**HANDWRITTEN DIGIT RECOGNITION**

**USING DEEP LEARNING**

**A PROJECT REPORT**

**Submitted by**

**MIHIR MISHRA**

3rd Year (6th Sem)

Enroll no.: 06918002720

*in partial fulfillment for the award of the degree*

*of*

**Bachelor of Technology**

*in*

**COMPUTER SCIENCE AND ENGINEERING**

****

**DELHI TECHNICAL CAMPUS**

**GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY**

**YEAR**

2020-2024

# **CERTIFICATE**

**Certified that this project report “HAND WRITTEN DIGIT RECOGNITION USING DEEP LEARNING” is the bonafide work of “MIHIR MISHRA (06918002720)” who carried out the project work under my supervision.**

|  |  |  |
| --- | --- | --- |
| **MEGHA KUMAR**  **Guide** | **Dr. SEEMA VERMA**  **Head of the Department** | **Dr. PRANAY TANWAR**  **Dean Academics** |

**COMPUTER SCIENCE AND ENGINEERING**

****

**DELHI TECHNICAL CAMPUS**

**GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY,**

**New Delhi**

# **ACKNOWLEDGEMENT**

I would like to express my special thanks of gratitude to my project guide Ms. Megha Kumar as well as our Dean Academics Dr. Pranay Tanwar who gave me the golden opportunity to do this wonderful project on the topic HANDWRITTEN DIGIT RECOGNITION, which also helped me in doing a lot of research and I came to know about so many new things I am really thankful to them.

Secondly, I would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.

|  |  |
| --- | --- |
| Date : | MIHIR MISHRA |
| Place : | 06918002720 |
|  | BTECH – CSE |

# **ABSTRACT**

Handwritten digit recognition has become increasingly important in the field of computer vision. In this project, we aim to develop a more accurate and efficient approach to accurately recognize and predict handwritten digits from 0 to 9. Our focus lies on leveraging a class of multilayer feed-forward neural network known as Convolutional Neural Network (CNN).

For this project, we will employ the widely used MNIST dataset for both training and recognition purposes. The dataset consists of a total of 70,000 images in which 60,000 for training and 10,000 for testing, comprising handwritten digits 0-9. Each digit is represented as a grayscale image with dimensions of 28x28 pixels, ensuring optimal results.

The digits from the dataset will be fed into the input layers of the CNN model, followed by a series of hidden layers containing convolutional, activation, and Dense layers. Subsequently, the processed data will be mapped to the fully connected layer and passed through a softmax and sigmoid classifier to accurately classify the digits.

To implement this network, we will utilize the Keras deep learning library, a powerful and user-friendly Python library specifically designed for building neural networks.

By implementing this advanced CNN architecture, we aim to achieve superior performance and accuracy in handwritten digit recognition, thereby contributing to the field of computer vision and deep learning.

# **LISTS OF FIGURES**

[Fig 2.1 Deep Neural Networks 4](#_Toc136865653)

[Fig. 3.1 Steps in model development 7](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865654)

[Fig 3.2 MNIST Dataset 12](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865655)

[Fig. 3.3 MNIST data processing classification with output 13](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865656)

[Fig. 3.4 Working of SVM on a dataset 17](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865657)

[Fig. 3.5 KNN Model 22](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865658)

[Fig. 3.6 output 24](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865659)

[Fig. 3.7 Working of CNN 27](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865660)

[Fig. 3.8 Working of CNN with HRD 30](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865661)

[Fig. 4.1 CNN Architecture 38](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865662)

[Fig. 4.2 Convolutional Layer Architecture 40](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865663)

[Fig 4.3 Pooling Layer 41](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865664)

[Fig 4.4 Fully connected Layer 42](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865665)

[Fig. 5.1 Accuracy and Loss Graph 48](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865666)

[Fig 5.2 Importing Libraries & normalizing the dataset 49](#_Toc136865667)

[Fig. 5.3 Shape of dataset 50](#_Toc136865668)

[Fig. 5.4 Resizing the Image 50](#_Toc136865669)

[Fig 5.5 Creating a DNN Model 51](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865670)

[Fig 5.6 Summary of the model 52](#_Toc136865671)

[Fig 5.7 Compilation of model 53](#_Toc136865672)

[Fig. 5.8 Fit the model 54](#_Toc136865673)

[Fig.5.9 Output of the Trained Model 54](#_Toc136865674)

[Fig. 5.10 Evaluation of Dataset 54](#_Toc136865675)

[Fig. 5.11 Testing & Output of the Test Model 55](#_Toc136865676)

[Fig 5.12 Confusion Matrix 56](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865677)

[Fig. 5.13 Code for Image Read 56](#_Toc136865678)

[Fig 5.14 Prediction of model 57](#_Toc136865679)

[Fig 5.15 Output ‘digit – 9’ 57](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865680)

[Fig 5.16 Output ‘digit – 8’ 58](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865681)

[Fig 5.17 Output ‘digit – 1’ 58](file:///D:\MIHIR\Project%20Report\HANDWRITTEN%20DIGIT%20RECOGNITION%20Project%20Report.docx#_Toc136865682)

[Fig 5.18 Output ‘digit – 3’ 59](#_Toc136865683)

[Fig 5.19 Output ‘digit – 0’ 59](#_Toc136865684)

[Fig 5.20 Output ‘digit – 6’ 59](#_Toc136865685)

**TABLE OF CONTENTS**

[**CERTIFICATE** i](#_Toc136882870)

[**ACKNOWLEDGEMENT** ii](#_Toc136882871)

[**ABSTRACT** iii](#_Toc136882872)

[**LISTS OF FIGURES** iv](#_Toc136882873)

[**LIST OF SYMBOLS AND ABBREVIATIONS** viii](#_Toc136882874)

[**CHAPTER 1** 1](#_Toc136882875)

[**INTRODUCTION** 1](#_Toc136882876)

[**1.1 HANDWRITTEN DIGIT RECOGNITION & DEEP LEARNING** 1](#_Toc136882877)

[**1.2 OBJECTIVE** 1](#_Toc136882878)

[**1.3** **DATASET** 2](#_Toc136882879)

[**1.4 BENEFITS** 2](#_Toc136882880)

[**CHAPTER 2** 3](#_Toc136882881)

[**LITERATURE SURVEY** 3](#_Toc136882882)

[**2.1 HANDWRITTEN DIGIT RECOGNITION USING MACHINE LEARNING** 3](#_Toc136882883)

[**2.2 CONCLUSION** 5](#_Toc136882884)

[**CHAPTER 3** 6](#_Toc136882885)

[**METHODOLOGY** 6](#_Toc136882886)

[**3.1 BASIC STEPS IN BUILDINNG A MACHINE LEARNING MODEL** 6](#_Toc136882887)

[**3.1.1 Data Collection** 6](#_Toc136882888)

[**3.1.2 Data Preprocessing** 6](#_Toc136882889)

[**3.1.3 Model Selection** 6](#_Toc136882890)

[**3.1.4 Train The Model** 6](#_Toc136882891)

[**3.1.5 Model Evaluation** 7](#_Toc136882892)

[**3.1.6 Model Optimization** 7](#_Toc136882893)

[**3.2 METHODOLOGIES FOR HANDWRITTEN DIGIT RECOGNITION MODEL** 7](#_Toc136882894)

[**3.2.1 Import The Libraries** 7](#_Toc136882895)

[**3.3 LOAD THE DATASET** 12](#_Toc136882896)

[**3.3.1 MNIST Dataset** 12](#_Toc136882897)

[**3.3.2 Data Cleaning** 13](#_Toc136882898)

[**3.3.3 Data Transformation** 14](#_Toc136882899)

[**3.4 MODEL CONSTRUCTION** 14](#_Toc136882900)

[**3.4.1 Algorithms** 15](#_Toc136882901)

[**3.4.2 Model That Can Be Used for This Project** 16](#_Toc136882902)

[**3.5 TRAINING AND VALIDATION** 30](#_Toc136882903)

[**3.5.1 Adam Optimizer** 31](#_Toc136882904)

[**3.6 MODEL EVALUATION & PREDICTION:** 35](#_Toc136882905)

[**3.6.1 CROSS VALIDATION** 36](#_Toc136882906)

[**CHAPTER 4** 37](#_Toc136882907)

[**CNN ARCHITECTURE** 37](#_Toc136882908)

[**4.1 CNN LAYERS** 38](#_Toc136882909)

[**4.1.1 Convolutional Layer** 39](#_Toc136882910)

[**4.2 POOLING LAYER** 40](#_Toc136882911)

[**4.3 FULLY CONNECTED LAYER** 41](#_Toc136882912)

[**4.4. ACTIVATION FUNCTIONS** 43](#_Toc136882913)

[**4.5 APPLICATION:** 44](#_Toc136882914)

[**CHAPTER 5** 45](#_Toc136882915)

[**EXPERIMENTAL ANALYSIS AND RESULTS** 45](#_Toc136882916)

[**5.1 SYSTEM CONFIGURATION** 45](#_Toc136882917)

[**5.2 LEARNING CURVES** 46](#_Toc136882918)

[**5.2.1 Accuracy Curve** 48](#_Toc136882919)

[**5.2.2 Loss Curve** 48](#_Toc136882920)

[**5.3 CODE SEGMENT** 49](#_Toc136882921)

[**5.3.1 Import the Libraries** 49](#_Toc136882922)

[**5.3.2 Shape of Data** 50](#_Toc136882923)

[**5.3.3 Resizing Image for Convolutional Operations** 50](#_Toc136882924)

[**5.3.4 Creating A Deep Neural Network** 51](#_Toc136882925)

[**5.3.5 Summary of The Model** 52](#_Toc136882926)

[**5.3.6 Compile the CNN Model** 52](#_Toc136882927)

[**5.3.7 Fit the Model** 53](#_Toc136882928)

[**5.3.8 Evaluation of MNIST Dataset** 54](#_Toc136882929)

[**5.3.9 Testing the Model** 55](#_Toc136882930)

[**5.3.10 Confusion Matrix** 56](#_Toc136882931)

[**5.3.11 Prediction of the Model** 56](#_Toc136882932)

[**5.4 OUTPUT** 57](#_Toc136882933)

[**CHAPTER 6** 60](#_Toc136882934)

[**APPENDIX** 60](#_Toc136882935)

[**CHAPTER 7** 64](#_Toc136882936)

[**CONCLUSION AND FUTURE WORK** 64](#_Toc136882937)

[**7.1 CONCLUSION:** 64](#_Toc136882938)

[**7.2 FUTURE WORK:** 64](#_Toc136882939)

[**REFERENCES** 65](#_Toc136882940)

# **LIST OF SYMBOLS AND ABBREVIATIONS**

**ABBREVIATIONS USED :**

1. CNN : Convolutional Neural Networks
2. DNN : Deep Neural Networks
3. HDR : Handwritten Digit Recognition
4. KNN : K-Nearest Neighbors
5. MNIST : Modified National Institute of Standards and Technology
6. SVM : Support Vector Machines

# **CHAPTER 1**

# **INTRODUCTION**

Handwritten digit recognition is a rapidly evolving field within the domain of computer vision and artificial intelligence. A human learns to accomplish a task by practicing and repeating it repeatedly until the skill is learned by memory. When this happens, his brain's neurons automatically get activated, enabling them to carry out the learnt task swiftly. Deep Learning is quite similar to this mechanism.

## **1.1 HANDWRITTEN DIGIT RECOGNITION & DEEP LEARNING**

Handwriting recognition refers to the capability of a machine to interpret and understand handwritten input from various sources, including paper documents, photographs, and touch screen devices. It involves the development of algorithms and systems that can effectively interpret and classify handwritten digits, enabling machines to understand and process human-generated numerical information. This technology finds widespread applications in various sectors, including finance, document processing, and automation.

By automating the recognition process, handwritten digits can be converted into machine-readable format, facilitating efficient data storage, retrieval, and analysis.

In recent years, deep learning techniques, particularly Convolutional Neural Networks (CNNs), have emerged as the state-of-the-art approach for handwritten digit recognition. CNNs offer several advantages, such as their ability to effectively extract features from complex input data, their robustness against variations in handwriting styles, and their capability to learn hierarchical representations from raw pixel values.

## **1.2 OBJECTIVE**

The primary objective of this project is to design and implement an expert system for handwritten digit recognition using deep learning. By leveraging the power of neural networks, we aim to develop a system that can accurately classify handwritten digits ranging from 0 to 9 with a high degree of accuracy and efficiency. To achieve this, we will utilize the MNIST dataset, a widely used benchmark dataset for digit recognition, consisting of thousands of handwritten digit images.

Throughout this project, we will explore the concepts of deep learning, specifically focusing on CNN architectures, and their applications in handwritten digit recognition. We will implement and train a neural network model using the Keras library, a powerful deep learning framework, and evaluate its performance based on accuracy metrics and comparison with existing approaches.

## **DATASET**

The MNIST (Modified National Institute of Standards and Technology) dataset is a widely used benchmark in the field of machine learning. It consists of a collection of handwritten digit images along with their corresponding labels. The MNIST model is a popular example of a deep learning model used to classify these digits accurately. MNIST consists of a large collection of handwritten digits from 0 to 9. The dataset contains a training set of 60,000 grayscale images and a test set of 10,000 grayscale images. Each image is a 28x28 pixel square, representing a handwritten digit. The digits are centered and normalized, making the dataset relatively clean and consistent.

## **1.4 BENEFITS**

The successful completion of this project will contribute to the advancement of handwritten digit recognition techniques and provide valuable insights into the capabilities and limitations of deep learning in this domain. Moreover, it has the potential to enhance automation processes, reduce human effort, and improve the efficiency of various applications that rely on accurate digit recognition.

In the following sections of this report, we will delve into the methodology, dataset, implementation details, experimental results, and analysis of the proposed handwritten digit recognition system. We will also discuss the challenges faced, possible future enhancements, and the broader impact of this research in the field of computer vision and artificial intelligence.

# **CHAPTER 2**

# **LITERATURE SURVEY**

Handwritten digit recognition using deep learning has been the subject of extensive research in recent years. Numerous studies have explored different approaches, architectures, and techniques to improve the accuracy and efficiency of handwritten digit recognition systems. In this literature survey, we will review some of the key works and contributions in the field.

1. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. In Proceedings of the IEEE (Vol. 86, No. 11, pp. 2278-2324).

- This seminal paper introduced the LeNet-5 architecture, a pioneering convolutional neural network for handwritten digit recognition. The authors demonstrated the effectiveness of convolutional layers in capturing local spatial dependencies, leading to superior performance compared to traditional approaches.

2. Simard, P. Y., Steinkraus, D., & Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. In Proceedings of the Seventh International Conference on Document Analysis and Recognition (Vol. 2, pp. 958-963).

- This work presented a comprehensive study on the design and optimization of convolutional neural networks for handwritten digit recognition. The authors explored various architectural choices, training strategies, and preprocessing techniques to achieve state-of-the-art results on the MNIST dataset.

3. Cireşan, D. C., Meier, U., Gambardella, L. M., & Schmidhuber, J. (2010). Deep, big, simple neural nets for handwritten digit recognition. Neural computation, 22(12), 3207-3220.

- This study proposed a deep neural network architecture consisting of multiple convolutional and fully connected layers for handwritten digit recognition. The authors achieved top performance on the MNIST dataset by leveraging the increased capacity of deep networks and incorporating architectural simplicity.

## **2.1 HANDWRITTEN DIGIT RECOGNITION USING MACHINE LEARNING**

In our project, we explored the application of various machine learning algorithms such as K-Nearest Neighbors (KNN), Support Vector Machines (SVM), and Neural Networks for handwritten digit recognition. We experimented with different parameters and feature scaling techniques, comparing the performance of these classifiers in terms of their accuracy and computational time.

The accuracy of the classifiers can vary depending on how the training and testing data are split. Increasing the size of the training and testing datasets can potentially improve accuracy. Each classifier has its own accuracy and time consumption characteristics. Additionally, if the computational power is upgraded from a CPU to a GPU, the classifiers can potentially achieve higher accuracy and faster processing times, leading to improved results.

I evaluated the performance of the classifiers based on their sensitivity (the ability to correctly identify a condition), accuracy (the proportion of true results), positive predictions (the number of false positives from the classification process), and specificity (the ability to correctly exclude a condition). I compared the performance of different machine learning and deep learning classifiers, highlighting that, so far, deep learning algorithms have shown better performance in the application of handwritten digit recognition.

By conducting this comparison and analysis, I aim to gain insights into the strengths and weaknesses of different classifiers and identify the most effective approach for our handwritten digit recognition system.



1 Fig 2.1 Deep Neural Networks

## **2.2 CONCLUSION**

There are a large number of papers and articles are being published these days about this topic. In research, it is shown that Deep Learning algorithm like multilayer CNN using Keras withTheano and Tensorflow gives the highest accuracy in comparison with the most widely used machinelearning algorithms like SVM, KNN & RFC. Because of its highest accuracy, Convolutional NeuralNetwork (CNN) is being used on a large scale in image classification, video analysis, etc. Many researchers are trying to make sentiment recognition in a sentence. CNN is being used in natural language processing and sentiment recognition by varying different parameters. It is pretty challenging to get a good performance as more parameters are needed for the large-scale neural

network.

These works represent a selection of influential research papers in the field of handwritten digit recognition using deep learning. They highlight the advancements in convolutional neural network architectures, training strategies, and regularization techniques that have significantly improved the accuracy and efficiency of digit recognition systems. Building upon these foundational studies, I had used CNN (Convolutional Neural Network) using Keras - Tensorflow with Convolutional layers and Fully connected layers.

# **CHAPTER 3**

# **METHODOLOGY**

## **3.1 BASIC STEPS IN BUILDINNG A MACHINE LEARNING MODEL**

### **3.1.1 Data Collection**

* It is important to ensure the quality and reliability of the collected data.
* The quality and diversity of the collected data directly impact the model's ability to generalize and perform well on unseen digit samples.
* The better and more the data (quantity), the accurate the model will be
* Using the pre-collected, or Online Datasets like available publicly available datasets specifically designed for these tasks, such as MNIST dataset or from kaggle.
* We can also collect data manually, like physically collecting handwritten digit samples and then digitized or scanned for further processing.

### **3.1.2 Data Preprocessing**

* Preprocessing is crucial to ensure data quality and compatibility.
* It typically involves tasks such as data cleaning, handling missing values, scaling features, and encoding categorical variables.
* Preprocessing may also include image normalization, resizing, and noise reduction techniques to enhance the quality of input images.

### **3.1.3 Model Selection**

* For handwritten digit recognition, various models can be considered, such as K-Nearest Neighbors (KNN), Support Vector Machines (SVM), or Convolutional Neural Networks (CNNs).

### **3.1.4 Train The Model**

* The training process involves feeding the training dataset to the model, allowing it to learn patterns and relationships between input features and corresponding digit labels.
* During training, model parameters are adjusted iteratively to minimize a defined loss or error function, optimizing the model's ability to generalize to unseen data.

### **3.1.5 Model Evaluation**

* Evaluation provides insights into the model's ability to correctly classify digits and helps identify areas for improvement.
* Uses a metric or set of metrics to "measure" the model's objective performance.

### **3.1.6 Model Optimization**

* Model optimization aims to improve the model's performance by fine-tuning its parameters or exploring hyperparameter settings.
* Techniques such as cross-validation, grid search, or Bayesian optimization can be employed to find optimal parameter configurations that maximize the model's accuracy and generalization capabilities.

## **3.2 METHODOLOGIES FOR HANDWRITTEN DIGIT RECOGNITION MODEL**

To train the model, I had employed the MNIST dataset, which consists of 70,000 handwritten digit grayscale pictures of 28x28 pixel square, of which 60,000 are used for training and the remaining 10,000 are used for testing validation. Preprocessing, Model Construction, Training & Validation, Model Evaluation & Prediction are the primary stages for the model.

Model Evaluation

Training and testing Validation

Building a model

Prediction

Preprocessing

2 Fig. 3.1 Steps in model development

### **3.2.1 Import The Libraries**

#### **3.2.1.1. TensorFlow**

TensorFlow is a widely-used open-source deep learning framework developed by Google. It is designed to provide a flexible and efficient platform for building and deploying machine learning models. One of the key features of TensorFlow is its ability to construct and train deep neural networks. It provides a high-level API called Keras, which simplifies the process of building and training deep learning models. Keras offers a user-friendly interface and abstracts away many complexities, allowing users to focus on model design and experimentation. The TensorFlow ecosystem offers a wide range of tools and resources for deep learning tasks.

TensorFlow supports both CPU and GPU acceleration, enabling faster computations and improved performance, especially when dealing with large datasets and complex models. It also provides support for distributed computing, allowing the training of models on multiple machines or clusters.

TensorFlow was created with the intention of being used in both research and development and production systems, in contrast to other numerical libraries developed for Deep Learning, such as Theano. This tutorial is set up so that we can quickly implement deep learning projects on TensorFlow.

To install, write this in command prompt :

* pip install tensorflow

#### **3.2.1.2. Keras**

Keras is a high-level neural network library that runs on top of TensorFlow, as well as other deep learning frameworks like Theano and Microsoft Cognitive Toolkit (CNTK). It provides a user-friendly and intuitive interface for building and training deep learning models.

One of the key advantages of Keras is its simplicity and ease of use. It allows users to quickly prototype and experiment with different architectures and configurations. Keras provides a wide range of pre-built layers and modules that can be easily combined to construct complex neural network architectures. These include convolutional layers, recurrent layers, dense (fully connected) layers, and many more.

Keras supports both sequential and functional API styles. The sequential API is suitable for building models with a linear stack of layers, while the functional API allows for more complex model architectures with shared layers, multiple inputs/outputs, and branching structures.

Keras also provides a comprehensive set of tools for model training and evaluation. It includes various optimizers (e.g., stochastic gradient descent, Adam), loss functions (e.g., categorical cross-entropy, mean squared error), and metrics (e.g., accuracy, precision, recall). Keras makes it easy to compile models with these components and train them on training data, while monitoring and evaluating performance on validation data.

Furthermore, Keras supports GPU acceleration, which can significantly speed up training and inference times. By leveraging the underlying backend (e.g., TensorFlow), Keras seamlessly utilizes the computational power of GPUs to perform efficient computations.

To install, write this in command prompt :

* pip install keras

#### **3.2.1.3. Matplotlib**

Matplotlib is a widely-used plotting library in Python that provides a comprehensive set of tools for creating various types of visualizations. It is highly customizable and offers a wide range of plotting options, making it suitable for data exploration, analysis, and presentation.

Key features of Matplotlib include:

1. Plotting Functions: Matplotlib provides a variety of plotting functions that enable the creation of different types of plots, such as line plots, scatter plots, bar plots, histograms, and more. These functions allow for the visualization of data in both two-dimensional and three-dimensional space.
2. Customization: Matplotlib offers extensive customization options, allowing users to tailor the appearance of their plots to suit their specific needs. Properties such as colors, line styles, marker styles, labels, titles, and legends can be adjusted to create visually appealing and informative plots.
3. Subplots and Layouts: Matplotlib allows the creation of multiple subplots within a single figure, enabling the comparison and visualization of multiple datasets or aspects of data. Users can define the arrangement of subplots using grids or custom layout specifications.
4. Exporting and Saving: Matplotlib provides options for exporting plots in various formats, including PNG, JPEG, PDF, SVG, and more. This allows users to save their plots for use in reports, presentations, or further analysis.
5. Integration with NumPy and Pandas: Matplotlib seamlessly integrates with other popular scientific computing libraries in Python, such as NumPy and Pandas. This allows for efficient plotting of data stored in these libraries' data structures, making data visualization tasks more straightforward.

To install, write this in command prompt :

* Pip install matplotlib

#### **3.2.1.4. NumPy**

NumPy is a fundamental numerical computing library in Python that provides efficient and versatile tools for working with large multi-dimensional arrays and matrices. It is a fundamental building block for numerous other libraries in the scientific computing ecosystem.

Some key features of NumPy:

1. Multi-dimensional Arrays: NumPy's main feature is its N-dimensional array object, called ndarray. It allows efficient storage and manipulation of homogeneous data, such as numbers. Ndarrays can have any number of dimensions and are highly optimized for numerical operations.
2. Mathematical Functions: NumPy provides a wide range of mathematical functions that can be applied element-wise to arrays. These functions include basic operations (addition, subtraction, multiplication, division), trigonometric functions, exponential and logarithmic functions, and more. NumPy's functions are optimized for performance and can be applied efficiently to large arrays.
3. Array Operations: NumPy supports various array operations, such as reshaping, slicing, and indexing. These operations enable efficient data manipulation and extraction of specific elements or subsets of data from arrays.
4. Integration with Other Libraries: NumPy seamlessly integrates with other scientific computing libraries, such as SciPy, Pandas, and Matplotlib. This interoperability enables efficient data manipulation, analysis, and visualization workflows.

To install, write this in command prompt :

* Pip install Numpy

#### **3.2.1.5. Seaborn**

Seaborn is a Python data visualization library that is built on top of Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn is particularly useful for visualizing complex datasets and exploring relationships between variables.

Some key features of Seaborn:

1. Easy-to-use API: Seaborn offers a simple and intuitive API for creating a variety of statistical plots. It provides a set of high-level functions that abstract away many of the complexities of plotting. With just a few lines of code, you can generate visually appealing and informative plots.
2. Integration with Pandas: Seaborn integrates seamlessly with Pandas, another popular Python library for data analysis. You can directly pass Pandas DataFrames to Seaborn functions, making it convenient to work with data stored in Pandas data structures.
3. Statistical Visualizations: Seaborn specializes in statistical visualizations, allowing you to create a wide range of plots to explore relationships and patterns in your data. It offers functions for creating scatter plots, line plots, bar plots, histograms, box plots, violin plots, heatmaps, and more. These plots often include additional statistical annotations or summarization, providing insights into the data distribution and relationships.

To install, write this in command prompt :

* pip install seaborn

#### **3.2.1.6. Cv2**

OpenCV (Open Source Computer Vision) or cv2 is a popular open-source computer vision library in Python. It provides a wide range of functions and tools for image and video processing, as well as various computer vision tasks.

Some key features of OpenCV (cv2):

1. Image and Video I/O: OpenCV allows you to read, write, and manipulate images and videos in various formats. It supports common image formats like JPEG, PNG, and TIFF, as well as video formats such as AVI and MP4. OpenCV provides functions for capturing video from cameras and processing individual frames.
2. Image Processing: OpenCV offers a comprehensive set of image processing functions, including resizing, cropping, rotating, filtering, and blending images. These functions enable tasks like image enhancement, noise reduction, edge detection, and feature extraction.
3. GUI and Visualization: OpenCV includes GUI components for creating graphical user interfaces. It provides tools for displaying images, drawing shapes and annotations, and interacting with the user. These features are useful for building interactive computer vision applications.
4. Performance Optimization: OpenCV is highly optimized for performance and provides support for parallel processing and hardware acceleration. It leverages features like multi-threading and GPU acceleration to speed up computations and improve efficiency.

To install, write this in command prompt :

* pip install opencv-python

## **3.3 LOAD THE DATASET**

### **3.3.1 MNIST Dataset**

The MNIST (Modified National Institute of Standards and Technology) dataset is a widely used benchmark in the field of machine learning. It consists of a collection of handwritten digit images along with their corresponding labels. The MNIST model is a popular example of a deep learning model used to classify these digits accurately. MNIST consists of a large collection of handwritten digits from 0 to 9. The dataset contains a training set of 60,000 grayscale images and a test set of 10,000 grayscale images. Each image is a 28x28 pixel square, representing a handwritten digit. The digits are centered and normalized, making the dataset relatively clean and consistent.



3 Fig 3.2 MNIST Dataset

The dataset consists of a total of 60,000 images that are used for training the model, and a subset of these images can also be utilized for cross-validation purposes. Additionally, there are 10,000 images specifically allocated for testing the model's performance. All the images in the dataset are grayscale and have a fixed size of 28x28 pixels. The intensity of each pixel is positioned at the center of the image. Therefore, each image can be represented as a 784-dimensional vector by flattening the 28x28 array. Each component of the vector represents a binary value indicating the pixel intensity.

**0**

**1**

**2**

**3**

**4**

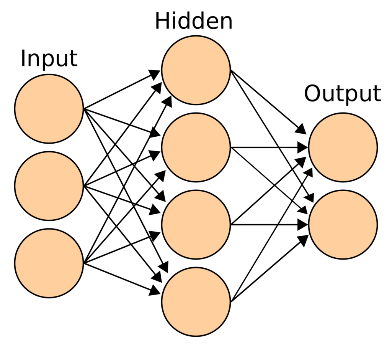
**5**

**6**

**7**

**8**

**9**



Neural Networks Layers

Data & Labels

4 Fig. 3.3 MNIST data processing classification with output

### **3.3.2 Data Cleaning**

Data cleaning, also known as data preprocessing, is a crucial step in preparing a dataset for analysis or model training. It involves identifying and handling various issues or inconsistencies in the data to ensure its quality and reliability.

Data cleaning may involve the following steps:

1. Handling Missing Values: Check if there are any missing values in the dataset. In the case of MNIST, since it is a well-curated and widely used dataset, missing values are unlikely. However, if there were any missing values, appropriate techniques such as imputation or removal of affected instances would be applied.
2. Outlier Detection: Identify and handle any outliers that might exist in the dataset. Outliers are data points that significantly deviate from the normal distribution. In the case of MNIST, outliers are less likely due to the standardized format of the dataset. However, if any outliers are present, they could be treated through techniques like trimming, winsorization, or removal.

### **3.3.3 Data Transformation**

Data transformation refers to the process of converting or manipulating the raw data in a dataset to make it more suitable for analysis or modeling. It involves applying various mathematical or statistical operations to modify the data values or distribution. Data transformation can help improve the performance of statistical models, ensure data compatibility, handle outliers, or meet assumptions of specific analysis techniques.

* Normalization: Also known as feature scaling, normalization rescales the data to a common scale. It ensures that all features have similar ranges and prevents one feature from dominating others. Common normalization techniques include Min-Max scaling and Z-score normalization.
* Standardization: Similar to normalization, standardization transforms the data to have zero mean and unit variance. It subtracts the mean from each data point and divides it by the standard deviation. Standardization is useful when the data has a non-Gaussian distribution or when the model algorithm assumes standardized features.

## **3.4 MODEL CONSTRUCTION**

Now, comes the very fun stage where we finally get to work on the prepared data that we processed earlier for model building. Model construction, also known as model building or model development, refers to the process of creating a predictive or descriptive model using a given dataset. It involves selecting an appropriate algorithm or method, training the model on the available data, and evaluating its performance.

Depending upon the data type of the target variable (find the digit form the given data, commonly referred as ‘y’) from the dataset, their comes a construction of classification model or a regression model.

The model construction phase would typically involve the following steps:

* Algorithm Selection: Choose a suitable deep learning algorithm or architecture for digit recognition. In this case, you have mentioned using a Convolutional Neural Network (CNN) model, which is a widely used architecture for image recognition tasks.
* Data Preparation: Preprocess the MNIST dataset to ensure it is in a suitable format for training the model. This includes tasks such as resizing the images to a consistent size (e.g., 28x28 pixels in your case), normalizing or standardizing the pixel values, and splitting the dataset into training, validation, and testing sets.

### **3.4.1 Algorithms**

Machine Learning algorithms are categorized in different types :

* **Supervised Learning:**

Supervised learning algorithms learn from labeled training data, where the input features and their corresponding output labels or target values are provided. The goal is to learn a mapping function that can predict the output labels for new, unseen data. Some common supervised learning algorithms include linear regression, logistic regression, decision trees, random forests, support vector machines (SVM), and neural networks. Supervised learning is used for tasks such as classification (predicting categorical labels) and regression (predicting continuous values).

**Classification :** In classification, the target variable or output label is a predefined set of classes or categories. The algorithm learns from labeled training data, where each data point is associated with a known class label. The classifier then uses the learned patterns to classify new data points into one of the predefined classes.

**Example :** Email spam detection (classifying emails as spam or not), sentiment analysis (classifying text as positive or negative), and image recognition (identifying objects in images).

**Regression :** In regression, the target variable or output label is a continuous variable. The algorithm learns from labeled training data, where each data point is associated with a known numerical value. The regression model then uses the learned patterns to predict the numerical output for new, unseen data points.

**Example :** predicting housing prices based on features like area and number of bedrooms, forecasting stock prices, and estimating sales based on advertising expenditure.

* **Unsupervised Learning:**

Unsupervised learning algorithms work with unlabeled data, where only the input features are provided without any corresponding output labels. The objective of unsupervised learning is to discover patterns, structures, or relationships within the data. It does not involve predicting specific target values. Common unsupervised learning algorithms include clustering algorithms like k-means clustering and hierarchical clustering, dimensionality reduction techniques like principal component analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE), and association rule mining algorithms. Unsupervised learning is useful for tasks such as data exploration, grouping similar data points, and anomaly detection.

**Clustering** : It involves grouping similar data points together based on their characteristics or features without any predefined labels or target variables. The objective of clustering is to find meaningful structures or clusters in the data, where data points within the same cluster are more similar to each other than to those in other clusters.

* **Reinforcement Learning**

Reinforcement learning is a type of machine learning that focuses on decision-making and learning through interactions with an environment. It is inspired by how humans and animals learn to perform certain actions or behaviors based on the feedback they receive from the environment.

In reinforcement learning, an agent learns to make sequential decisions in an environment to maximize a notion of cumulative reward. The agent interacts with the environment by taking actions, and the environment provides feedback in the form of rewards or penalties based on the agent's actions. The goal of the agent is to learn a policy, which is a mapping from states to actions, that maximizes the expected cumulative reward over time.

### **3.4.2 Model That Can Be Used for This Project**

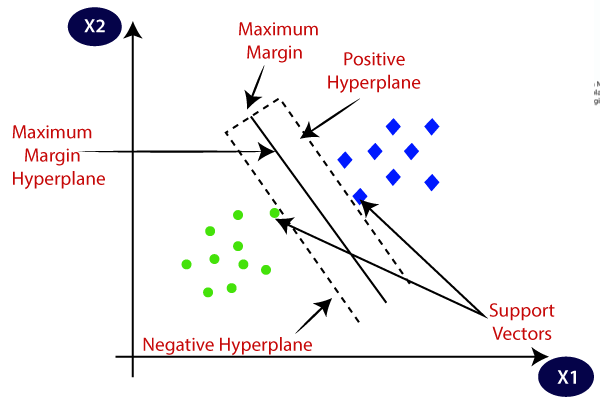
1. **SUPPORT VECTOR MACHINE (SVM):**

Support Vector Machines (SVM) is a supervised machine learning algorithm that can be used for classification and regression tasks. In the context of the Handwritten Digit Recognition project, SVM can be utilized as a classifier to identify and classify the handwritten digits.

The objective of the SVM algorithm is to construct an optimal hyperplane, which serves as a decision boundary, to effectively separate classes in an n-dimensional space. By creating this boundary, the SVM algorithm enables accurate categorization of new data points into the appropriate class category in the future.

The SVM algorithm selects the most crucial points or vectors, known as support vectors, to construct the hyperplane. This choice of support vectors is what gives the algorithm its name, Support Vector Machine. SVM is a versatile algorithm that can be applied to various tasks such as face detection, image classification, text categorization, and more. It is particularly effective in scenarios where accurate boundary separation is crucial for accurate classification or identification.

**Working of SVM:**



5 Fig. 3.4 Working of SVM on a dataset

The distance between the decision–boundary and the closest data points is called the margin.

Important concepts in Support Vector Machines (SVM) that are key to understanding and effectively using the algorithm:

1. Hyperplane:

A hyperplane is a decision boundary that separates the data points into different classes. In SVM, the goal is to find the optimal hyperplane that maximizes the margin between the support vectors of different classes.

1. Support Vectors:

Support vectors are the data points that lie closest to the decision boundary. These points have the most influence on determining the position and orientation of the hyperplane. SVM algorithm only considers these support vectors in the training process, making it memory-efficient.

1. Margin:

The margin is the distance between the decision boundary (hyperplane) and the nearest support vectors from each class. The optimal hyperplane is the one that maximizes this margin. A larger margin implies better generalization and robustness of the classifier.

The primary objective of SVM is to separate datasets into classes by identifying the maximum margin hyperplane (MMH). This can be achieved through the following two steps:

* Initially, SVM generates hyperplanes in an iterative manner to best segregate the classes.
* Subsequently, it selects the hyperplane that effectively separates the classes with the highest accuracy.

In summary, SVM aims to find the optimal hyperplane that maximizes the margin between classes, ensuring accurate classification.

**Pros of SVM:**

1. Effective in high-dimensional spaces: SVM performs well even in cases where the number of features exceeds the number of samples. This makes it suitable for tasks involving large feature spaces, such as image classification or text categorization.

2. Robust to outliers: SVM is less affected by outliers in the training data due to its reliance on support vectors. It focuses on the critical data points near the decision boundary, making it more resistant to noise and outliers.

3. Versatile kernel functions: SVM offers various kernel functions (linear, polynomial, radial basis function, etc.) to handle both linearly separable and nonlinearly separable data. This flexibility allows for complex decision boundaries and improved classification accuracy.

4. Control over margin and regularization: The C parameter in SVM provides control over the balance between maximizing the margin and minimizing classification errors. This allows for fine-tuning the model based on the specific requirements of the problem.

**Cons of SVM:**

1. Computationally intensive: SVM can be computationally expensive, particularly when dealing with large datasets. Training time can increase significantly as the number of samples and features grow.

2. Difficult to interpret: The decision boundaries produced by SVM may be challenging to interpret and visualize, especially in high-dimensional spaces. Understanding the underlying factors driving the classification may not be straightforward.

3. Sensitivity to parameter tuning: SVM performance is sensitive to the selection of hyperparameters, such as the choice of kernel function and regularization parameter. Suboptimal parameter settings may lead to subpar performance or overfitting.

4. Memory usage: SVM stores a subset of the training data (support vectors) for prediction, which can be memory-intensive, especially when dealing with large datasets. Memory usage scales with the number of support vectors.

It's worth noting that while SVM has its advantages and disadvantages, the suitability of the algorithm depends on the specific problem, dataset characteristics, and the availability of computational resources. Experimentation and comparison with other algorithms are recommended to determine the best approach for a particular task.

**2. K-NN ALGORITHM:**

* The K-Nearest Neighbour (K-NN) algorithm is a straightforward supervised learning technique in the field of Machine Learning.
* K-NN algorithm works on the assumption of similarity between new data and existing data points, placing the new data into the category that bears the closest resemblance.
* The algorithm stores all available data and classifies new data points based on their similarity to existing data. In this way, when new data is encountered, it can be easily classified into an appropriate category using the K-NN algorithm.
* K-NN algorithm can be applied to both regression and classification problems, although it is primarily used for classification tasks.

K-NN is categorized as a non-parametric algorithm, indicating that it does not make any assumptions about the underlying data distribution. Moreover, it is referred to as a lazy learner since it does not immediately learn from the training set. Instead, it stores the dataset and performs actions on it during the classification phase.

During the training phase, the K-NN algorithm simply retains the dataset. When presented with new data, it classifies it into a category that closely resembles the new data. This process of classification is based on the similarity between the new data and the stored dataset.

To implement K-Nearest Neighbors (KNN) algorithm from scratch, follow these steps:

* Compute the Euclidean distance between the test data point and all the training data points.
* Sort the calculated distances in ascending order.
* Select the top k rows from the sorted array, which represent the k nearest neighbors.
* Determine the majority class among these k nearest neighbors.
* Return the predicted class as the output.
* To evaluate the accuracy of the predictions, compare the predicted class with the actual class labels and calculate the accuracy score.

1)Calculation of Euclidean distance:

Euclidean distance is the square root of the sum of squared distance between two points.

**def** Euclidean\_distance(row1, row2):  
distance = 0  
**for** i **in** **range**(**len**(row1)-1):  
distance += (row1[i] — row2[i])\*\*2  
**return** sqrt(distance)

* Two points are passed (here row1 &row2)
* The Euclidean \_distance function calculates the difference between the squares of the points and finally the square root of the difference.

2)Get the k nearest neighbors after sorting distance

**def** Get\_Neighbors(train, test\_row, num):  
   
 distance = list() # []  
 data = []  
 **for** i **in** train:  
 dist = Euclidean\_distance(test\_row, i)  
 distance.append(dist)  
 data.append(i)  
 distance = np.array(distance)  
 data = np.array(data) #Finding the index in ascending order  
 index\_dist = distance.argsort() #Arranging data according to index  
 data = data[index\_dist] #slicing k value from number of data  
 neighbors = data[:num]  
   
 **return** neighbors

* In order to find the neighbors we need to first sort the distance in ascending order,np.argsort () is used to find the index of minimum distance .
* After that we will arrange the data according to the sorted index.
* Slicing the data according to the number of neighbors.

3)Predicting the class of the new data point

**def** predict\_classification(train, test\_row, num):  
Neighbors = Get\_Neighbors(train, test\_row, num)  
Classes = []  
**for** i **in** Neighbors:  
Classes.append(i[-1])  
prediction = **max**(Classes, key= Classes.count)  
**return** prediction

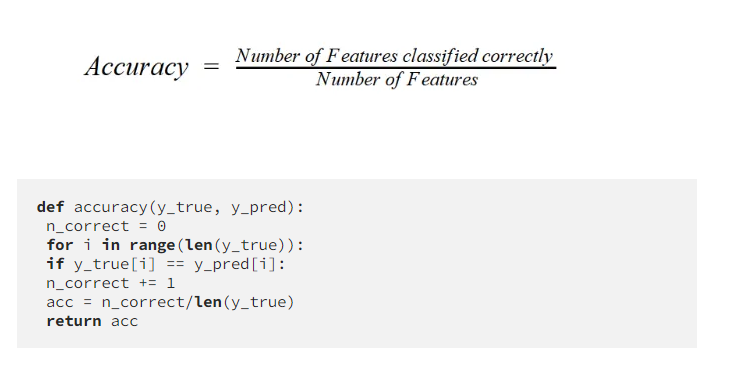
The test data will present in the class with majority of the votes. So, to find that we will use max () function

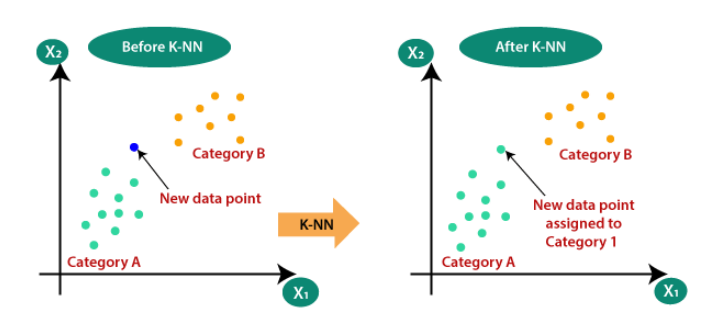
They key in the max function groups the neighbors w.r.t. to their classes and .count will count the number of neighbors in each class.

Finally max returns the class with majority votes which will be the predicted class of the test data.

4)Accuracy calculation

Accuracy shows how close the measured value to the true value.



* Accuracy is calculated by dividing the correctly classified samples count by total samples.
* Higher the accuracy, more efficient the model.

6 Fig. 3.5 KNN Model

**Building KNN Project on Handwritten Digit Recognition**

1. **Import the libraries**

from sklearn.datasets import fetch\_openml, load\_digits  
import numpy as np  
import pandas as pd   
import matplotlib.pyplot as plt  
import matplotlib

1. **Loading the dataset**

It may take time to load as it is huge dataset using fetch\_openml

Also we can use small dataset by using load\_digits() function.

mnist = fetch\_openml(“mnist\_784”) #load\_digits()

1. **Preprocessing of the data**

The preprocessing steps include converting the data into standard formats, determining the independent and target variables, and shuffling the data using the np.random.permutation() function.

By performing these preprocessing steps, the data becomes suitable for training and testing machine learning models in the handwritten digit recognition project.

# checking the column names and preprocessing target values in standard format  
mnist.**keys**()  
mnist.target = mnist.target.**astype**(np.int8)#Determining independent and dependent variable and finding the shape  
x = **np.array**(mnist.data)  
y = **np.array**(mnist.target)  
x.**shape**, y.shape  
#output (**(70000, 784), (70000,))**# shuffling the values of x and y  
si = **np.random.permutation**(x.shape[0])  
x = x[si]  
y = y[si]

1. **Visualization of data**

Visualization requires reshaping of 784 columns into 28\*28 pixel image because all the data must be normalized within a same range of 0 – 1.

For plotting, plt.imshow() used from matplotlib library.

some\_digit=x[12]  
some\_digit\_image=some\_digit.reshape(28,28)  
plt.imshow(some\_digit\_image,cmap=matplotlib.cm.binary)  
plt.axis(“off”)  
plt.show()

1. **Prediction of the model**

Due to the bulkiness of the dataset, I am training only for 2000 samples . So, we are slicing the dataset to 2000 values.

Inserting the y in x using np.insert()

Predict using the prediction function defined above in KNN.

Return prediction

#slicing data  
trainx = x[:2000]  
trainy = y[:2000]#Inserting trainy in trainx  
train = np.insert(trainx, 784, trainy, axis = 1)prediction = predict\_classification(train, train[1244], 4)  
prediction#Output **8.0**

train[1244][-1]  
#output **8.0**

#Plotting the output  
some\_digit = train[1244][:-1]  
some\_digit\_image = some\_digit.reshape(28, 28)  
plt.imshow(some\_digit\_image, cmap=matplotlib.cm.binary)  
plt.axis(“off”)  
plt.show()

7. Fig. 3.6 output

1. **Accuracy**

Storing true values and predicted values

Use accuracy function above to calculate the accuracy score.

y\_pred=[]

y\_true=train[:,-1]

for i in range(len(train)):

prediction = predict\_classification(train, train[i], 4)

y\_pred.append(prediction)

# Accuracy

accuracy(y\_true, y\_pred)

>>> Output 0.929 (i.e., 92%)

The probability of the true class is classified correctly is 0.929.

**Advantages of K-Nearest Neighbors (KNN) algorithm:**

1. Simple and easy to understand: KNN is a simple algorithm that is easy to implement and interpret. It does not require complex mathematical calculations or assumptions.

2. No training phase: KNN is a lazy learning algorithm, meaning it does not explicitly train a model. It stores the entire training dataset and makes predictions based on similarity measures at runtime.

3. Flexibility in handling different data types: KNN can handle both numerical and categorical data, making it versatile for various types of datasets.

4. Non-parametric nature: KNN does not make assumptions about the underlying data distribution. It can effectively handle complex decision boundaries and nonlinear relationships.

5. Works well with small datasets: KNN can perform well when the dataset size is small, as it relies on comparing distances between data points.

**Disadvantages of K-Nearest Neighbors (KNN) algorithm:**

1. Computational complexity: As the number of training samples increases, the computational cost of KNN grows significantly. Finding the nearest neighbors for each prediction can be time-consuming, especially with large datasets.

2. Sensitivity to feature scaling: KNN uses distance metrics to calculate similarity. If the features have different scales, the ones with larger scales may dominate the distance calculations, leading to biased results. Feature scaling is necessary to avoid this issue.

3. Memory usage: KNN stores the entire training dataset in memory, which can be memory-intensive for large datasets. This limits its scalability in terms of handling big data.

4. Not suitable for high-dimensional data: KNN's performance tends to degrade as the number of dimensions/features increases. This is known as the "curse of dimensionality," where the distance between points becomes less meaningful in high-dimensional spaces.

5. Choosing the optimal value of k: The selection of the optimal number of neighbors (k) is crucial in KNN. A small value of k may lead to overfitting, while a large value may result in underfitting. Finding the right balance is essential.

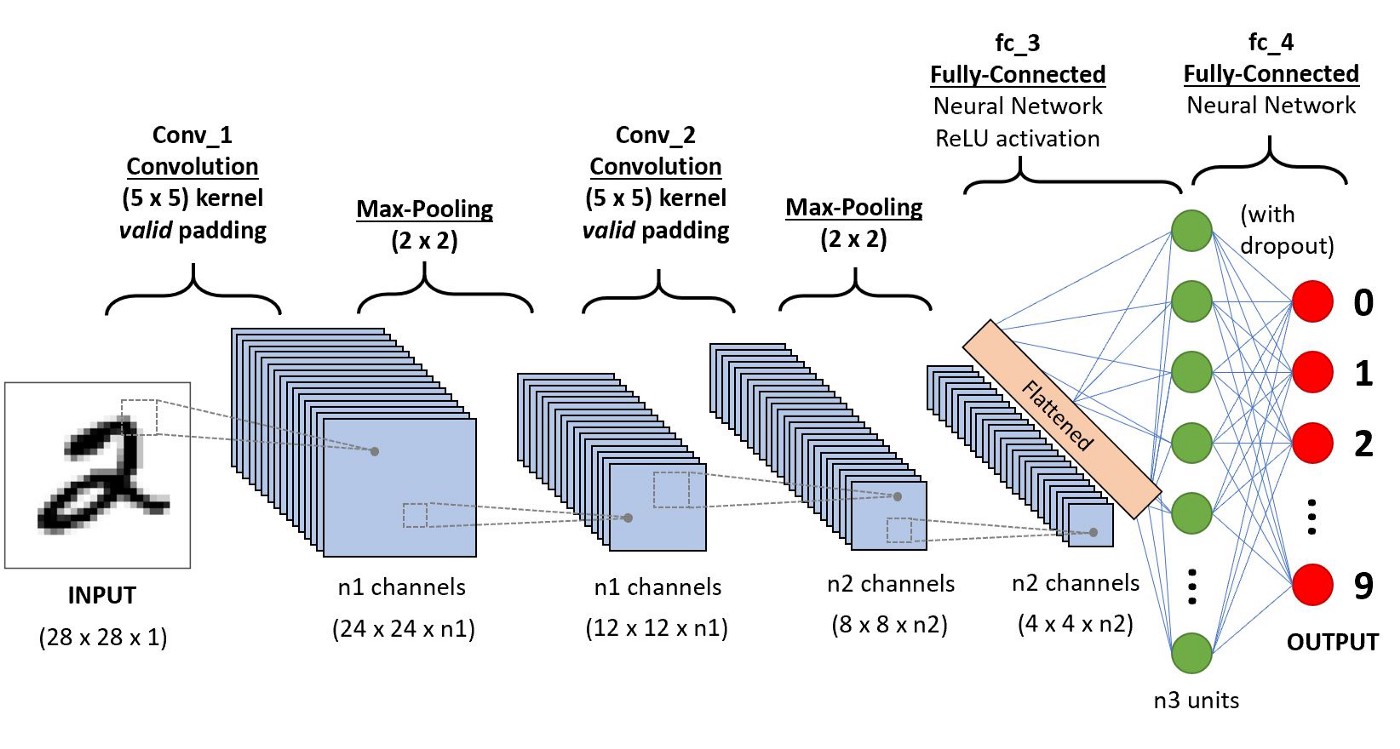
**3. CONVOLUTIONAL NEURAL NETWORK**

In simpler terms, a Convolutional Neural Network (CNN) is a type of artificial neural network that is specifically designed to identify patterns in data, particularly images. It achieves this by using different filters of various sizes and quantities to detect these patterns. The main component of a CNN is the convolutional layer, which performs a process called convolution.

Convolution refers to the mathematical operation performed on the input data using convolutional filters. It is a fundamental operation in CNNs that helps extract features from the input data. Convolution involves sliding a small matrix, called a kernel or filter, over the input data. At each position, the kernel performs element-wise multiplication with the corresponding input values within its receptive field, followed by summing up the results. This process generates a single value, which represents a feature or characteristic of the input data.

The purpose of convolution in CNNs is to capture local patterns and spatial relationships within the data. By applying multiple convolutional filters, the network can learn to detect various features at different levels of abstraction. These features can include edges, textures, shapes, or higher-level representations specific to the task at hand, such as object parts or structures.

A typical CNN model consists of convolutional layers, pooling layers, and a dropout layer. It is especially effective for analyzing data arranged in grid-like structures, which is why it is commonly used for image classification tasks. The dropout layer helps prevent overfitting by deactivating some neurons during training.

In my project, I employed a CNN model that involves feature extraction using convolution and binary classification. The convolution and max pooling operations are applied to extract features from the image. Specifically, we use 32 3x3 convolution filters on a 28x28 image, followed by a max pooling layer with a pooling size of 2x2. Another convolution layer is then applied using 64 3x3 filters.

8. Fig. 3.7 Working of CNN

After applying convolution and pooling operations in the convolutional layers, we obtain 7x7 images as the output. These 7x7 images are then flattened into a series of 128 values. The flattened values are then fed into a dense layer consisting of 128 neurons. These neurons are connected to the final output layer, which consists of 10 neurons representing the categorical output.

In the convolutional layers, a smaller filter is used compared to the input data. The operation applied between a filter-sized patch of the input and the filter is a dot product, which involves element-wise multiplication and subsequent summation to obtain a single value. This dot product operation is sometimes referred to as the "scalar product."

Using a smaller filter size allows the same set of weights (filter) to be multiplied by different parts of the input array. This process is applied systematically to each overlapping part or filter-sized patch of the input data, starting from the top-left corner and moving left to right, top to bottom.

Overall, this approach of using smaller filters and applying them systematically allows the network to learn and extract features from different regions of the input data, leading to the development of meaningful representations and improved performance in the task at hand.

**Working of CNN in Hand Digit Recognition:**

1. Convolutional Layers:

* The input images of handwritten digits are passed through convolutional layers.
* Each convolutional layer consists of multiple filters (also known as kernels), which are small matrices.
* Each filter is applied to the input image using the convolution operation.
* The convolution operation involves sliding the filter over the input image and computing the dot product of the filter weights and the corresponding pixels in the input.
* The result is a feature map that highlights different patterns and features in the input image.
* Multiple filters in each convolutional layer capture different features at different levels of abstraction.

1. Activation Function:

* After the convolution operation, an activation function (usually ReLU - Rectified Linear Unit) is applied element-wise to the feature map.
* The activation function introduces non-linearity and helps in capturing complex patterns and non-linear relationships in the data.

1. Pooling Layers:

* The feature maps are passed through pooling layers, which reduce the spatial dimensions of the data.
* The most commonly used pooling operation is max pooling, where the maximum value within a small region (e.g., 2x2) of the feature map is selected.
* Pooling helps in reducing the computational complexity, controlling overfitting, and extracting the most important features.

1. Flattening:

* After several convolutional and pooling layers, the feature maps are flattened into a one-dimensional vector.
* This flattening process converts the spatial information into a format suitable for feeding into a fully connected neural network.

1. Fully Connected Layers:

* The flattened vector is then connected to one or more fully connected layers, also known as dense layers.
* Dense layers consist of neurons that are fully connected to the previous layer.
* These layers learn complex combinations of features from the flattened input vector.

1. Output Layer:

* The final dense layer is connected to the output layer, which has neurons equal to the number of classes (0-9 in the case of hand digit recognition).
* The output layer uses an appropriate activation function (e.g., softmax for multi-class classification) to produce class probabilities.

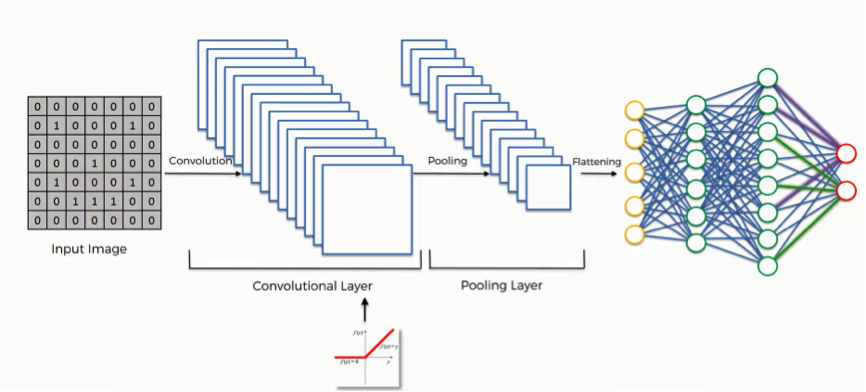
1. Training and Optimization:

* The CNN is trained using labeled training data, where the network learns to minimize a loss function (e.g., cross-entropy loss) by adjusting the weights through backpropagation and gradient descent.
* Optimization techniques like stochastic gradient descent (SGD) or its variants are commonly used to update the network weights.

1. Inference:

* Once the CNN is trained, it can be used for inference on unseen test data.
* The input test images go through the trained network, and the output layer provides the predicted class probabilities.
* The class with the highest probability is considered as the predicted digit.

By leveraging the hierarchical structure of convolutional layers, along with pooling and dense layers, CNNs are able to automatically learn and extract relevant features from the input images, leading to accurate hand digit recognition.



9. Fig. 3.8 Working of CNN with HRD

## **3.5 TRAINING AND VALIDATION**

After constructing the model, it needs to be compiled in order to train it using the available dataset. The compilation of the model involves specifying the optimizer, loss function, and metrics. Optimizers are algorithms used to adjust the attributes of the neural network, such as weights and learning rate, to minimize the losses and solve optimization problems.

In our case, I will use the 'adam' optimizer, which is a commonly used optimizer that dynamically adjusts the learning rate during training. The learning rate determines the speed at which the optimal weights for the model are calculated. A smaller learning rate can result in more accurate weights, but it may also increase the computation time.

For the loss function, I will use 'categorical\_crossentropy', which is a commonly used loss function for classification tasks. A lower score in this function indicates better performance of the model. Additionally, we will use the 'accuracy' metric to evaluate the model's performance on the validation set during training.

The goal of training and testing a data model is to achieve a high learning rate and maximize validation. By increasing the size of the training and testing datasets, I can improve the model's learning rate and validation.

Once the model is properly assembled and compiled, we can proceed to train it using the training data. In our case, I will train the model for 100 iterations, but it is important to be cautious of overfitting as the number of iterations increases. To mitigate this, we limit the training process to achieve at least 98% accuracy, as we are using real-world data for prediction and the test data is used to validate the model's performance.

Different optimizers used in Neural Networks are:

1. Gradient Descent

2. Stochastic Gradient Descent (SGD)

3. Mini Batch Stochastic Gradient Descent (MB-SGD)

4. SGD with momentum

5. Nesterov Accelerated Gradient (NAG)

6. Adaptive Gradient (AdaGrad)

7. AdaDelta

8. RMSprop

9. Adam

## **3.5.1 Adam Optimizer**

Adaptive Moment Estimation (Adam) is a highly efficient optimization algorithm used in gradient descent. It is particularly beneficial for large-scale problems involving a significant amount of data or parameters. Adam combines the strengths of the "gradient descent with momentum" and "RMSP" algorithms, resulting in improved performance and memory efficiency. This optimization algorithm has gained popularity in deep learning applications, especially in computer vision and natural language processing. Adam offers an alternative to traditional stochastic gradient descent for updating network weights during iterative training. It was introduced in a 2015 paper titled "Adam: A Method for Stochastic Optimization" by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto.

According to the authors, the Adam optimization algorithm combines the benefits of two other extensions of stochastic gradient descent: AdaGrad and RMSProp.

* **Adaptive Gradient Algorithm (AdaGrad)**

The Adaptive Gradient Algorithm (AdaGrad) is an optimization technique that adjusts the learning rate for each parameter in a neural network based on the magnitude of its gradients. It is particularly useful for problems with sparse gradients, such as those encountered in natural language processing and computer vision tasks.

The main advantage of AdaGrad is its ability to give larger updates to parameters with smaller gradients and vice versa. This adaptive learning rate scheme helps to converge faster by allocating more attention to parameters that are updated less frequently.

However, the AdaGrad method has a limitation. Since it accumulates the squared gradients over time, the learning rates may become too small, making it difficult for the algorithm to make further progress in training.

To address this limitation, the Adaptive Moment Estimation (Adam) algorithm was introduced. Adam combines the benefits of AdaGrad with the concept of momentum. In addition to keeping track of the exponentially decaying average of past squared gradients (vt) like Adadelta and RMSprop, Adam also maintains an exponentially decaying average of past gradients (mt), similar to the momentum technique.

By incorporating both the squared gradients and the past gradients, Adam provides a more comprehensive picture of the parameter updates. This enables the algorithm to adaptively adjust the learning rates for each parameter, considering both the magnitude and direction of the gradients.

The combination of these techniques makes Adam a popular choice in deep learning applications. It offers efficient and adaptive learning rates, allowing for effective optimization of neural networks on various tasks, including natural language processing and computer vision.

* **The Root Mean Square Propagation (RMSProp)**

RMSprop is an optimization algorithm that adjusts the learning rates for each parameter in a neural network based on the average of recent magnitudes of the gradients. It is particularly effective for handling online and non-stationary problems, such as those characterized by noisy data.

The main idea behind RMSProp is to adaptively update the learning rates by taking into account the history of gradient magnitudes. It calculates the moving average of the squared gradients for each weight parameter. By averaging the magnitudes of recent gradients, RMSProp provides a measure of how quickly the weight is changing and adjusts the learning rate accordingly.

The advantage of using the average of gradient magnitudes is that it helps stabilize the learning process, especially in scenarios where the gradients exhibit large fluctuations or noise. The algorithm is less sensitive to noisy updates, allowing it to perform well in online learning settings where data is presented sequentially.

RMSProp addresses the limitation of AdaGrad, which accumulates squared gradients over time and may result in diminishing learning rates. By using a moving average instead of the accumulated sum, RMSProp ensures that the learning rates remain adaptive and do not become overly small during training.

By adaptively adjusting the learning rates based on the recent history of gradient magnitudes, RMSProp enables the optimization algorithm to converge faster and more reliably. It helps navigate through complex optimization landscapes, where different parameters may require different learning rates to achieve optimal performance.

In summary, RMSProp is a powerful optimization algorithm that maintains per-parameter learning rates based on the average of recent gradient magnitudes. This adaptive learning rate scheme allows for effective training in online and non-stationary problems, enhancing the stability and convergence of neural networks in the presence of noise or varying data distributions.

**Properties of Adam:**

The Adam optimization algorithm possesses several key properties that contribute to its effectiveness and popularity:

**1.** Bounded Step Size: The actual step size taken by Adam in each iteration is approximately bounded by the step size hyperparameter. This property provides an intuitive understanding of the learning rate hyperparameter, allowing for better control and fine-tuning of the optimization process.

**2.** Invariance to Gradient Magnitude: The step size of the Adam update rule is invariant to the magnitude of the gradient. This property is particularly beneficial when traversing areas with tiny gradients, such as saddle points or ravines. Unlike stochastic gradient descent (SGD), which struggles to navigate through such areas, Adam's step size adaptation enables it to efficiently traverse these regions without getting stuck.

**3.** Combination of Adagrad and RMSprop Advantages: Adam was specifically designed to combine the advantages of two other optimization algorithms—Adagrad and RMSprop. Adagrad performs well with sparse gradients, while RMSprop excels in online settings. By incorporating both of these advantages, Adam becomes a versatile optimization algorithm suitable for a broader range of tasks.

**4.** RMSprop and SGD with Momentum Combination: Another perspective on Adam is that it combines the properties of RMSprop and SGD with momentum. RMSprop provides adaptive learning rates based on the average of recent gradient magnitudes, while SGD with momentum helps navigate through optimization landscapes by incorporating momentum. The combination of these two components in Adam further enhances its optimization capabilities.

**3.5.2 WHY ADAM?**

Adam is a suitable optimization algorithm for digit recognition tasks due to the following reasons:

1. Improved Optimization: Adam is known to outperform classical stochastic gradient descent (SGD) in many cases. It combines the advantages of two popular algorithms, AdaGrad and RMSProp, to provide efficient optimization. This makes it well-suited for training neural networks used in digit recognition tasks.

2. Handling Sparse Gradients: Digit recognition problems often involve sparse gradients, especially when dealing with high-dimensional image data. Adam's ability to handle sparse gradients, inherited from AdaGrad, allows it to effectively learn from the data and update the network weights accordingly.

3. Adaptability to Noisy Problems: Noisy data is a common challenge in digit recognition, as images may contain variations, distortions, or imperfections. Adam's incorporation of RMSProp's adaptive learning rates enables it to handle noisy problems and converge to good solutions even in the presence of such variations.

4. Ease of Configuration: Adam has default configuration parameters that are generally suitable for a wide range of problems, including digit recognition. This means that even without extensive parameter tuning, Adam can still perform well in most cases. This simplicity of configuration makes it an attractive choice for practitioners.

## **3.6 MODEL EVALUATION & PREDICTION:**

In order to make predictions on real-world images for image classification, some image pre-processing steps are required since the model was trained on grayscale raster images. The pre-processing steps involve:

1. Loading the image.

2. Converting the image to grayscale.

3. Resizing the image to a size of 28x28 pixels.

4. Converting the image into a matrix representation.

5. Reshaping the matrix into a shape of 28x28x1.

After pre-processing the image, we can pass it through the trained neural network to predict its label. The output of the network is a list of 10 activation values, representing the probabilities for each possible label (0 to 9). The predicted label for the image is determined by selecting the position with the highest activation value.

These neural network structures are commonly known as Convolutional Neural Networks (CNNs), which have greatly contributed to the field of computer vision and image analysis. CNNs are specialized deep learning models designed for analyzing visual images and have found applications in various domains such as image and video recognition, image classification, medical image analysis, computer vision, and natural language processing.

When it comes to evaluating the performance of a model, there are two main methods: holdout and cross-validation. Both methods involve using a separate test set, which consists of data that the model has not seen during training, to assess the model's performance. It is not recommended to evaluate the model using the same data that was used for training because this can lead to overfitting, where the model memorizes the training set and performs poorly on unseen data.

Therefore, to ensure an unbiased evaluation, it is important to use separate test data for evaluating the model's performance.

In this approach, the dataset is divided randomly into three subsets:

1. Training set: It is a subset of the dataset used for building the predictive models. The training set is used to train the model by adjusting its parameters based on the available data.

2. Validation set: It is a subset of the dataset used to evaluate the performance of the model that was built during the training phase. The validation set provides a platform for fine-tuning the model's parameters and selecting the best-performing model. Not all modeling algorithms require a validation set, but it is commonly used in machine learning.

3. Test set: Also known as unseen data, the test set is a subset of the dataset used to assess the model's performance on new, unseen data. It serves as a measure of how well the model is likely to perform in real-world scenarios. If the model fits the training set significantly better than the test set, it may indicate overfitting, where the model is too specialized to the training data and performs poorly on new data.

By dividing the dataset into these subsets, we can train the model on the training set, fine-tune it using the validation set, and finally evaluate its performance on the test set to get an understanding of how well it generalizes to new, unseen data.

### **3.6.1 CROSS VALIDATION**

Cross-validation is a technique used for assessing the performance and generalization ability of a model. It involves dividing the original dataset into two main subsets: a training set and an independent set for evaluation.

The most commonly used cross-validation method is k-fold cross-validation. In this approach, the dataset is divided into k equal-sized subsamples or folds. Each fold is used as a validation set once, while the remaining k-1 folds are combined to form the training set. This process is repeated k times, with each fold serving as the validation set exactly once. The performance of the model is evaluated on each validation set, and the results are averaged to obtain an overall measure of the model's effectiveness.

The choice of k depends on the specific requirements and preferences. Typically, values of k such as 5 or 10 are commonly used. By repeating the training and evaluation process multiple times with different subsets, cross-validation provides a robust estimation of the model's performance and its ability to generalize to unseen data.

The advantage of using cross-validation is that it provides a more reliable assessment of the model's performance compared to a single train-test split. It helps to mitigate the potential bias introduced by a specific data split. Additionally, cross-validation allows for better tuning of model parameters and can provide insights into the model's stability and variability.

# **CHAPTER 4**

# **CNN ARCHITECTURE­­**

CNNs, or Convolutional Neural Networks, are a type of deep learning model specifically designed for analyzing visual images. They are widely used in various applications such as image recognition, image classification, medical image analysis, computer vision, and natural language processing.

The term "convolution" in CNN refers to a mathematical operation where two functions, in this case, images represented as matrices, are multiplied to generate a third function that captures the modified shape of one function by the other. This convolution operation plays a crucial role in extracting features from images.

In the context of CNN architecture, the input image undergoes a series of processing steps during training and testing. These steps include passing the image through convolution layers with filters (kernels), applying pooling operations, utilizing fully connected layers (FC), and applying a Softmax function to classify objects based on probabilistic values between 0 and 1. This complete flow of operations allows the CNN model to process the input image and make predictions about the objects present in the image.

Overall, CNNs are powerful models that leverage convolution operations and other specialized layers to effectively extract features from visual images and perform tasks such as classification and recognition.

The architecture of a CNN consists of two main components:

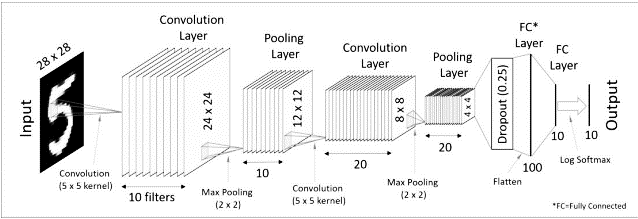
* Convolutional Layers: These layers are responsible for extracting and identifying different features of the image through a process called feature extraction. They apply convolution operations to the input image, enabling the network to capture important patterns and local relationships within the image.
* Fully Connected Layers: These layers utilize the output obtained from the convolutional layers and make predictions about the class or category of the image based on the features extracted in the earlier stages. The fully connected layers analyze the extracted features and map them to the appropriate class labels, allowing the network to classify the image accurately.

The CNN architecture consists of convolutional layers for feature extraction and fully connected layers for prediction and classification based on the extracted features. This combination of layers enables CNNs to effectively analyze and classify images.

## **4.1 CNN LAYERS**

The CNN architecture consists of multiple layers that work together to process and analyze the input data:

* Input Layer: The raw pixel values of the image are provided as input to the network.
* Convolutional Layer: This layer applies filters or kernels to the input data, sliding them over the image to extract important features. Each filter covers a small window of the input and captures the maximum intensity pixels.
* Rectified Linear Unit (ReLU) Layer: This layer applies an activation function to the output of the convolutional layer. ReLU function helps in preventing the pixel values from changing during the backpropagation process.
* Pooling Layer: This layer performs downsampling by reducing the dimensions of the input data (width and height). It helps in reducing the computational complexity and capturing the most important features.
* Fully Connected Layer: This layer focuses on classifying the input data by assigning a score to each class. The maximum score indicates the predicted class for the input image.

As the network goes deeper with more layers, the complexity of the model increases. This can potentially improve the accuracy of the predictions, but it also leads to increased computational time.

10. Fig. 4.1 CNN Architecture

### **4.1.1 Convolutional Layer**

The Convolutional Layer is an essential component of a Convolutional Neural Network (CNN) that plays a crucial role in feature extraction from input images. It involves performing a mathematical operation known as convolution between the input image and a filter (also referred to as a kernel or weight matrix).

The filter used in the convolutional layer has a specific size, typically represented as M x M, where M refers to the dimensions of the filter. The purpose of the filter is to capture and detect various visual patterns or features present in the input image.

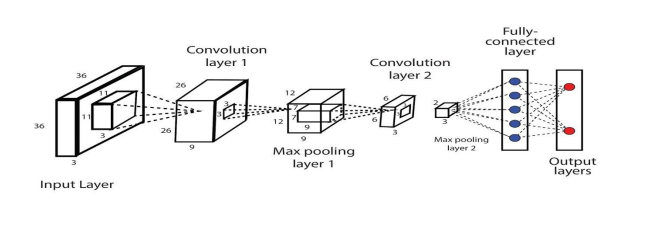
During the convolution process, the filter is systematically applied to different parts of the input image by sliding it across the image. At each position, a dot product operation is performed between the filter and the corresponding pixels of the input image covered by the filter (based on the size of the filter).

The result of this dot product operation is a single value, which represents the weighted sum of the pixel values in the filter's receptive field. By sliding the filter across the entire input image, a new output matrix is generated, known as the feature map or activation map. This feature map contains information about the presence and intensity of specific features detected by the filter.

The convolution operation allows the network to learn and extract meaningful features from the input images, such as edges, textures, shapes, or other visual patterns. These extracted features are then passed to subsequent layers of the CNN for further processing and analysis.

Overall, the Convolutional Layer serves as the initial step in the CNN architecture, enabling the network to capture and identify important visual features in the input images, which ultimately aids in accurate classification or analysis tasks.

The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image



11. Fig. 4.2 Convolutional Layer Architecture

## **4.2 POOLING LAYER**

Following the Convolutional Layer in a Convolutional Neural Network (CNN), it is common to have a Pooling Layer. The main purpose of this layer is to reduce the size of the convolved feature maps, leading to a reduction in computational costs. This is achieved by decreasing the number of connections between layers, thereby simplifying the subsequent computations.

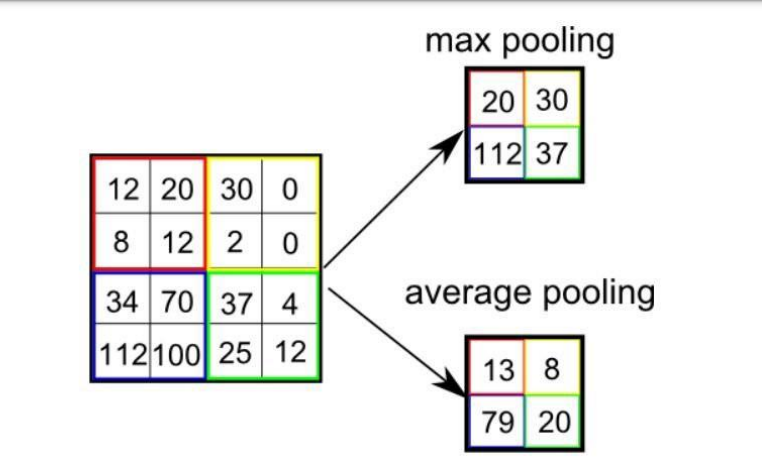
The Pooling Layer operates independently on each feature map and performs a down sampling operation. There are several types of pooling operations that can be used, depending on the specific method employed.

One common type is Max Pooling, where the maximum value within a predefined region of the feature map is selected. This helps to retain the most prominent features while discarding less relevant information.

Another type is Average Pooling, which calculates the average value of the elements within a predefined section of the feature map. This provides a way to capture the overall intensity or average representation of the features present.

Sum Pooling is another variant, where the total sum of the elements within the predefined section of the feature map is computed. This can be useful in situations where the absolute values or the cumulative presence of features are of interest.

The Pooling Layer acts as a bridge between the Convolutional Layer and the Fully Connected (FC) Layer. It reduces the spatial dimensions of the feature maps, making them more manageable for subsequent processing. By reducing the size, the Pooling Layer helps to extract the most salient information while discarding redundant details.

Overall, the Pooling Layer plays a crucial role in reducing the computational burden and providing a more compact representation of the features extracted by the Convolutional Layer. It serves as an important intermediary step in the CNN architecture, facilitating the transition from the convolutional feature extraction to the subsequent fully connected layers for final classification or analysis.

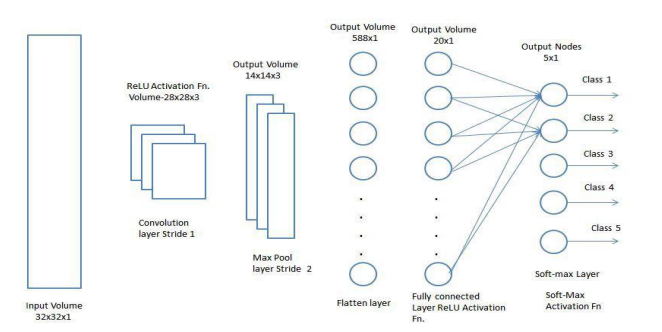
12. Fig 4.3 Pooling Layer

## **4.3 FULLY CONNECTED LAYER**

The Fully Connected (FC) layer is a crucial component of a Convolutional Neural Network (CNN) and is responsible for connecting neurons between different layers. It typically appears towards the end of the network, just before the output layer, and is composed of neurons along with their associated weights and biases.

In the FC layer, each neuron is connected to every neuron in the previous layer. This means that each neuron in the FC layer receives input from all the neurons in the preceding layer. This dense connectivity allows for more complex relationships and higher-level feature representation.

The FC layer plays a significant role in capturing the learned features from the earlier layers of the CNN and combining them to make predictions or classifications. By leveraging the connections between neurons, the FC layer can extract and process information from the extracted features, leading to more accurate and meaningful outputs.

Overall, the FC layer is a key component in the CNN architecture, acting as a link between the earlier layers and the final output layer. It enables the network to learn complex patterns and relationships within the data, contributing to the overall performance and effectiveness of the CNN model.

13. Fig 4.4 Fully connected Layer

In the Fully Connected (FC) layer, the input image from the preceding layers is transformed into a flattened vector and passed through the FC layer. The flattened vector then goes through additional FC layers where mathematical operations and computations typically occur. This stage marks the initiation of the classification process, where the network starts making predictions and assigning labels to the input data based on the learned features and connections in the FC layers.

## **4.4. ACTIVATION FUNCTIONS**

An activation function in a neural network determines how the weighted sum of inputs is transformed into an output from a node or layer. It is sometimes referred to as a "transfer function" or "squashing function" if its output range is limited. Activation functions are typically nonlinear and play a crucial role in the capability and performance of the network.

The choice of activation function can vary across different parts of the model and has a significant impact on the network's behavior. Activation functions are used within or after the internal processing of each node in the network, and they are usually consistent within a layer. Neural networks typically consist of input layers, hidden layers, and output layers. The hidden layers typically use the same activation function, while the output layer may have a different function depending on the desired prediction.

Activation functions are typically differentiable, meaning their derivatives can be calculated. This is necessary for training neural networks using backpropagation, which requires the derivative of prediction error to update the model's weights.

There are various types of activation functions used in neural networks, but only a few are commonly employed for hidden and output layers. Some well-known activation functions include ReLU, Softmax, tanh, and Sigmoid. Each function serves a specific purpose, such as binary classification using sigmoid or softmax functions, and multi-class classification often utilizing softmax.

The activation function is a critical parameter in CNN models as it enables the learning and approximation of complex relationships between network variables. In simpler terms, it determines which information should be activated during forward propagation and which should not. By adding non-linearity, it enhances the network's capability to model complex patterns.

Commonly used activation functions include ReLU, Softmax, tanh, and Sigmoid. Each function serves a specific purpose. For binary classification CNN models, sigmoid and softmax functions are typically preferred. For multi-class classification, softmax is commonly used.

In summary, the activation function plays a vital role in a CNN model by controlling the flow of information and introducing non-linearity, allowing the network to capture intricate relationships among variables. Different activation functions are selected based on the specific requirements of the classification task.

A key component of CNN architecture is the convolutional layer, which extracts and isolates different features of an image for further analysis. This process is known as feature extraction. The extracted features are then passed to a fully connected layer, which utilizes the output of the convolutional process to predict the class or category of the image based on the learned features from previous stages.

## **4.5 APPLICATION:**

1. Object detection: CNNs have revolutionized object detection in various applications such as autonomous vehicles and facial detection. Advanced models like R-CNN, Fast R-CNN, and Faster R-CNN, which rely on CNNs, are widely used for object detection tasks.

2. Semantic segmentation: In the field of image segmentation, CNNs have made significant advancements. Deep Parsing Network, developed by researchers from Hong Kong, utilized CNNs to enhance image segmentation by incorporating comprehensive information. Similarly, researchers from UC Berkeley developed fully convolutional networks that achieved state-of-the-art results in semantic segmentation.

# **CHAPTER 5**

# **EXPERIMENTAL ANALYSIS AND RESULTS**

## **5.1 SYSTEM CONFIGURATION**

The average system configuration for a digit recognition system can vary depending on the specific requirements and scale of the application. However, here is a general outline of the components and their configuration:

**1. Hardware:**

* **CPU:** A multi-core processor (e.g., Intel Core i5 or i7) or a dedicated GPU (e.g., NVIDIA GeForce GTX or RTX series) for accelerated training and inference.
* **Memory (RAM):** At least 4GB or 8GB, but preferably 16GB or more, to handle the data and model requirements.
* **Storage:** Sufficient storage capacity for storing datasets, trained models, and other related files.

**2. Software:**

* **Operating System:** A popular choice is a Linux-based OS like Ubuntu, which offers good compatibility with machine learning frameworks and libraries.
* **Python:** Python is a widely used programming language that was created by Guido Van Rossum and introduced in 1991. It is an interpreted language known for its high-level nature and is designed to prioritize code readability by utilizing significant whitespace. Python's language features and object-oriented approach are intended to assist programmers in writing clear and organized code for projects of all sizes. It is a dynamically typed language with built-in garbage collection. Python supports various programming paradigms, including procedural, object-oriented, and functional programming.
* **Deep Learning Framework:** TensorFlow, PyTorch, or Keras are commonly used frameworks for building and training neural networks.
* **IDE :** **Jupyter Notebook:** Jupyter is a popular and freely available web-based tool that serves as a computational notebook. It allows researchers to integrate software code, computational results, explanatory text, and multimedia content within a single document. Although computational notebooks have existed for a long time, Jupyter has gained significant popularity in recent years. This can be attributed to its thriving community of user-developers and an enhanced architecture that enables seamless notebook functionality.

3**. Data Preprocessing:**

* Image Resizing: Resize input images to a consistent size suitable for the model architecture (e.g., 28x28 pixels for MNIST).
* **Normalization:** Normalize pixel values to a common range (e.g., 0 - 1) to improve convergence and performance.

**4. Model Architecture:**

* **Convolutional Neural Network (CNN):** Use a CNN architecture suitable for digit recognition, such as LeNet, AlexNet, or a custom-designed network.
* **Layer Configuration:** Configure the number and size of convolutional layers, pooling layers, and fully connected layers based on the complexity of the task and available computational resources.

**5. Training and Optimization:**

* **Loss Function:** Choose an appropriate loss function, such as categorical cross-entropy, for multi-class digit classification.
* **Optimization Algorithm:** Adam, as discussed earlier, is a commonly used optimization algorithm that works well for digit recognition.
* **Hyperparameter Tuning:** Experiment with different learning rates, batch sizes, and regularization techniques (e.g., dropout) to optimize model performance.

**6. Evaluation and Testing:**

* **Performance Metrics:** Accuracy, precision, recall, and F1-score are commonly used metrics to evaluate the model's performance on the test dataset.
* **Cross-Validation:** Perform k-fold cross-validation to assess the model's generalization ability and mitigate overfitting.

It's important to note that the above configuration is a general guideline, and specific system requirements may vary depending on the dataset size, model complexity, and available computational resources.

## **5.2 LEARNING CURVES**

A learning curve is a graphical representation that illustrates the relationship between cost and output over a specific time period. It is commonly used to analyze the efficiency of production and predict costs. The concept of the learning curve was initially introduced by psychologist Hermann Ebbinghaus in 1885. In the visual depiction of a learning curve, a steeper slope indicates that initial learning leads to greater cost savings, while subsequent learning experiences result in diminishing cost savings at a slower pace.

A learning curve is a graphical representation that showcases the progress of a specific learning metric during the training of a machine learning model. It provides insights into the learning process and helps monitor the model's performance over time or progress.

Typically, the x-axis represents time or progress, while the y-axis represents the error or performance metric. Learning curves are valuable tools for diagnosing issues, optimizing prediction performance, and gaining a deeper understanding of the model's evolution throughout the learning process.

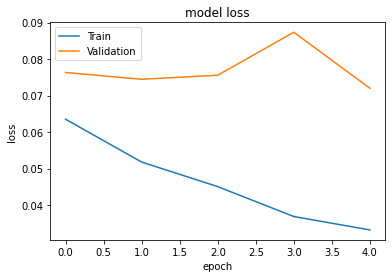
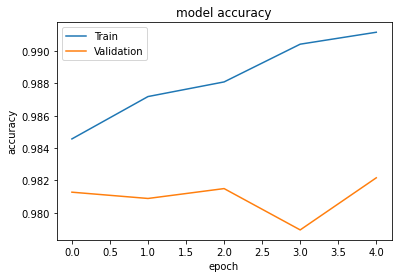
There are commonly two types of learning curves that appear in charts:

1. **Optimization Learning Curves:** These learning curves are calculated based on the metric used to optimize the parameters of the model. Common metrics for optimization includes loss functions like Mean Squared Error (MSE). Optimization learning curves show how the optimization metric changes as the model's parameters are updated during the learning process. They provide insights into the convergence and efficiency of the optimization algorithm.

2. **Performance Learning Curves**: These learning curves are calculated based on the metric used to evaluate and select the model's performance. Examples of performance metrics include accuracy, precision, recall, or F1 score. Performance learning curves depict how the model's performance on the chosen metric evolves as the training progresses. They help assess the model's ability to generalize and make accurate predictions on unseen data.

Both optimization and performance learning curves are important in understanding the behavior of the model during training and can guide decisions on model improvement and selection.

Accuracy and Loss are the two most well-known and discussed metrics in machine learning.



14. Fig. 5.1 Accuracy and Loss Graph

From the above curve we can say that accuracy during training and validation has increased with

increase in number of epochs and loss has been subsequently decreases during training and validation.

### **5.2.1 Accuracy Curve**

The accuracy curve is a graphical representation of the performance of a classification model. It measures the proportion of correct predictions made by the model, usually expressed as a percentage. Accuracy indicates how well the model predicts the true values compared to the predicted values. It is a binary measure, representing the correctness or incorrectness of predictions for individual samples. During the training phase, accuracy is often plotted and monitored, although its value is commonly associated with the overall or final accuracy of the model. Accuracy is a more straightforward metric to interpret than loss.

### **5.2.2 Loss Curve**

The loss curve represents the performance of a model by measuring the discrepancy between predicted values and true values. It is a quantitative measure of how well the model is able to make predictions. Unlike accuracy, which is a percentage, the loss is a numerical value that indicates the sum of errors made by the model on the training or validation data.

During the training process, the objective is to minimize the loss by adjusting the model's parameters, such as weights in a neural network. Various loss functions can be used, such as log loss, cross-entropy loss, mean squared error, or likelihood loss, depending on the specific problem. These loss functions capture the differences between predicted and true values in different ways.

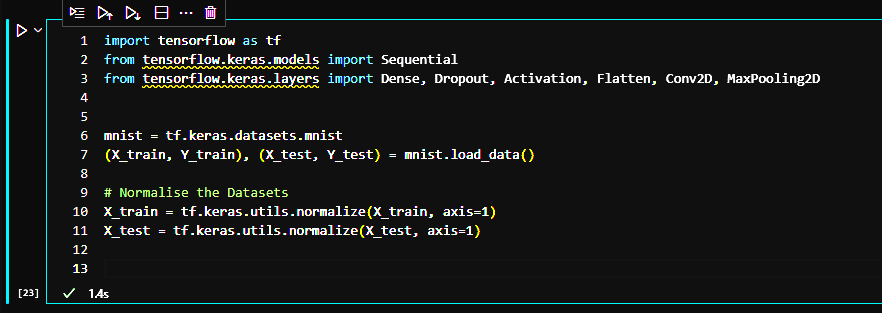
The loss curve is commonly used in both classification and regression problems, providing insight into how well the model is learning and improving over time. By monitoring the loss curve during training, researchers can assess the model's performance and adjust as needed.

One of the most used plots for debug a neural network is a Loss curve during training. It gives us an overview of the training process and the direction in which the network learns.

## **5.3 CODE SEGMENT**

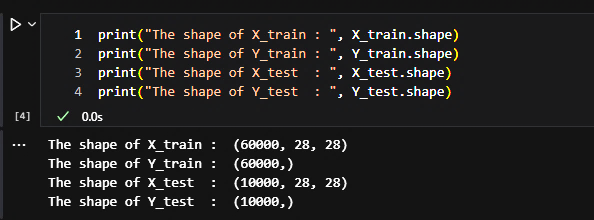
### **5.3.1 Import the Libraries**

To begin training our model, we need to import the necessary modules. We will be using the Keras library, which includes preloaded datasets like MNIST. By using the mnist.load\_data() method, we can easily import the MNIST dataset, which provides us with the training data, corresponding labels, as well as the testing data and its labels.



15. Fig 5.2 Importing Libraries & normalizing the dataset

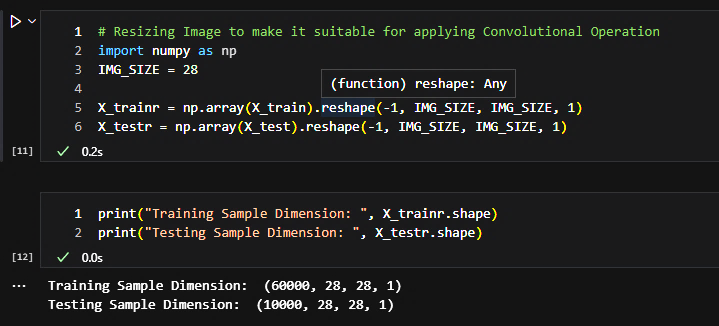
### **5.3.2 Shape of Data**

****

**16**. Fig. 5.3 Shape of dataset

### **5.3.3 Resizing Image for Convolutional Operations**

To apply convolutional operations in a CNN, it is necessary to reshape the input images to include the channel dimension. The MNIST dataset consists of grayscale images, so the channel dimension will have a value of 1. The following code snippet demonstrates how to resize the MNIST images to the appropriate dimensions:

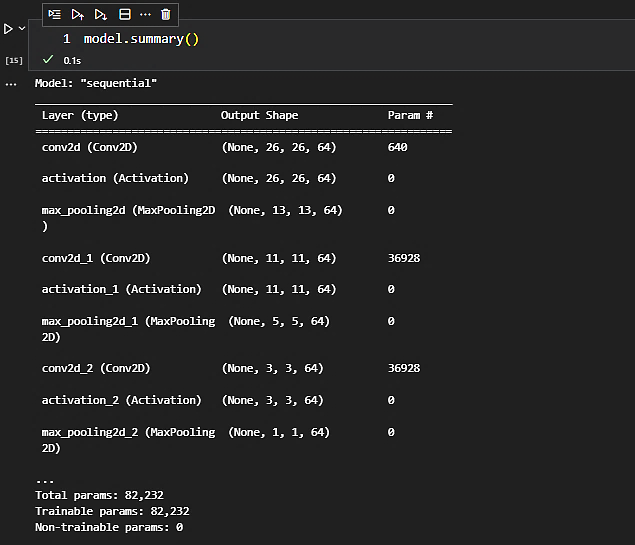


17. Fig. 5.4 Resizing the Image

### **5.3.4 Creating A Deep Neural Network**

18. Fig 5.5 Creating a DNN Model

### **5.3.5 Summary of The Model**



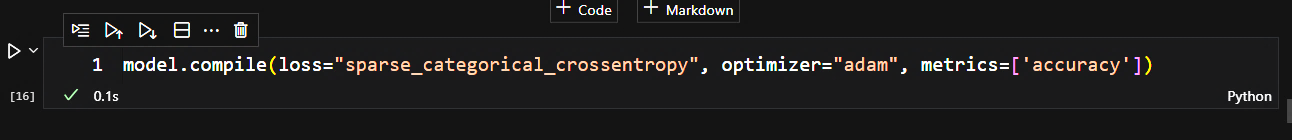
19. Fig 5.6 Summary of the model

### **5.3.6 Compile the CNN Model**

The ‘model.compile()’ function is used to configure the learning process of the model. It specifies the loss function, optimizer, and metrics to be used during training and evaluation. Here's an explanation of the arguments used in model.compile():

* loss: The loss function measures the difference between the predicted output and the true output. In this case, `"sparse\_categorical\_crossentropy"` is used as the loss function. This loss function is suitable for multi-class classification problems where the labels are integers (e.g., the MNIST dataset contains integer labels representing digits from 0 to 9).
* optimizer: The optimizer determines how the model is updated based on the calculated gradients. The `"adam"` optimizer is used in this case. Adam `(Adaptive Moment Estimation)` is a popular optimizer that combines the benefits of two other optimizers, AdaGrad and RMSProp. It adapts the learning rate for each parameter, leading to efficient and effective optimization.
* metrics: Metrics are used to evaluate the performance of the model. The specified metrics are calculated and reported during training and evaluation. In this case, the model is evaluated based on the accuracy metric, which measures the proportion of correctly classified samples.

By calling ‘model.compile()’ with the specified arguments, the model is prepared for training and evaluation. It sets up the necessary components to optimize the model's weights using the specified loss function and optimizer, and it tracks the specified metrics to assess the model's performance.



20. Fig 5.7 Compilation of model

### **5.3.7 Fit the Model**

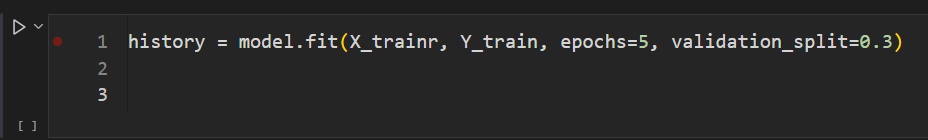
The `model.fit()` function is used to train the model on the provided training data. Here's a brief explanation of the arguments used in `model.fit()`:

- X\_trainr: The training data, which includes the preprocessed and resized MNIST images as input to the model.

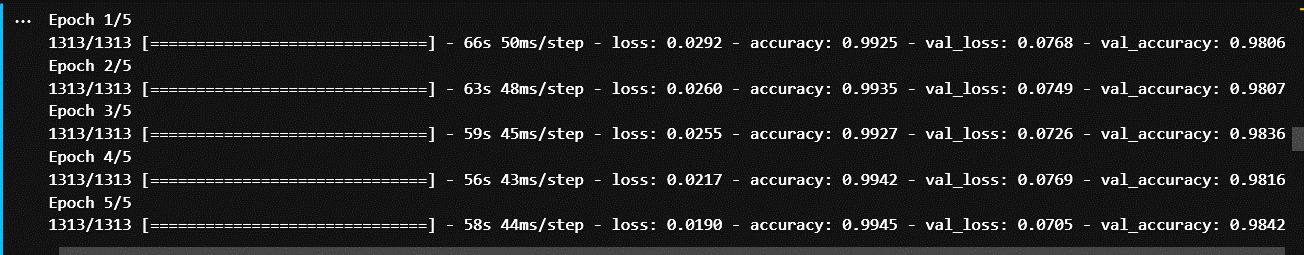
- Y\_train: The corresponding labels for the training data, representing the true digits for each image.

- epochs: The number of epochs defines the number of times the model will iterate over the entire training dataset. In this case, the model will train for 5 epochs, meaning it will go through the entire training dataset 5 times during training.

- validation\_split: The validation\_split parameter specifies the fraction of the training data to be used for validation. In this case, 30% of the training data will be reserved for validation during training.

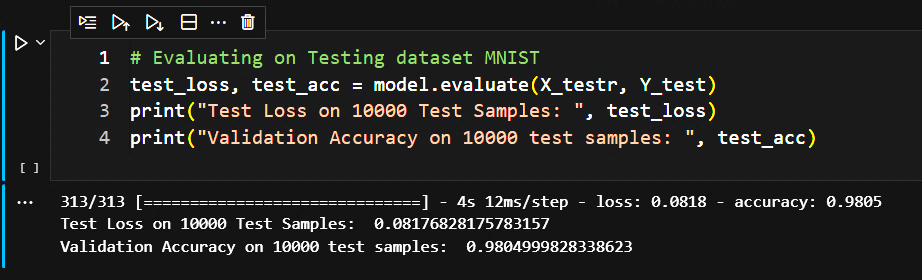


21. Fig. 5.8 Fit the model



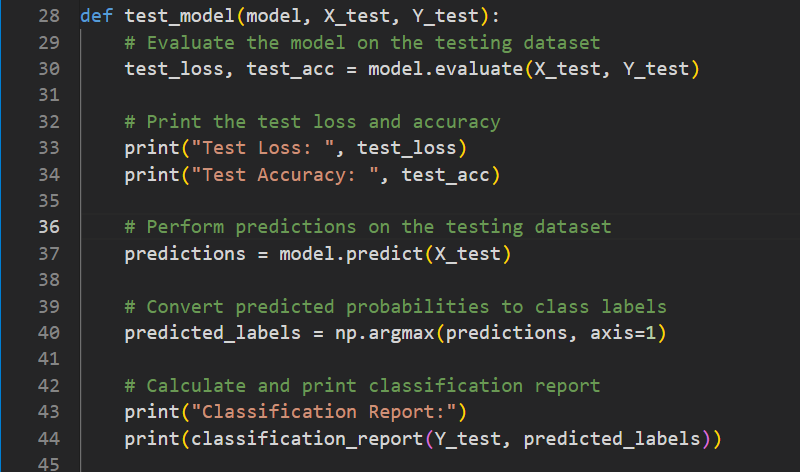
22. Fig.5.9 Output of the Trained Model

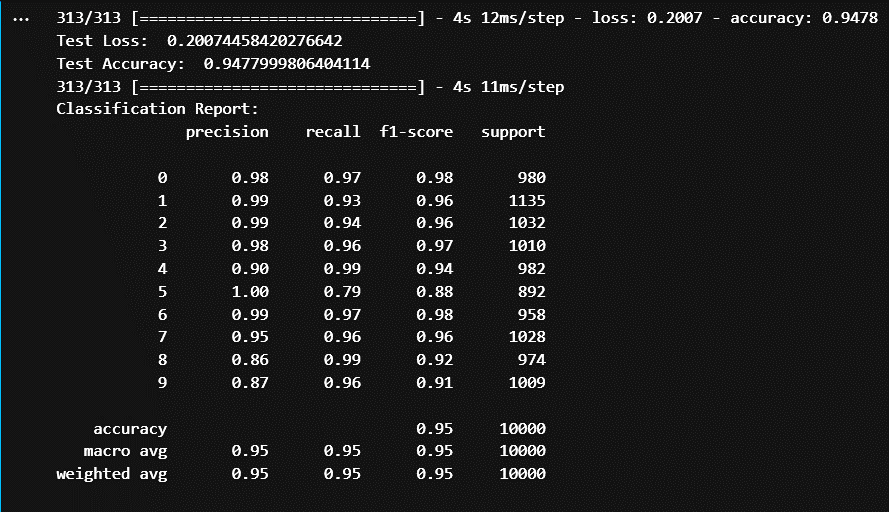
### **5.3.8 Evaluation of MNIST Dataset**



23. Fig. 5.10 Evaluation of Dataset

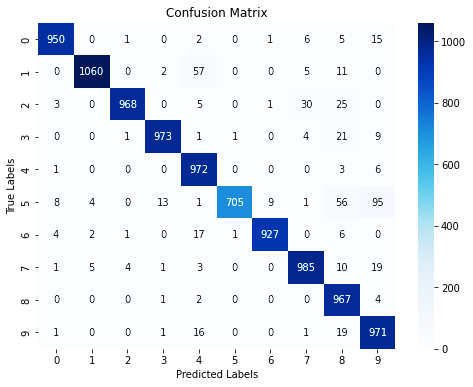
### **5.3.9 Testing the Model**

****



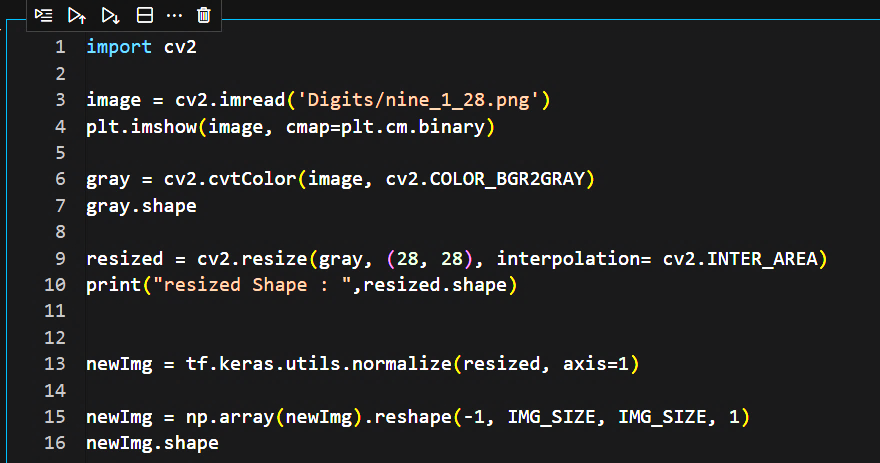
24. Fig. 5.11 Testing & Output of the Test Model

### **5.3.10 Confusion Matrix**

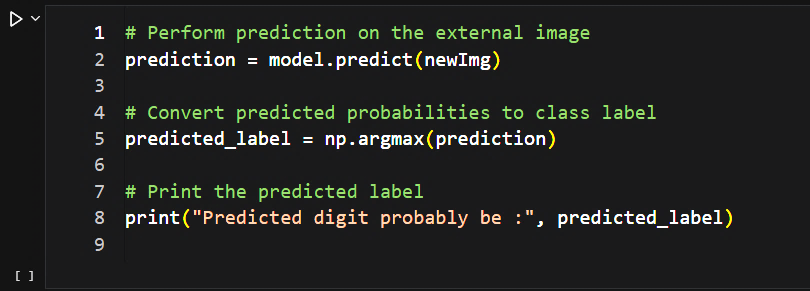


25. Fig 5.12 Confusion Matrix

### **5.3.11 Prediction of the Model**

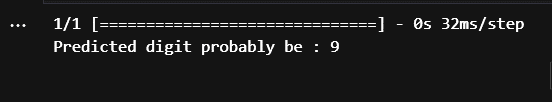
****

26. Fig. 5.13 Code for Image Read

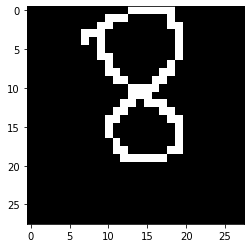
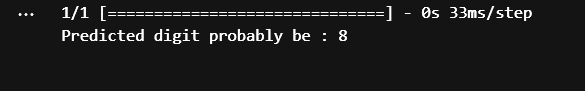


27. Fig 5.14 Prediction of model

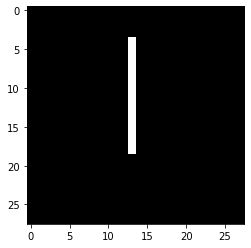
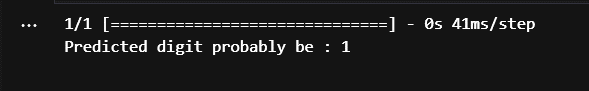
## **5.4 OUTPUT**



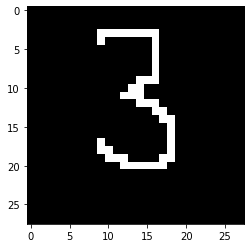
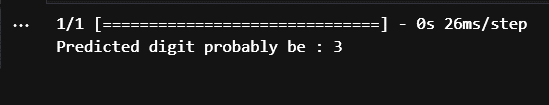
28. Fig 5.15 Output ‘digit – 9’



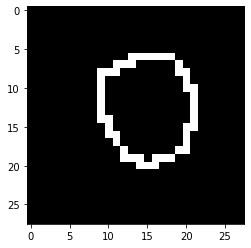
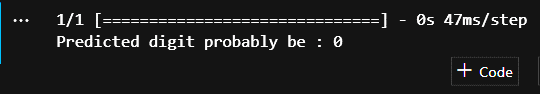
29. Fig 5.16 Output ‘digit – 8’



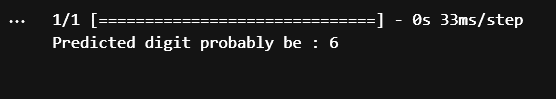
30. Fig 5.17 Output ‘digit – 1’



31. Fig 5.18 Output ‘digit – 3’



32. Fig 5.19 Output ‘digit – 0’



33. Fig 5.20 Output ‘digit – 6’

# **CHAPTER 6**

# **APPENDIX**

**PYTHON**:

Python is a programming language developed by Guido Van Rossum, released in 1991. It is an interpreted, high-level language that is widely used for various purposes. Python prioritizes code readability by utilizing significant whitespace. Its design philosophy focuses on enabling programmers to write clear and logical code for projects of all sizes. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming. It is dynamically typed and features automatic garbage collection.

**JUPYTER NOTEBOOK:**

JupyterLab is an interactive development environment based on the web, specifically designed for working with Jupyter notebooks, code, and data. It offers flexibility in terms of configuring and arranging the user interface to accommodate various workflows in data science, scientific computing, and machine learning. JupyterLab's extensibility and modularity allow users to create plugins that introduce new components and seamlessly integrate them with the existing functionality.

**KERAS:**

Keras is a Python library that offers a user-friendly and efficient framework for developing and evaluating deep learning models. It serves as a wrapper for the numerical computation libraries TensorFlow and Theano, simplifying the process of defining and training neural network models with minimal code. While TensorFlow and Theano are powerful but complex libraries for creating neural networks, Keras provides a higher-level abstraction that makes it easier to understand and work with. By leveraging TensorFlow or Theano as its backend, Keras enables the quick and straightforward creation of deep learning models. Therefore, Keras is an excellent choice for various deep learning applications.

To create a model using Keras, the following steps are typically followed:

1) Define the network model: This involves specifying the architecture of the model, including the type and number of layers, as well as their connectivity. The Keras library provides the necessary modules for defining the model structure, such as `Sequential` for a linear stack of layers.

2) Compile the model: After defining the model, it needs to be compiled to configure its learning process. This step involves specifying the loss function, the optimizer, and any evaluation metrics that will be used during training. The loss function measures the error between the predicted output and the true output, while the optimizer determines how the model's parameters are updated based on the computed gradients.

3) Train or fit the network: In this step, the model is trained on a training dataset to learn the underlying patterns and relationships in the data. The model is presented with input samples along with their corresponding target outputs, and it adjusts its parameters iteratively to minimize the defined loss function. The training process continues for a specified number of epochs or until a convergence criterion is met.

To implement these steps in code, you can import the necessary modules from Keras: `Sequential` from `keras.models`, and `Dense`, `Activation`, and `Dropout` from `keras.layers`. These modules provide the tools for defining the model architecture, specifying the activation functions, and incorporating regularization techniques like dropout.

**TensorFlow:**

TensorFlow is a popular Python library created by Google for efficient numerical computing. It serves as a fundamental tool for developing Deep Learning models directly or through wrapper libraries built on top of it. The TensorFlow tutorial caters to beginners and professionals, covering essential concepts in machine learning and deep learning, including deep neural networks, image processing, and sentiment analysis.

As a well-known deep learning framework, TensorFlow offers a range of features. It is an open-source software library developed in Python, enabling easy implementation of deep learning projects. Its design prioritizes simplicity and efficiency, making it user-friendly. Unlike other numerical libraries primarily focused on deep learning, such as Theano, TensorFlow is versatile and suitable for both research and production systems. It can be utilized on various hardware configurations, including single CPU systems, GPUs, mobile devices, and distributed systems involving numerous machines.

**NUMPY:**

NumPy is a Python library specifically designed for array manipulation. It offers a wide range of functionalities in the areas of linear algebra, Fourier transform, and matrix operations. NumPy, short for Numerical Python, consists of multidimensional array objects and a collection of routines for efficient processing of these arrays. It provides the ability to perform mathematical and logical operations on arrays, making it a valuable tool for numerical computations.

This tutorial provides a comprehensive introduction to NumPy, covering its architecture, environment, and various array functions. It also explores topics such as array indexing and manipulation. NumPy is an open-source project, freely available for use. It aims to enhance computational efficiency by providing an array object, called ndarray, that can be up to 50 times faster than traditional Python lists. Given its significant speed advantage, arrays from NumPy are extensively utilized in data science applications where performance and resource utilization are critical factors.

**MATPLOTLIB:**

Matplotlib is a widely used data visualization library in Python that provides a comprehensive set of tools for creating various types of plots and charts. It is a powerful and flexible library that allows users to generate high-quality visualizations for data analysis, exploration, and presentation.

Key Features:

1. Plotting Functions: Matplotlib offers a wide range of plotting functions that enable the creation of line plots, scatter plots, bar plots, histograms, pie charts, heatmaps, and many more. These functions provide flexibility in customizing plot aesthetics, such as colors, markers, line styles, labels, and annotations.

2. Object-Oriented API: Matplotlib provides an object-oriented API that allows users to have fine-grained control over the elements of a plot. This API allows for the creation and manipulation of Figure and Axes objects, providing more advanced customization options and enabling the creation of complex, multi-panel plots.

3. Integration with NumPy and Pandas: Matplotlib seamlessly integrates with other popular scientific libraries in Python, such as NumPy and Pandas. This integration allows users to directly plot data stored in NumPy arrays or Pandas DataFrames, simplifying the data visualization process.

4. Customization and Styling: Matplotlib provides extensive customization options, allowing users to style their plots according to their preferences. Users can modify plot elements such as axes, grids, legends, fonts, and colors to create visually appealing and informative visualizations.

**MACHINE LEARNING:**

Machine learning is an approach to analyzing data that involves the automated construction of analytical models. It falls under the field of artificial intelligence and operates on the principle that systems can gain knowledge from data, recognize patterns, and make informed decisions with limited human involvement.

**DEEP LEARNING:**

Deep learning is a branch of artificial intelligence (AI) that simulates the functioning of the human brain to process data and generate patterns used for decision-making. It is a specific subset of machine learning within AI, characterized by its ability to learn from unstructured or unlabeled data through unsupervised learning. Often referred to as deep neural learning or deep neural networks, deep learning algorithms enable complex data analysis and pattern recognition.

**NEURAL NETWORKS:**

A neural network is a collection of algorithms designed to identify inherent connections within a dataset, emulating the functioning of the human brain. Neural networks can be organic or artificial systems composed of interconnected neurons. They are used to uncover patterns and relationships in data by mimicking the cognitive processes of the human brain.

# **CHAPTER 7**

# **CONCLUSION AND FUTURE WORK**

## **7.1 CONCLUSION:**

Our project, called HANDWRITTEN DIGIT RECOGNITION, focuses on the task of identifying handwritten digits. The primary objective of this project is to develop an automated method for accurately recognizing handwritten digit strings. To achieve this, various machine learning techniques such as Support Vector Machine (SVM), Artificial Neural Networks (ANN), and Convolutional Neural Networks (CNN) architectures are employed. These methods are chosen to ensure optimal performance in accurately recognizing digit strings.

## **7.2 FUTURE WORK:**

In future work, the existing system can be enhanced and extended to develop a Handwritten Character Recognition System. Currently, the system takes input images of size 28x28 pixels and focuses on recognizing handwritten digits. However, with additional modifications and improvements to the dataset and the model, it is possible to expand the system's capabilities to recognize and predict human handwritten characters. This would involve training the model on a larger and more diverse dataset of handwritten characters, and implementing appropriate modifications to the architecture and algorithms used in the recognition process. By undertaking these enhancements, the system can be adapted to handle a broader range of handwritten characters and provide more comprehensive recognition capabilities.

# **REFERENCES**

1. [**www.tensorflow.org**](https://www.tensorflow.org/)
2. [**python.org/downloads**](https://www.python.org/downloads/)
3. [**jupyter.org**](https://jupyter.org/)
4. [**docs.python.org/3/library**](https://docs.python.org/3/library/)
5. [**stackoverflow.com**](https://stackoverflow.com/)
6. [**www.kaggle.com/code/arunrk7/digit-recognition-using-cnn-99-accuracy**](https://www.kaggle.com/code/arunrk7/digit-recognition-using-cnn-99-accuracy)
7. [**medium.com/handwritten-digit-recognition-hdr-using-k-nearest-neighbors-knn-f4c794a0282a**](https://medium.com/@nikhilanandikam/handwritten-digit-recognition-hdr-using-k-nearest-neighbors-knn-f4c794a0282a)
8. [**matplotlib.org**](https://matplotlib.org/)
9. [**https://code.visualstudio.com/docs/datascience/jupyter-notebooks**](https://code.visualstudio.com/docs/datascience/jupyter-notebooks)
10. [**cse.anits.edu.in/projects/projects2021A10.pdf**](https://cse.anits.edu.in/projects/projects2021A10.pdf)
11. [**www.datacamp.com/tutorial/convolutional-neural-networks-python**](https://www.datacamp.com/tutorial/convolutional-neural-networks-python)
12. [**towardsdatascience.com/image-recognition-with-machine-learning-on-python-image-processing-3abe6b158e9a**](https://towardsdatascience.com/image-recognition-with-machine-learning-on-python-image-processing-3abe6b158e9a)