/or regression. It is more preferred for classification but is sometimes very useful for regression as well. Basically, SVM finds a hyper-plane that creates a boundary between the types of data.

A kernel is nothing but a measure of similarity between data points. The kernel function in a kernelized SVM tells you, that given two data points in the original feature space, what the similarity is between the points in the newly transformed feature space.

```
import numpy as np

from sklearn import svm, metrics

from sklearn import datasets, svm

from sklearn.model_selection import train_test_split


# Load the digits dataset

digits = datasets.load_digits()


# Split the dataset into training and testing sets
```

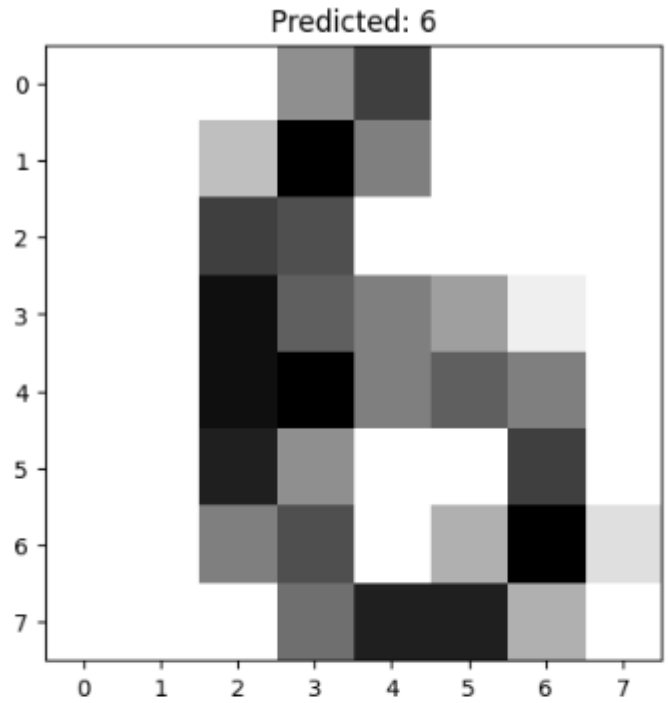
```
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.2, random_state=42)

# Create an SVM classifier with a radial basis function kernel
clf = svm.SVC(kernel='linear', C=1, gamma='auto')

# Fit the SVM classifier on the training data
clf.fit(X_train, y_train)

# Predict the labels of the testing data
y_pred = clf.predict(X_test)
# Calculate precision, recall, f1-score, and accuracy of the SVM classifier
precision = metrics.precision_score(y_test, y_pred, average='weighted')
recall = metrics.recall_score(y_test, y_pred, average='weighted')
f1_score = metrics.f1_score(y_test, y_pred, average='weighted')
accuracy = metrics.accuracy_score(y_test, y_pred)

# Print the precision, recall, f1-score, and accuracy of the SVM classifier
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
print("Accuracy:", accuracy)

# Display some sample predictions
for i in range(10):
    plt.imshow(X_test[i].reshape(8, 8), cmap=plt.cm.gray_r)
    plt.title(f"Predicted: {predictions[i]}")
    plt.show()
```

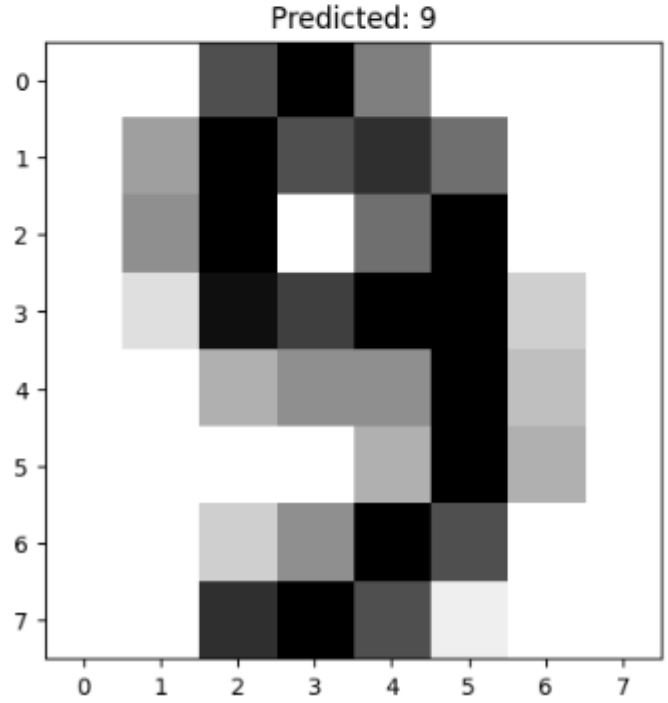
Precision: 0.9778736953613801
Recall: 0.9777777777777777
F1-score: 0.9776863807154582
Accuracy: 0.9777777777777777

Now applying GridSearchcv for kernel = linear, poly, rbf and C = 1, 5, 10, 15 and finding the best score and parameters

```
from sklearn.model_selection import GridSearchCV

clf = GridSearchCV(svm.SVC(gamma='auto'), {

   'C':[1,5,10,15],

   'kernel': ['rbf','linear','poly']

   }, cv = 3, return_train_score=False)

clf.fit(X_train, y_train)

best_C = clf.best_params_['C']

best_kernel = clf.best_params_['kernel']

svm_model = svm.SVC(kernel=best_kernel, C=best_C)

svm_model.fit(X_train, y_train)


# Evaluate the SVM model on the test data

test_accuracy = svm_model.score(X_test, y_test)

print('Test accuracy:', test_accuracy)


# Make predictions on the test data and calculate the recall, precision, and F1 score

y_pred = svm_model.predict(X_test)

recall = recall_score(y_test, y_pred, average='macro')

precision = precision_score(y_test, y_pred, average='macro')

f1 = f1_score(y_test, y_pred, average='macro')

print('Recall:', recall)

print('Precision:', precision)
```

```
print('F1 score:', f1)

Clf.best_params_
```





Test Score: 0.984690
Recall: 0.9865978306216103
Precision: 0.9867321407559204
F1 score: 0.9865810542797318

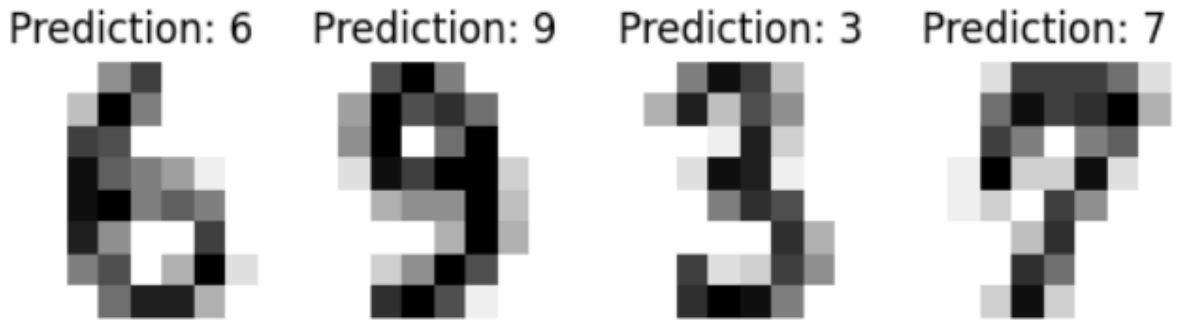
Best Parameters: 'C': 1, 'kernel': 'poly'

3.1.6 Logistic Regression

Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class. It is used for classification algorithms its name is logistic regression.

The logistic regression model transforms the linear regression function continuous value output into categorical value output using a sigmoid function, which maps any real-valued set of independent variables input into a value between 0 and 1.
Equation:

$$p(X;b,w) = \frac{e^{w\cdot X+b}}{1+e^{w\cdot X+b}} = \frac{1}{1+e^{-w\cdot X+b}}$$



```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score,
accuracy_score
# Load the digits dataset
digits = load_digits()
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.2)
# Create a logistic regression classifier
clf = LogisticRegression()
# Fit the logistic regression classifier on the training data
clf.fit(X_train, y_train)
# Predict the labels of the testing data
y_pred = clf.predict(X_test)
# Calculate precision, recall, f1-score, and accuracy of the logistic
regression classifier

precision = precision_score(y_test, y_pred, average=&#39;weighted&#39;)
recall = recall_score(y_test, y_pred, average=&#39;weighted&#39;)
f1_score = f1_score(y_test, y_pred, average=&#39;weighted&#39;)
accuracy = accuracy_score(y_test, y_pred)
# Print the precision, recall, f1-score, and accuracy of the logistic
regression classifier
print(&quot;Precision:&quot;, precision)
print(&quot;Recall:&quot;, recall)
print(&quot;F1-score:&quot;, f1_score)
```

print("Accuracy:", accuracy)

Chapter-4 : Results

Support Vector Machine

SNo	Kernel	C	F1 Score	Recall	Precision	Accuracy
1	linear	1	0.97768	0.977977	0.977873	0.977977
4	poly	1	0.9860745	0.986111	0.986196	0.98611
7	rbf	1	0.520404	0.4666666	0.91976	0.46666
2	linear	5	0.977686	0.977677	0.977873	0.977677
5	poly	5	0.98607	0.98611	0.986196	0.98611
8	rbf	5	0.5853035	0.522222	0.92114	0.5222222
3	linear	10	0.97768	0.977877	0.977873	0.977877
6	poly	10	0.986074	0.98611	0.9861964	0.9861111
9	rbf	10	0.58530	0.52222	0.921143	0.522222

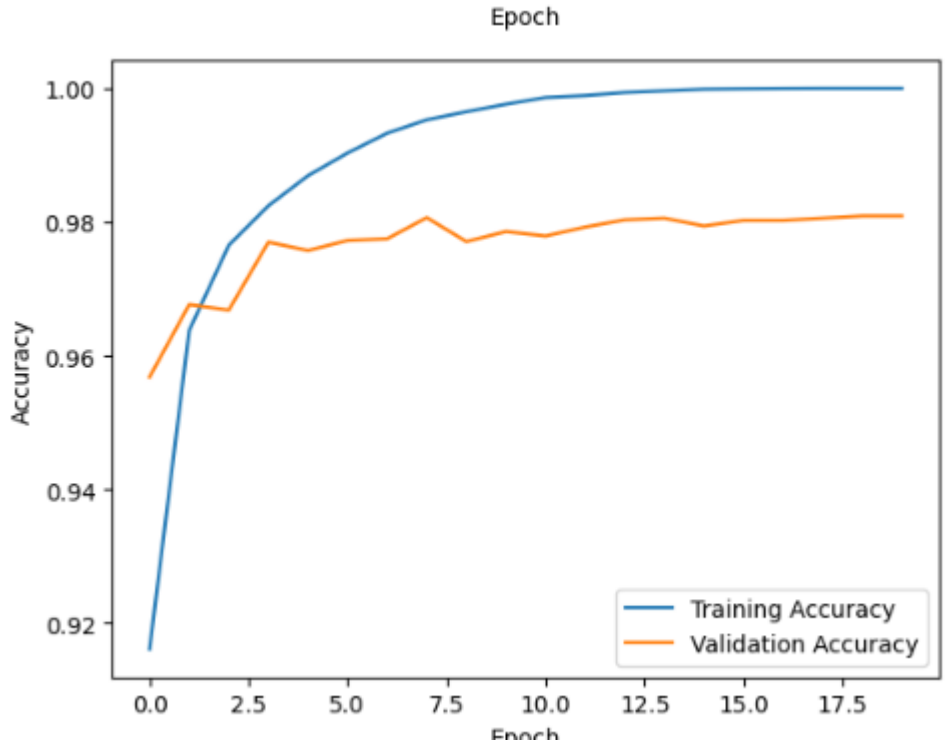
ANN 28*28

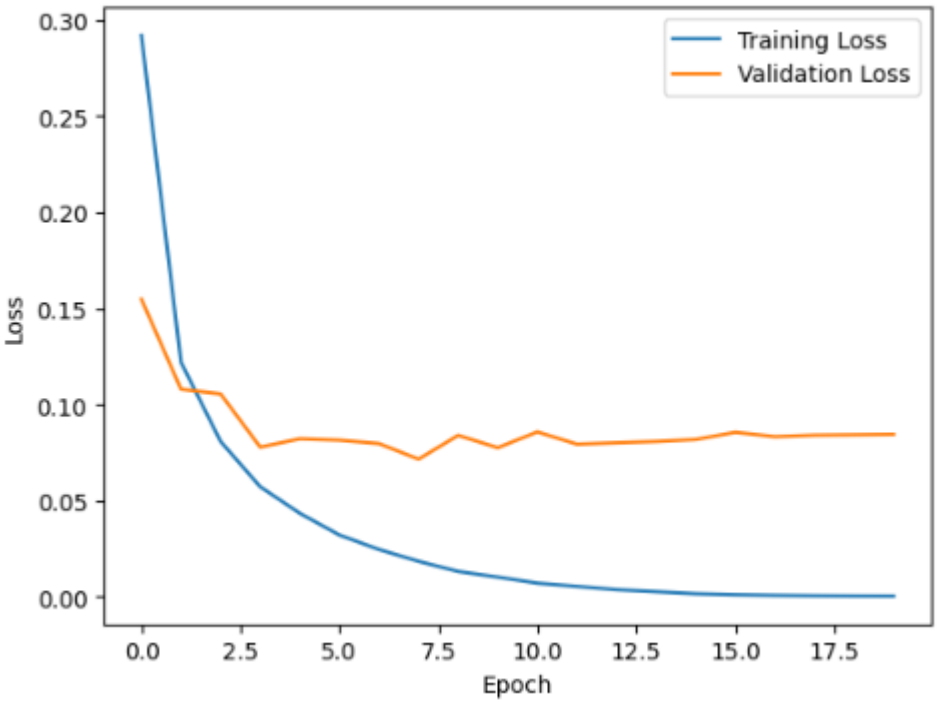
Activation	**tanh**	**sigmoid**	**relu**	**tanh**	**relu**	**relu**	**sigmoid**
Optimizer	**adam**	**adam**	**adam**	**RMSprop**	**RMSprop**	**SGD**	**RMSprop**
F1 - score	0.9787	0.9783	0.9790	0.9756	0.9786	0.9518	0.9762
Precision	0.97882	0.9784	0.9790	0.9757	0.9787	0.9518	0.9763
Recall	0.978	0.978	0.979	0.9756	0.9787	0.9519	0.9762
Accuracy	0.9788	0.9783	0.9788	0.9756	0.9786	0.9519	0.9761
Execution time	83.064 seconds	142.84 seconds	142.87 secon	83.18198seconds	142.7178 seconds	62.910 seconds	80.0166261 seconds

			ds				

ANN 8*8

Activation	tanh	sigmoid	relu	tanh	relu	relu	sigmoid
Optimizer	adam	adam	adam	RMSprop	RMSprop	SGD	RMSprop
F1 - score	0.9586	0.9719	0.9518	0.9626	0.9722	0.9335	0.9676
Precision	0.9582	0.9723	0.9516	0.9636	0.9727	0.9383	0.9683
Recall	0.9598	0.9717	0.9524	0.9630	0.9723	0.9327	0.9671
Accuracy	0.9583	0.9722	0.9527	0.9638	0.9722	0.9361	0.9666
Execution time	1.6199 seconds	4.47825seconds	3.98090seconds	2.0130186 seconds	1.350089seconds	2.278602seconds	2.2262 seconds





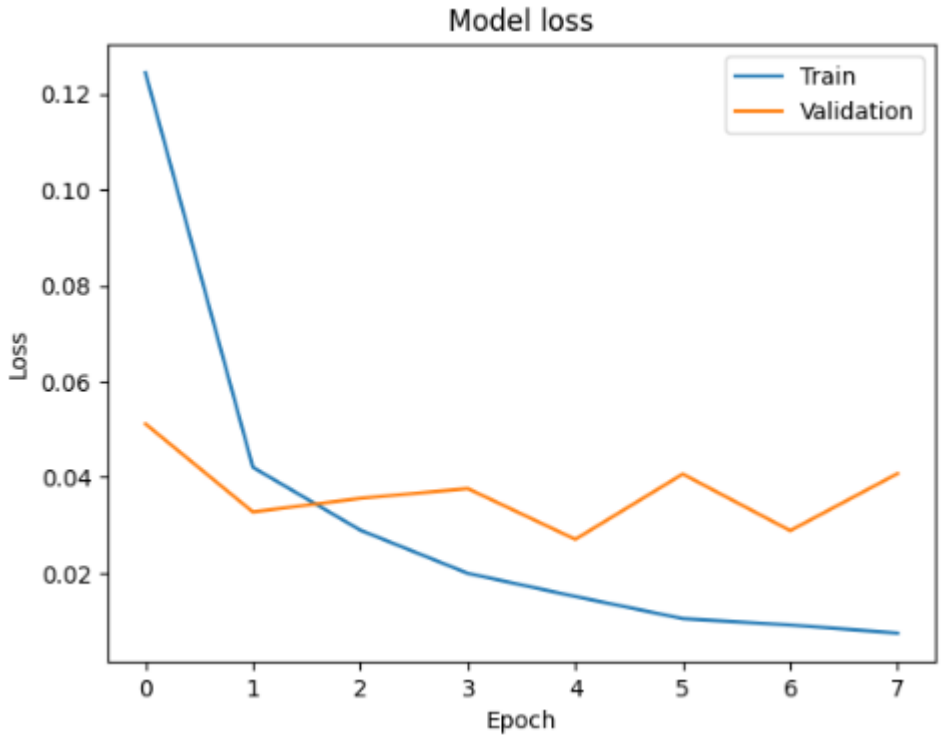
CNN 28*28

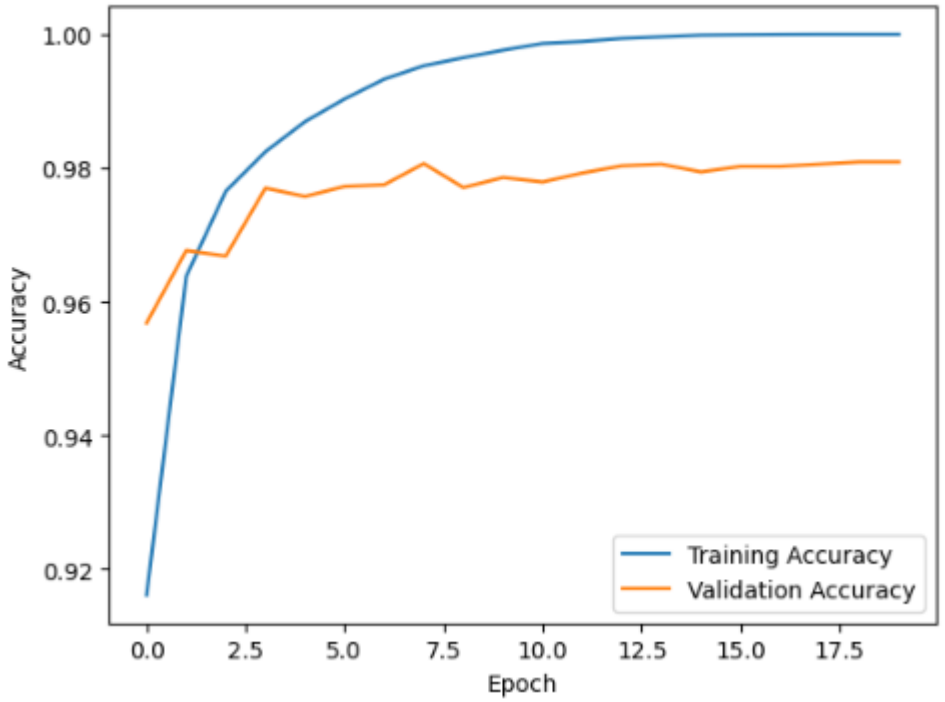
Activation	**tanh**	**sigmoi**	**relu**	**tanh**	**relu**	**relu**	**sigmoid**

		d					
Optimizer	**adam**	**adam**	**adam**	**RMSprop**	**RMSprop**	**SGD**	**RMSprop**
F1 - score	0.9850	0.9829	0.9796	0.98677	0.9845	0.9759	0.9779
Precision	0.9850	0.9829	0.9803	0.9869	0.9849	0.9760	0.9781
Recall	0.9851	0.9830	0.9794	0.9866	0.9843	0.9760	0.9779
Accuracy	0.9850	0.9830	0.9797	0.9868	0.9846	0.9761	0.9781
Execution time	203.108 seconds	262.887 seconds	262.848seconds	202.6773 seconds	262.75241 seconds	262.65 seconds	202.9133 seconds

CNN 8*8

Activation	**tanh**	**sigmoid**	**relu**	**tanh**	**relu**	**relu**	**sigmoid**
Optimizer	**adam**	**adam**	**adam**	**RMSprop**	**RMSprop**	**SGD**	**RMSprop**
F1 - score	0.99	0.99	0.99	0.99	0.99	0.98	0.99
Precision	0.99	0.99	0.99	0.99	0.99	0.98	0.99
Recall	0.99	0.99	0.99	0.99	0.99	0.98	0.99
Accuracy	0.9861	0.9833	0.99166	0.9916	0.9889	0.9805	0.9861
Execution time	6.9766	6.9766	6.9766	6.9766	6.9766	6.9766 seconds	6.9766



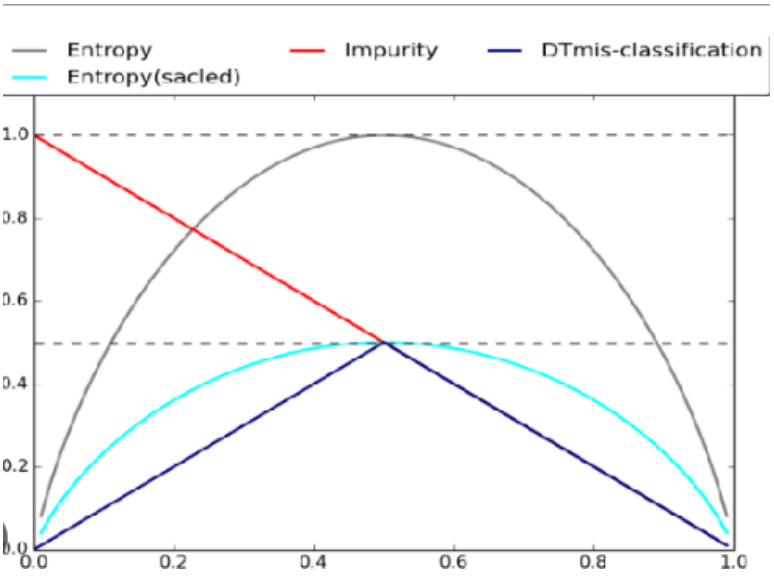


Decision Trees- 8X8

	CV=5	CV=10	CV=15
Accuracy	84.72%	85.55%	85.27%
Precision	85.83%	85.26%	85.95%
Recall	85.04%	85.02%	84.62%
FI-Score	84.58%	85.06%	84.99%

Decision Tree 28X28

	CV=5	CV=10	CV=15
Accuracy	88%	85.65%	87.27%
Precision	88%	88.26%	85.65%
Recall	88%	85.02%	83.62%
FI-Score	88%	85.06%	88%

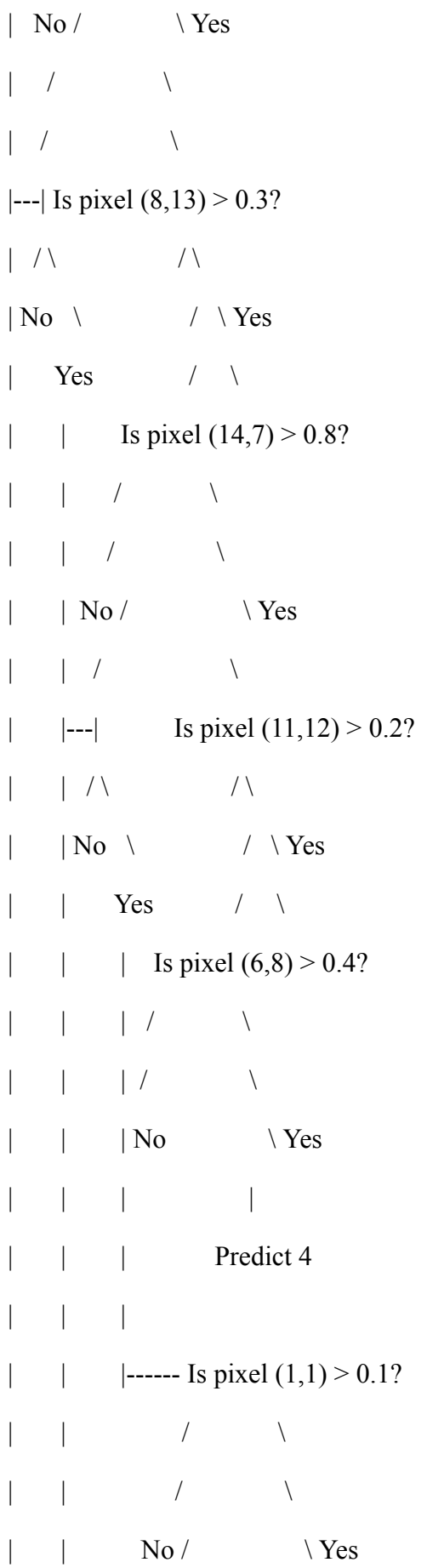


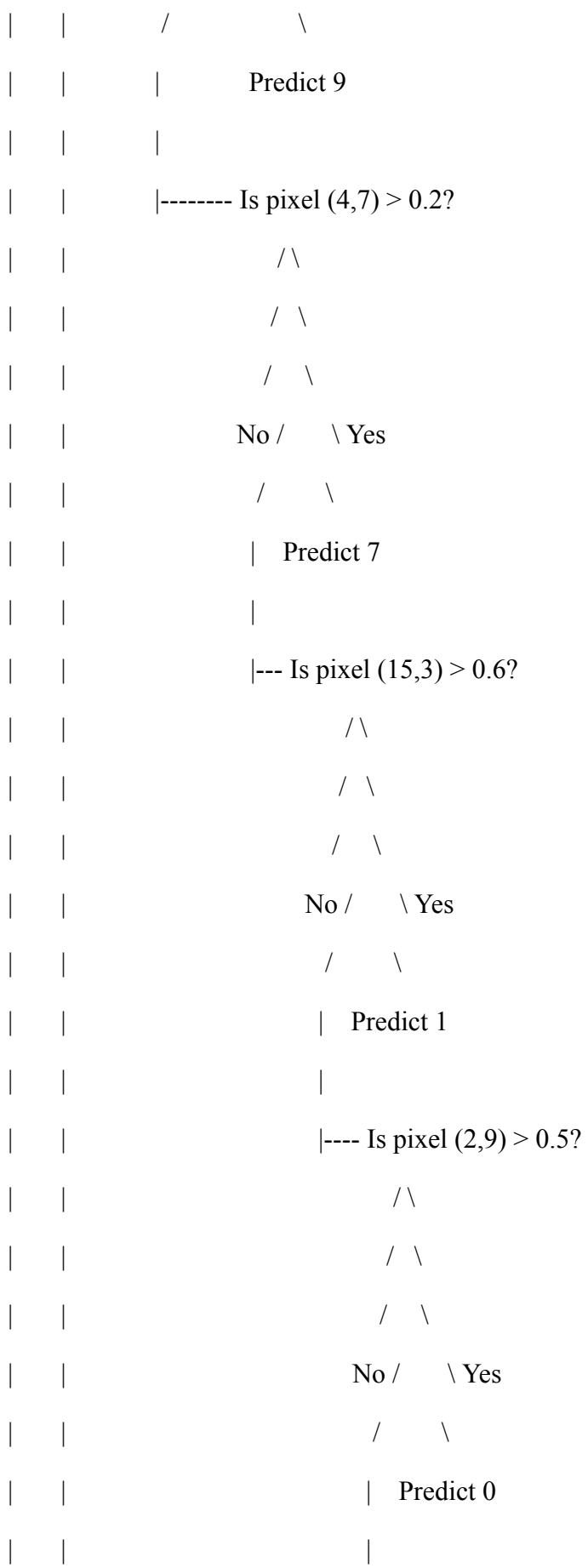
DECISION TREE LOOKS LIKE

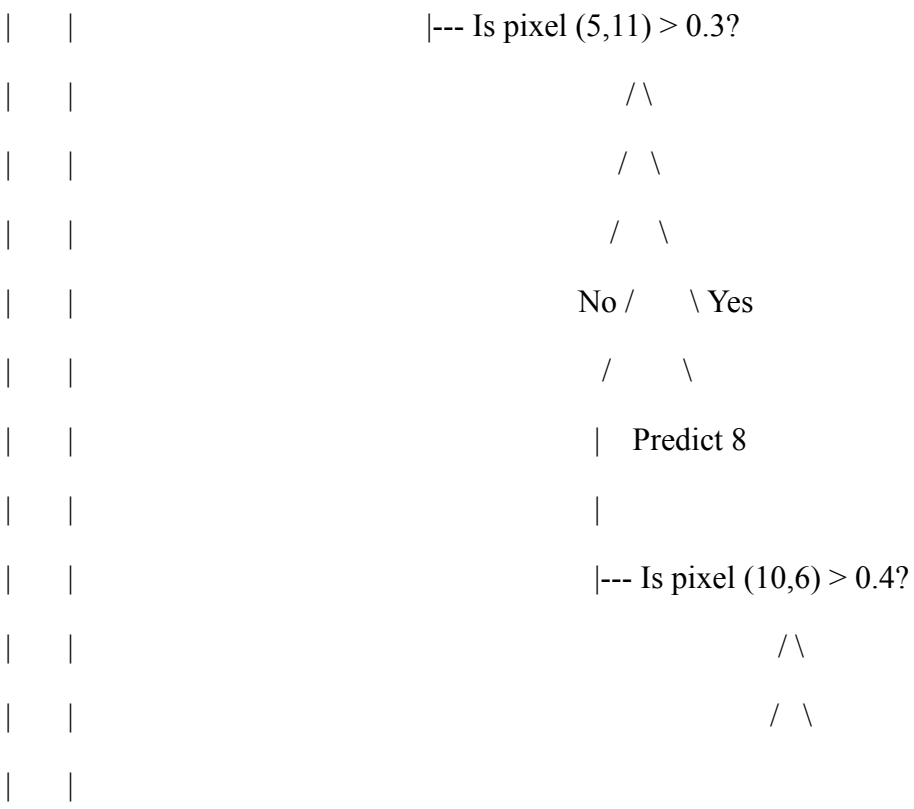
|--------- Is pixel (12,4) > 0.5?

| / \

| / \


No
Yes
Is pixel (8,13) > 0.3?
No
Yes
Is pixel (14,7) > 0.8?
No
Yes
Is pixel (11,12) > 0.2?
No
Yes
Is pixel (6,8) > 0.4?
No
Yes
Predict 4
Is pixel (1,1) > 0.1?
No
Yes


Predict 9
Is pixel (4,7) > 0.2?
No
Yes
Predict 7
Is pixel (15,3) > 0.6?
No
Yes
Predict 1
Is pixel (2,9) > 0.5?
No
Yes
Predict 0



Random Forest

8x8

	CV=5	**CV=10**	**CV=15**
Accuracy	97.59%	85.65%	87.27%
Precision	97.69%	88.26%	85.65%
Recall	97.64%	85.02%	83.62%
FI-Score	97.63%	85.06%	88%

LIMITATION OF MINOR PROJECT

Handwritten digit recognition has made significant progress in recent years, but it still needs several limitations that researchers and developers are working to overcome. Here are some of the main limitations of handwritten digit recognition:

1. **Variability in handwriting**: One of the main challenges of handwritten digit recognition is the variability in handwriting styles. People write digits in different ways, and this variability can make it difficult for recognition systems to accurately identify the digit.
2. **Image quality:** The quality of the image can also affect the accuracy of the recognition system. Handwritten images can be blurry, noisy, or contain artefacts, which can make it difficult for the system to accurately identify the digit.
3. **Overfitting:** Overfitting occurs when the recognition system is trained on a specific dataset and becomes too specialised, resulting in poor performance on new data. This can limit the generalizability of the recognition system and its ability to accurately identify digits in different contexts.

4. **Limited dataset size**: The availability of a large dataset is critical for training recognition systems. However, there are a limited number of labelled datasets available for handwritten digit recognition, which can limit the performance of the recognition system.
5. **Lack of context**: Handwritten digits are often part of a larger document, such as a form or letter, which provides context for the digit. Without this context, the recognition system may struggle to accurately identify the digit.

FUTURE WORK

There are several areas of future work for handwritten digit recognition. Here are some of the most promising directions for research:

Improved algorithms and architectures: While deep learning has shown great success in handwritten digit recognition, there is still room for improvement in the algorithms and architectures used. Researchers are exploring novel deep-learning architectures and techniques, such as capsule networks and attention mechanisms, which may lead to further improvements in accuracy.

Multi-modal approaches: Multi-modal approaches combine different types of data, such as images and text, to improve recognition accuracy. Future work may explore the use of multi-modal approaches for handwritten digit recognition, which could potentially improve the accuracy and robustness of the recognition system.

Transfer learning: Transfer learning is the process of transferring knowledge from one domain to another. This could be applied to handwritten digit recognition by pre-training recognition systems on large datasets of related

tasks, such as recognizing letters or symbols, before fine-tuning them on the digit recognition task.

Real-time recognition: Real-time recognition is critical for applications such as signature verification or online handwriting recognition. Future work may focus on developing real-time recognition systems that can recognize digits as they are being written, which could have important practical applications.

Unsupervised learning: Unsupervised learning is the process of learning from unlabeled data. Future work may explore the use of unsupervised learning techniques, such as autoencoders or generative adversarial networks, for handwritten digit recognition. This could potentially improve recognition accuracy and address the limited dataset size limitation

1.