A Mini Project Report

On

NUMERAL RECOGNITION BY HANDWRITING USING CONVOLUTIONAL NEURAL NETWORKS AND CAPSULE NETWORKS

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Under the Guidance of

DR. D. HIMAJA

Assistant Professor

BY

V.BHAVYA SREE	**(22D21A05J1)**
M.KAVYA	**(22D21A05F2)**
D.NAVYA SRI	**(22D21A05D3)**

Department of Computer Science and Engineering

SRIDEVI WOMEN'S ENGINEERING COLLEGE

(An UGC Autonomous Institution)

(Estd. 2001 | Approved by AICTE & Govt. of TS |Affiliated to JNTUH

Accredited by NBA and NAAC(A++) |Certified with ISO 9001:2015

V.N.PALLY, Gandipet, Hyderabad-75

2024-25

Department of Computer Science and Engineering

SRIDEVI WOMEN'S ENGINEERING COLLEGE

(An UGC Autonomous Institution)

(Estd. 2001 | Approved by AICTE & Govt. of TS |Affiliated to JNTUH

Accredited by NBA and NAAC(A++) |Certified with ISO 9001:2015

V.N.PALLY, Gandipet, Hyderabad-75

2024-25

CERTIFICATE

This is to certify that the MINI PROJECT report entitled **"NUMERAL RECOGNITION BY HANDWRITING USING CONVOLUTIONAL NEURAL NETWORKS AND CAPSULE NETWORKS"** is being submitted by **V.Bhavya Sree (22D21A05J1), M.Kavya (22D21A05F2), D.Navya Sri (22D21A05D3)** in partial fulfillment for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** is a record of bonafide work carried out by them.

INTERNAL GUIDE	COORDINATOR	HEAD OF THE DEPARTMENT
DR. D. HIMAJA **Assistant Professor**	**DR. D. HIMAJA** **Assistant Professor**	**Dr. U. SRI LAKSHMI** **Professor**

EXTERNAL EXAMINER

DECLARATION

We, hereby declare that Mini project entitled "**Numeral Recognition By Handwriting Using Convolutional Neural Networks and Capsule Networks"** is the work done during the period from **27 January, 2025 to 14 June, 2025** and is submitted in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering from Jawaharlal Nehru Technological University, Hyderabad.

V.BHAVYA SREE	**(22D21A05J1)**
M.KAVYA	**(22D21A05F2)**
D.NAVYA SRI	**(22D21A05D3)**

ACKNOWLEDGEMENT

We would like to express our sincere gratitude and indebtedness to our **Guide DR. D. HIMAJA, Assistant Professor, Department of Computer Science and Engineering** for his/her valuable guidance, suggestions, and keen personal interest throughout the course of **industry oriented Mini Project** and for his tireless patience in hearing all our seminars, minutely seeing all the reports and giving appropriate guidance and suggestions.

We would like to express our sincere gratitude and indebtedness to our **Project Coordinator, DR. D. HIMAJA , Assistant Professor Department of Computer Science and Engineering** for her **expert counsel, insightful suggestions, and sincere concern** throughout the course of **Industry Oriented Mini Project.**

We would like to express our sincere gratitude **to Dr. U.SRI LAKSHMI, PROFESSOR & HOD of Department of Computer Science and Engineering** for her valuable suggestions and advices

We also extremely thankful to **Dr.K.SIVA NAGI REDDY, Dean Academics** for his precious guidance.

We also extremely thankful to Professor and **DR. A.NARMADA, Principal, Sridevi Women's Engineering College** for her precious suggestions.

Finally, we would like to take this opportunity to thank all our faculty, family and friends for their support throughout this work. We also sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

V.BHAVYA SREE	**(22D21A05J1)**
M.KAVYA	**(22D21A05F2)**
D.NAVYA SRI	**(22D21A05D3)**

INDEX

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Handwritten numeral recognition is essential for many real-world applications, including digital data entry, bank check verification, and postal automation. Through the use of deep learning techniques, specifically Convolutional Neural Networks (CNNs) and Capsule Networks (CapsNets), this study investigates an effective and precise method for numeral recognition. By using convolutional and pooling operations to learn spatial hierarchies of data, CNNs have shown impressive performance in picture classification challenges. Their dependence on max-pooling, however, may result in the loss of crucial geographical data. In response, Capsule Networks are presented, which provide better generalization for affine-transformed or rotated digits while maintaining hierarchical pose relationships. In this work, typical handwritten number datasets, such MNIST, are used to compare CNNs and CapsNets in order to assess classification accuracy, robustness, and computing economy. According to the data, Capsule Networks' dynamic routing mechanism gives them a higher accuracy in identifying distorted or overlapping numbers than CNNs, despite CNNs' quick and dependable performance. This study demonstrates how appropriate deep learning architectures can be combined or chosen to provide high-performance handwritten numeral recognition in real-world applications.

CHAPTER 1

INTRODUCTION

Numerical Recognition Using Handwriting enables computers to recognize and comprehend handwritten numerals (0–9) from photographs or written texts. It allows machines to read human-written numbers and transform them to a digital version for later processing. To improve accuracy, this system combines two powerful techniques: Convolutional Neural Networks (CNN) and Capsule Networks (CapsNets). CNN is frequently used for image identification because it effectively finds patterns in images, whereas CapsNets offer an improved method that not only identifies patterns but also retains their spatial relationships, such as location and form. Using these strategies enhances the system's recognition accuracy and robustness, making it more suitable for real-world applications. To address these shortcomings, Capsule Networks (CapsNets) have been proposed as a more effective option for modeling spatial hierarchies using dynamic routing and vector-based representation. This study looks into the usage of CNNs and CapsNets for accurate and robust handwritten numeral recognition, comparing their strengths and weaknesses in managing complex variances in handwritten digits.

1.1 PURPOSE:

The main purpose of this study is to create a reliable and accurate system for handwritten numeral recognition utilizing cutting-edge deep learning techniques, particularly Convolutional Neural Networks (CNNs) and Capsule Networks (CapsNets). Because handwritten numbers frequently differ in size, orientation, and writing style, good accuracy and generalization may be difficult for typical machine learning techniques to achieve. In order to enhance recognition performance, this study intends to utilize CNNs' feature extraction capabilities and Capsule Networks' spatial relationship-preserving characteristics. The study compares the advantages and disadvantages of the two architectures in an effort to determine which method is best for handwritten number recognition under various difficult circumstances. The ultimate goal is to contribute to the development of intelligent systems that can automate digit recognition tasks in real-world applications such as banking, education, and postal services.

1.2 SCOPE:

The study's scope includes the creation, application, and assessment of deep learning models for handwritten numeral recognition using Convolutional Neural Networks (CNNs) and Capsule Networks (CapsNets). To enable standardized performance comparison, this study focuses on training and testing these models using benchmark datasets like MNIST. The study looks at how well CNNs learn local spatial features and how well CapsNets handle distortions like rotations and overlapping digits while maintaining spatial hierarchies. It also compares the accuracy, training time, computational efficiency, and robustness of the two architectures. While the primary focus is on digit recognition in the English numeral system, the methodology can be applied to other numeral systems in future research. The scope is limited to offline recognition (scanned or digitally captured images), and it excludes real-time digit recognition from live video feeds or handwriting in cursive scripts.

CHAPTER 2

LITERATURE SURVEY

Traditional Approaches to Handwritten Digit Recognition:

Early research in handwritten numeral recognition relied heavily on traditional machine learning techniques like Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Multilayer Perceptrons (MLPs). These methods necessitated manual feature extraction, which included zoning, pixel density, and contour-based descriptors. While these techniques demonstrated reasonable accuracy on datasets such as MNIST, their performance suffered significantly when exposed to variations in handwriting style, orientation, and noise. These limitations highlighted the need for models that could learn features automatically from data, prompting the use of deep learning techniques in this domain. For instance, SVMs with radial basis function kernels performed well on MNIST, with accuracy exceeding 98%. However, their success was hampered by computational inefficiencies in large datasets and poor generalization to noisy or distorted images.

Emergence of Convolutional Neural Networks (CNNs) :

Convolutional Neural Networks transformed image classification and recognition by incorporating automated feature learning via convolutional layers. LeCun et al.'s pioneering work on LeNet-5 for handwritten digit recognition was highly successful, laying the groundwork for CNN-based numeral recognition systems. CNNs became popular because of their ability to learn hierarchical patterns and reduce dimensionality through pooling operations. Several studies improved LeNet by employing deeper architectures, dropout regularization, and ReLU activations, achieving cutting-edge accuracy on datasets such as MNIST and EMNIST. However, CNNs remained sensitive to affine transformations and lacked mechanisms for comprehending spatial hierarchies between features. Furthermore, CNNs performed well not only in clean and uniform datasets, but also with noisy, rotated, or scaled digit images, especially when data augmentation techniques were used during training. The architecture's ability to learn local patterns such as edges, corners, and shapes from various layers made it resistant to intra-class handwriting variations. Despite their success, CNNs have limitations when dealing with transformations such as rotation and skew because spatial relationships are lost during pooling operations. This shortcoming has prompted the development of alternative models such as Capsule Networks, which aim to better preserve spatial hierarchies.

Limitations of CNNs and the Rise of Capsule Networks :

Despite their success, CNNs have a significant flaw: they treat spatial relationships as less important due to pooling operations. This presents difficulties when recognizing digits with unusual orientations or overlapping strokes. In response, Geoffrey Hinton introduced Capsule Networks (CapsNets) in 2017, a novel architecture that preserves spatial hierarchies using vector-based neurons known as "capsules." CapsNets use dynamic routing algorithms rather than max-pooling, which allows them to capture part-whole relationships more efficiently. CapsNets outperformed CNNs in recognizing distorted or rotated digits, according to MNIST experiments, indicating that they hold promise for applications requiring higher spatial awareness. To overcome these limitations, Geoffrey Hinton et al. proposed Capsule Networks (CapsNets) in 2017. CapsNets introduce a new computational unit known as a capsule, which is a group of neurons that collectively represent both the presence of a feature and its pose (orientation, position, and scale). Capsules produce vector outputs, which encode more complex information, rather than scalar outputs like traditional neurons. The dynamic routing mechanism between capsules replaces max-pooling by allowing the network to learn part-whole relationships—specifically, how features combine to form more complex patterns like digits.

Comparative Studies Between CNNs and CapsNets :

Several researchers compared CNNs and CapsNets for numeral recognition. Sabour et al. (2017) found that CapsNets achieved 0.25% error on MNIST with fewer parameters than traditional deep CNNs. Later studies extended this comparison to more complex datasets, such as Fashion-MNIST and EMNIST, and discovered that, while CNNs are computationally faster, CapsNets provide greater accuracy and robustness in the presence of image transformations. Other studies found that CapsNets require longer training times and more computational resources, but they provide better generalization in cases of occlusion, rotation, and segmentation noise. Sabour et al. (2017) demonstrated in their original paper that CapsNets outperformed many deep CNN architectures on the MNIST dataset, with a test error rate of 0.25% and fewer training examples. CapsNets demonstrated a stronger ability to recognize digits even when portions of them were obscured or rotated, which is a direct result of their pose-aware feature representation.

Applications and Future Directions :

The combination of CNNs and CapsNets has inspired hybrid architectures that seek to strike a balance between CNN efficiency and CapsNet spatial learning capacity. Some researchers proposed using CNNs as feature extractors and capsule layers for classification, which would improve both speed and accuracy. There is an increasing interest in applying these models to real-world numeral recognition scenarios, such as mobile OCR apps, bank cheque processing, and regional numeral systems like Devanagari and Arabic. Future research is expected to focus on optimizing CapsNets for lower hardware requirements, expanding their application to online handwriting recognition, and investigating their efficacy in multilingual numeral systems. CNN-based models have already been integrated into many of these applications due to their high speed and consistent performance in standard conditions. However, their limitations under distortion, rotation, or partial occlusion limit their usefulness in uncontrolled or real-world settings. This makes a compelling case for incorporating Capsule Networks (CapsNets) into future OCR systems, particularly in situations requiring greater spatial awareness and transformation robustness. CapsNets, for example, can be extremely effective in autonomous systems like robotic readers or smart classroom tools that must interpret digits written in different orientations or on uneven surfaces.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM AND DISADVANTAGES :

The existing system for handwritten numeral recognition employs Convolutional Neural Networks (CNNs), which has proven to be extremely effective in extracting patterns and spatial features from image data. CNNs use convolutional layers to detect local features like edges, curves, and textures, which makes them ideal for tasks like digit classification. Despite their success, CNNs have several significant limitations, particularly when used in real-world scenarios involving handwritten digits. One of the most noticeable issues is CNNs' inability to maintain accuracy when digits are shifted, rotated, skewed, or distorted. These transformations can cause the network to misinterpret a digit, even if the overall shape is still identifiable to the human eye. Another significant limitation is the invariance problem. CNNs are inherently trained on fixed-position data, so they frequently fail to recognize the same digit when it appears in a different location or orientation within the image. This positional sensitivity results from max-pooling operations, which reduce feature spatial resolution and cause the loss of precise positional information. As a result, CNNs cannot accurately interpret overlapping digits or digits written with perspective variations.

1. CNNs are extremely sensitive to spatial variations, such as translation, rotation, and scaling, resulting in lower accuracy when digits are distorted, misaligned, or written at an unusual angle. This sensitivity stems from the use of max-pooling layers, which ignore spatial hierarchies and precise positional information.

2. CNNs treat features as independent and do not capture the hierarchical relationships between parts of a digit, so they can misclassify digits with similar components. Furthermore, CNNs' fixed receptive fields make them less effective at recognizing digits that appear in different sizes or locations in an image.

3. CNNs lack interpretability—they can recognize certain features but cannot explain how they are combined to form a digit. This makes the model susceptible to errors caused by minor variations in handwriting style, overlapping digits, or noise.

4. CNNs require a large amount of labeled training data to achieve good generalization, and they are prone to overfitting when the dataset is small or imbalanced.

5. Despite their high performance on benchmark datasets like MNIST, CNNs frequently underperform in real-world scenarios where digits are drawn quickly, written in cursive, or include strokes outside of typical boundaries.

6. CNNs are trained to recognize patterns in specific positions and are incapable of handling changes in digit perspective efficiently.

7. Small differences in handwriting style (e.g., slant, pressure, size) can confuse the network and cause errors.

8. CNNs require large amounts of labeled data to generalize effectively, and they are prone to overfitting on small or imbalanced datasets.

3.2 PROBLEM STATEMENT:

Handwritten numeral recognition is a difficult task, owing to the wide variation in handwriting styles between individuals, as well as distortions caused by factors such as writing speed, pen pressure, and medium used. In addition to stylistic differences, digit position, scale, rotation, and orientation all complicate the recognition process. Existing systems that use Convolutional Neural Networks (CNNs) have demonstrated significant success in extracting features and recognizing digits in controlled settings. When confronted with such real-world variations, these models frequently struggle to generalize effectively. CNNs frequently misclassify rotated, shifted, or overlapping digits because they are unable to fully capture spatial hierarchies and relationships between digit components. This limitation reduces their accuracy and dependability in practical situations. As a result, there is an increasing demand for more robust recognition systems capable of maintaining high accuracy in the face of these challenges. Such systems should be able to recognize not only the presence of digit features, but also their spatial arrangement and orientation, allowing for better differentiation between similar digits under a variety of conditions. Improving the robustness of handwritten numeral recognition has significant implications for a variety of industries, including banking (e.g., automated check processing), healthcare (e.g., digitizing patient records), postal services (e.g., mail sorting), and government document management.

3.3 PROPOSED SYSTEM AND ADVANTAGES:

The proposed system for handwritten numeral recognition combines the strengths of Convolutional Neural Networks (CNNs) and Capsule Networks to create an efficient and highly accurate model that can recognize handwritten digits in a variety of fonts, sizes, and orientations. Using CNNs' powerful feature extraction capabilities, the system first extracts important low- and mid-level features such as edges, curves, and textures from the input images. These features are then processed by Capsule Networks, which use vectorized representations and dynamic routing mechanisms to maintain spatial hierarchies and feature relationships. This enables the model to better understand digits' pose, orientation, and part-whole relationships, addressing key limitations of traditional CNNs. The use of Capsule Networks improves the system's resilience to variations such as rotation, scaling, skewing, and overlapping digits, which are common in real-world handwritten data. Furthermore, the model is intended to perform well even when confronted with noisy or distorted inputs, ensuring high reliability across a variety of handwriting styles. The end-to-end architecture will be trained on large datasets, using data augmentation techniques to simulate various transformations and improve generalization. The proposed system aims to provide a more comprehensive and adaptable solution for handwritten numeral recognition, making it suitable for practical applications in fields such as banking, postal services, healthcare, and automated form processing that require accuracy and robustness. By combining CNNs and Capsule Networks, the system aims to reduce misclassification and improve performance over existing models, resulting in more effective and intelligent handwriting recognition technologies.

- Integrating Capsule Networks with CNNs improves recognition accuracy by capturing spatial relationships between features, surpassing that of CNNs alone.

- The proposed model is robust to variations in handwriting styles, digit orientations, scales, and distortions, reducing errors from rotation, translation, and skewing.

- Capsule Networks preserve spatial hierarchies, allowing the system to understand digit structure and pose rather than just detecting isolated features.

- Because of dynamic routing and vectorized feature representations, the system performs well with unknown handwriting styles and noisy inputs, improving reliability across a wide range of datasets.

- Understanding feature orientation and arrangement reduces the system's likelihood of confusing digits with similar shapes (for example, '6' and '9', '1' and '7').
- The model is trained from start to finish, eliminating the need for manual feature engineering and allowing for the automatic learning of optimal features from raw image data.
- The hybrid architecture can be extended or adapted to recognize numerals in multiple scripts or languages, making it suitable for global applications.
- The system's increased robustness and accuracy make it suitable for practical applications like banking, postal services, healthcare record digitization, and automated form processing, all of which require handwritten digit recognition.

Table 3.3 Performance metrics

Performance Metrics	**Description**	**Value/Result**
Accuracy	Proportion of correctly classified digits	98%
Precision	Proportion of true positives out of all positive predictions	98.5%
Recall	Proportion of true positives out of all actual positives	97.8%
F1 Score	Harmonic mean of precision and recall	98.15%

CHAPTER 4

SYSTEM REQUIREMENT SPECIFICATIONS

4.1FUNCTIONAL REQUIREMENTS:

1) **Handwritten Digit Recognition:** Supports digit identification from 0 to 9. Ensures recognition n despite variations in handwriting style, size, or rotation.
2) **Image Preprocessing:** Includes noise reduction, grayscale conversion, and resizing.
3) **Model Training and Evaluation:** Uses MNIST dataset for supervised training. Implements accuracy, precision, and recall metrics for evaluation.
4) **Prediction and Feedback**: Provides probability scores for predictions.Includes visual feedback for predicted digit confirmation.
5) **Data Storage and Logging:** Stores model results, error rates, and improvement insights

4.2 NON-FUNCTIONAL REQUIREMENTS:

1. **Performance:** The system should perform well with little latency in prediction. It must be able to recognise and classify handwritten digits in real-time or near-real-time, particularly when integrated into applications such as automated form processing or banking systems. The average response time for a single prediction should not be more than a few milliseconds to ensure a smooth user experience and operational efficiency.

2. **Scalability:** The system must be designed to handle increasing volumes of input data while maintaining performance. Whether deployed on a local server, in the cloud, or on edge devices, the architecture should be scalable to allow for batch processing of large datasets or concurrent recognition tasks from multiple users. This ensures that it remains effective even as the application's scope or user base grows.

3. **Reliability and Accuracy:** High reliability is required for the system to be trusted in critical applications like banking, healthcare, and government data entry. The model must consistently make accurate predictions, with few instances of failure or misclassification. This includes maintaining high precision and recall across a wide range of handwriting

styles, image qualities, and digit orientations.

4. **Maintainability and Upgradability:** The system should be modular and well-documented to allow for easy maintenance, debugging, and upgrading. New models, datasets, or modules (for example, additional digit sets from different scripts) should be integrable without requiring significant changes to the existing architecture. This ensures long-term sustainability and adaptability as handwriting trends or application requirements change.

5. **Security and Data Privacy:** The system must ensure that all image data, particularly sensitive information such as bank checks, medical forms, or identification documents, is handled securely. Data encryption should be used during storage and transmission. If the system is deployed online or in the cloud, it should follow standard data protection regulations (such as GDPR or HIPAA, as applicable) to prevent unauthorized access or data breaches.

4.3 HARDWARE REQUIREMENTS:

1) **Processor (CPU):** Intel Core i5 or higher.
2) **RAM:** 16GB for efficient data processing and model training.
3) **Storage Device:** SSD (Solid State Drive) with at least 50GB of free space for datasets and models.
4) **Display Monitor:** Standard monitor for viewing results and debugging.
5) **High-Performance Computing Server:** For large-scale training and complex deep learning tasks.

4.4 SOFTWARE REQUIREMENTS:

1) **Programming Language**: Python
2) **Libraries & Frameworks:** TensorFlow, Keras, NumPy, Matplotlib, OpenCV
3) **Development Environment / Software:** Google Colab / Anaconda
4) **Operating System:** Windows / Linux / macOS

4.5 TECHNOLOGIES USED:

- **Python for development** – The entire project will be built using Python due to its extensive libraries and ease of use in machine learning.
- **TensorFlow and PyTorch** – These deep learning frameworks will be used to implement Convolutional Neural Networks (CNNs) and Capsule Networks (CapsNet) to improve digit recognition accuracy.
- **Flask / Streamlit** – A web interface will be developed using Flask or Streamlit to provide an interactive platform for users to input handwritten digits and get predictions.
- **SQLite / MongoDB** – A database system like SQLite or MongoDB will be used to store processed images, model outputs, and other relevant data for efficient management.

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECTURE:

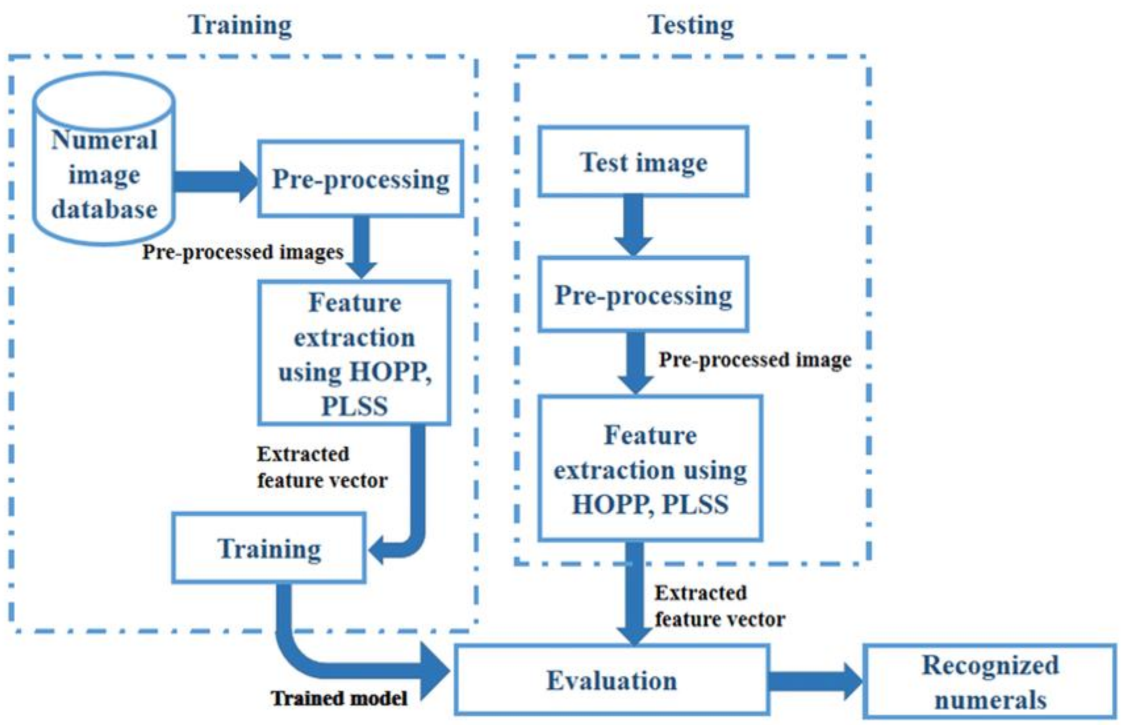


Fig. 5.1 System Architecture

5.2 SYSTEM MODULES:

1. **Image Preprocessing Module:** Converts raw images to grayscale for easier processing. Performs noise reduction and normalization to enhance image quality. Resizes images to a fixed dimension for efficient model input.

2. **Feature Extraction Module:** Uses Convolutional Neural Networks (CNNs) to identify key visual patterns in handwritten digits.

3. **Capsule Network Module:** Implements Capsule Networks (CapsNets) to understand spatial relationships between features. Enhances recognition accuracy for rotated, scaled, or distorted digits.

4. **Prediction Module:** Classifies extracted features into digits (0-9) using trained models. Uses a Softmax layer to generate probability scores for each digit.
5. **Database Module:** Stores processed digit data, model weights, and recognition results. Ensures fast retrieval for performance analysis and system improvement.
6. **User Interface (UI) Module:** Provides a web-based interface for users to upload handwritten digit images.Displays recognized digits along with confidence scores to ensure transparency.

5.3 DATA FLOW DIAGRAM:

The Data Flow Diagram (DFD) of the handwritten numeral recognition system depicts how data flows through various processing stages, from input to final output. At the first level, the system accepts raw image input from users, scanners, or digital devices that includes handwritten numerals. To ensure consistent input quality, this image data is routed to the Preprocessing Module and processed through operations such as grayscale conversion, resizing, noise reduction, and normalization. The pre-processed image is then sent to the CNN Feature Extraction Module, where convolutional and pooling layers extract local features such as edges, corners, and curves from the numerical data. These features are sent to the Capsule Network Layer, which encodes the spatial relationships between the components using vector representations and dynamic routing algorithms.

The Capsule Network refines the feature maps and determines the pose and orientation of the digit components. Following that, the data is sent to the Digit Classification Module, where the model's final output layer classifies the image into one of ten digits (0-9) and assigns a confidence score. The identified digit is then transferred to the Output Interface, where the result is presented to the user or routed to an external system (such as a database or application). The DFD ensures that each module processes data consecutively and quickly, allowing for precise, real-time handwritten numerical recognition that is robust to style and orientation variances.

Goals of DFD

- **Visualize the Flow of Data:** The DFD tries to clearly show how data moves through the handwritten numeral recognition system, from input picture acquisition to final digit classification, by creating a visual roadmap of data processing phases.

- **Define System Components and Boundaries:** It assists in identifying all major functional modules in the system, such as preprocessing, feature extraction, classification, and output creation, as well as defining the boundaries between internal processes and external entities (e.g., user, database, or application).
- **Identify Data Sources and Destinations:** The DFD identifies where data comes from (e.g., people or devices), how it is transformed, and where it ends up (e.g., output interface or data storage), ensuring that all data interactions are documented.
- **Clarify System Responsibilities:** Each component in the DFD is assigned a specific task—such as picture cleaning, feature extraction, or digit recognition—to assist stakeholders understand the system's duties and capability.
- **Support System Design and Development:** A well-structured DFD serves as a basis for system architecture planning, coding, and testing. It ensures that developers may create modular, maintainable code by knowing how different system components interact with one another.
- **Aid in Communication Among Stakeholders:** The DFD serves as a communication tool for technical and non-technical stakeholders, allowing them to debate, assess, and adjust the system's data processes without requiring extensive technical knowledge of the underlying machine learning models.
- **Identify Bottlenecks or Inefficiencies:** The DFD identifies potential flaws, such as data redundancy, processing delays, or poor data handling, that may influence system performance or accuracy.

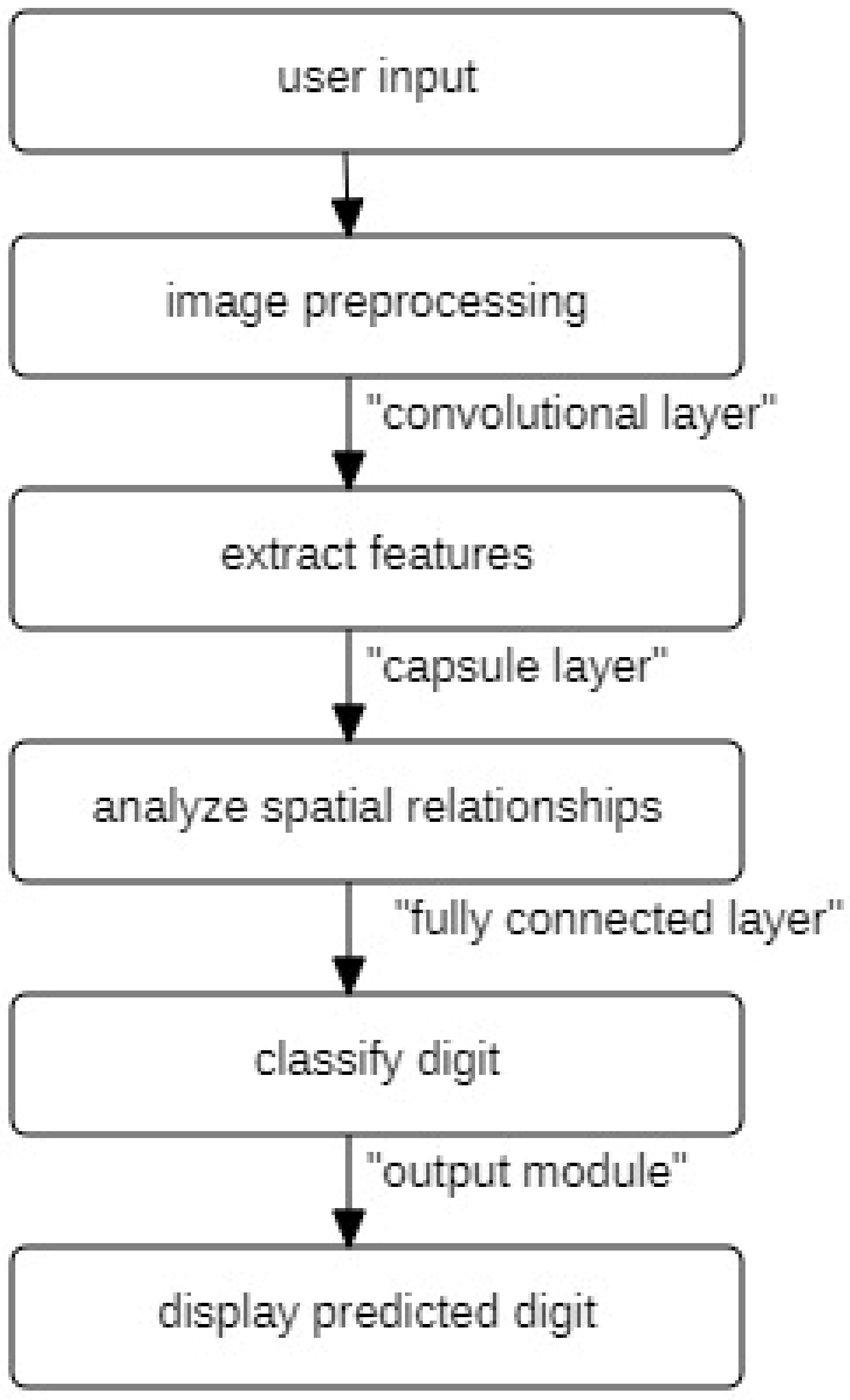


Fig. 5.3 Data Flow Diagram

5.4 UML DIAGRAMS:

The UML (Unified Modeling Language) diagram for the handwritten number identification system based on CNNs and Capsule Networks depicts the system's structure and behavior by representing its essential components, linkages, and interactions. The class diagram shows the important classes: ImageInput, Preprocessor, CNNModule, CapsuleNetwork, Classifier, and UserInterface. Each class has important properties and methods—for example, the Preprocessor class has methods like resizeImage(), normalizePixels(), and removeNoise(), whereas the CNNModule class provides the convolution and pooling processes used in feature extraction. The CapsuleNetwork class encapsulates the dynamic routing algorithm and vectorized output generation, which are required for expressing spatial relationships in digit structures. The Classifier class contains logic that generates the anticipated digit and confidence score. The sequence diagram depicts the dynamic flow of processes, beginning with a user submitting a picture, followed by the system doing preprocessing, feature extraction, capsule routing, classification, and result presentation. Furthermore, the activity diagram depicts the process logic, emphasizing conditional flows (for example, if image quality is bad, reprocess or reject input).

Goals of UML

- **Visualize System Architecture Clearly:** UML diagrams are standardized visual representations of a number recognition system's structure and behavior. This allows developers, designers, and stakeholders to understand how components such as CNNs, Capsule Networks, preprocessing units, and classification modules are organized and interact.

- **Support Object-Oriented Design:** UML diagrams are useful for defining classes, properties, methods, and relationships using object-oriented principles. To facilitate modularity and reuse, the system's modules (for example, preprocessing, feature extraction, and classification) might be described as classes.

- **Simplify Complex System Interactions**: Complex workflows like image input, feature extraction, dynamic routing, and digit prediction can be broken down into intelligible interactions and processes using diagrams, particularly sequence and activity diagrams.

- **Facilitate Communication Among Stakeholders:** UML provides a single language for technical and non-technical stakeholders. It closes the gap between system requirements and implementation, ensuring that everyone involved in the project has a common knowledge of the system's design.

- **Assist in System Development and Maintenance:** UML diagrams serve as blueprints for system creation and future enhancement by explicitly describing the roles and interactions of each component. They help new developers comprehend the architecture fast and make debugging and upgrading easier.

- **Identify Dependencies and Responsibilities:** UML diagrams show how different pieces of the system interact with one another and which components are responsible for specific activities. This is critical for successful system design, particularly when including machine learning models such as CNNs and Capsule Networks.

- **Improve Documentation and Planning:** UML diagrams provide formal system documentation and are valuable for project planning, version control, and long-term system administration. They ensure that architectural decisions are documented and maintained throughout the software's lifecycle.

5.4.1 Use Case Diagram:

The Use Case Diagram for the handwritten numeral identification system based on CNNs and Capsule Networks depicts the interactions between external actors and the system's functional components. The diagram's main player is the User, who interacts with the system by uploading handwritten digit images, viewing recognition results, and potentially offering feedback. The system's main use cases are Image Preprocessing, which cleans and resizes uploaded images for consistent input; Feature Extraction via CNN, which identifies essential visual patterns such as edges and curves; Digit Recognition using Capsule Network, which encodes spatial hierarchies and classifies the digit; and Results Display, which shows the user the recognized digit and its confidence score.

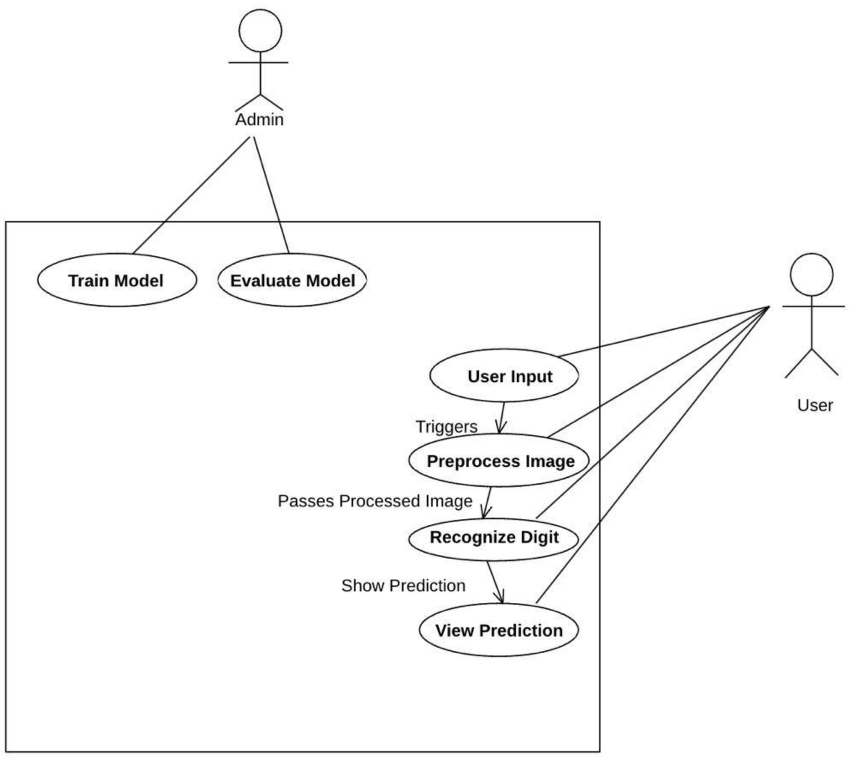


Fig. 5.4.1 Use Case Diagram

5.4.2 Class Diagram:

The Class Diagram for the handwritten numeral recognition system depicts the application's static structure by outlining its core classes, characteristics, methods, and relationships. The basic class is ImageInput, which accepts and stores picture data and has characteristics like imageID, format, and resolution, as well as methods like loadImage() and validateInput(). The Preprocessor class performs actions such as resizeImage(), convertToGrayscale(), and normalizeData() to prepare the image for subsequent processing. The CNNModule class is intended to execute feature extraction using attributes such as numLayers, kernelSize, and activationFunction, as well as methods like apply Convolution() and poolFeatures().

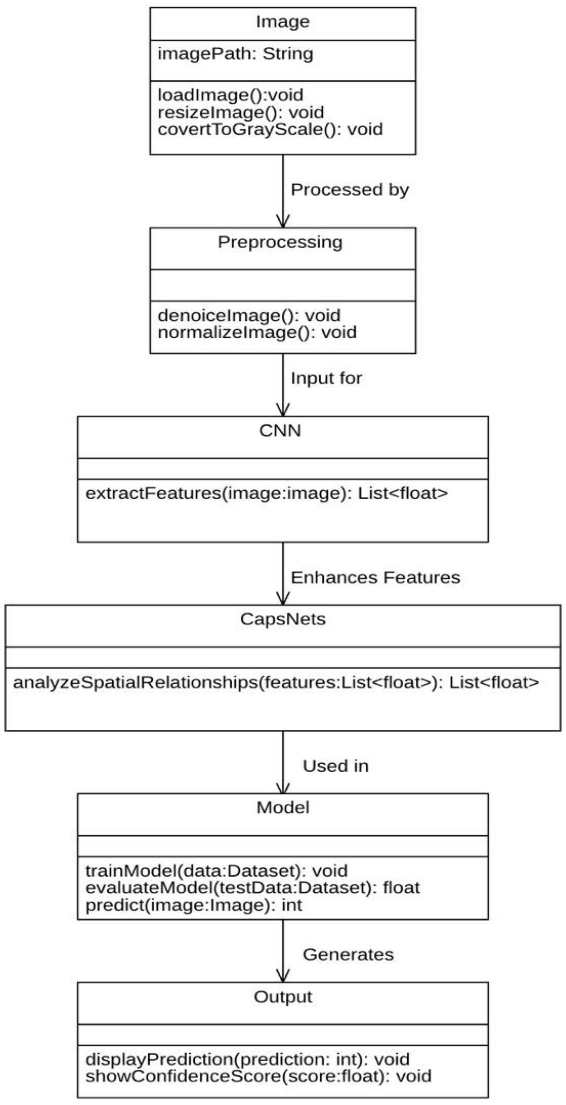


Image
imagePath: String
loadImage():void
resizeImage(): void
covertToGrayScale(): void
Processed by
Preprocessing
denoiceImage(): void
normalizeImage(): void
Input for
CNN
extractFeatures(image:image): List<float>
Enhances Features
CapsNets
analyzeSpatialRelationships(features:List<float>): List<float>
Used in
Model
trainModel(data:Dataset): void
evaluateModel(testData:Dataset): float
predict(image:Image): int
Generates
Output
displayPrediction(prediction: int): void
showConfidenceScore(score:float): void

Fig. 5.4.2 Class Diagram

5.4.3 Sequence Diagram:

The Sequence Diagram for the handwritten numeral recognition system depicts the step-by-step interaction and message flow between system components throughout time, emphasizing the sequential order in which actions are carried out. The sequence starts with the user starting the interaction by uploading a handwritten numerical image via the User Interface. This input is routed to the ImageInput module, which then sends it to the Preprocessor. The Preprocessor is responsible for activities like as scaling, grayscale conversion, and normalization, which ensure that the picture data is clean and consistent. The preprocessed image is delivered to the CNNModule, which uses convolution and pooling to extract spatial information. These feature maps are then sent to the CapsuleNetwork, where dynamic routing and vector-based spatial hierarchies are used to collect digit component pose and orientation. The output vectors from the capsules are fed into the DigitClassifier, which guesses the digit and computes a confidence score. This result is returned to the User Interface for presentation.

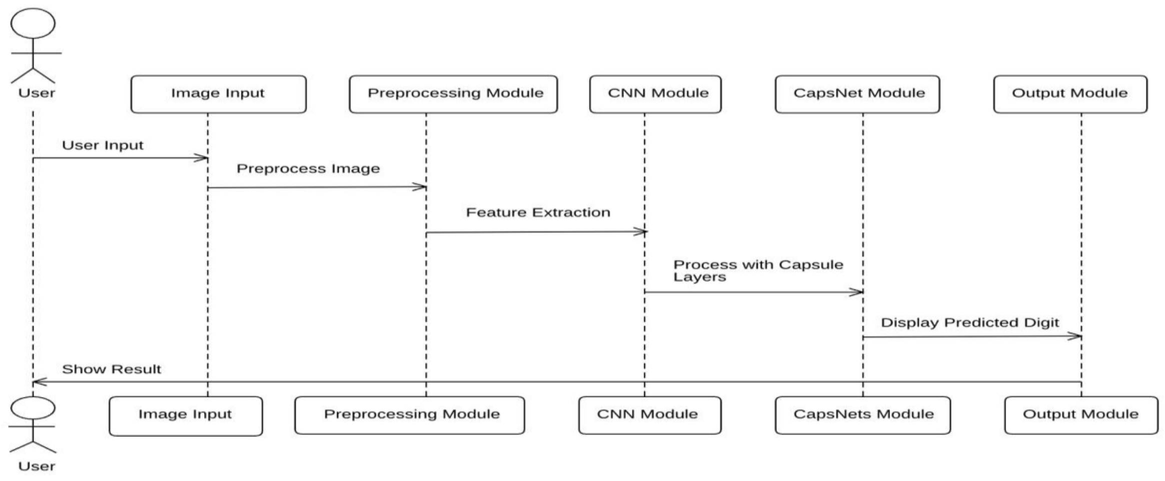


Fig. 5.4.3 Sequence Diagram

5.4.4 Collaboration Diagram:

The Collaboration Diagram for the handwritten number recognition system emphasizes the structural arrangement and interaction of various system components by emphasizing on how objects work together to complete the digit identification process. The User Interface object, located in the center of the diagram, commences the process by delivering the handwritten digit image to the ImageInput object. The image is then handed to the Preprocessor object, which uses its internal methods to perform image normalization, scaling, and noise reduction. After preprocessing is complete, the CNNModule object is triggered to extract low-level features using convolution and pooling techniques. These attributes are then sent to the CapsuleNetwork object, which employs dynamic routing techniques to detect complicated geographical relationships.

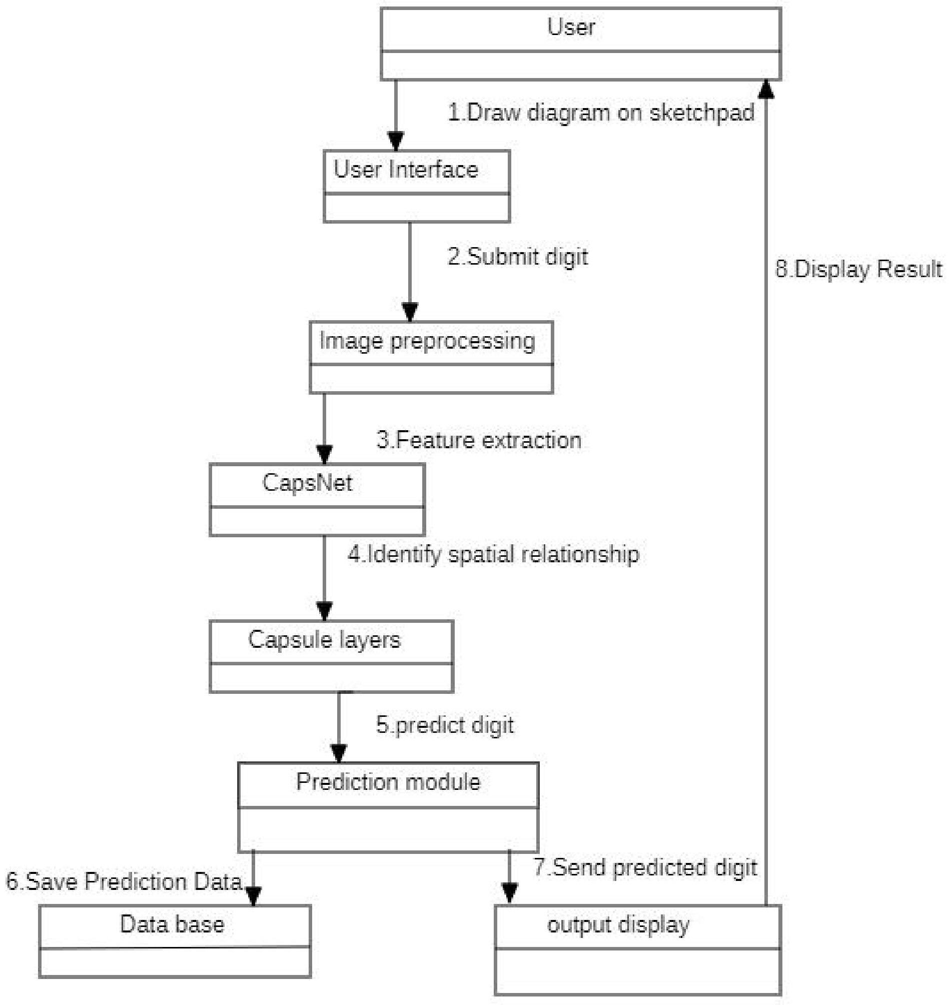


Fig. 5.4.4 Collaboration Diagram

5.4.5 Activity Diagram:

The Activity Diagram for the handwritten numeral recognition system depicts a dynamic perspective of the workflow involved in processing and classifying handwritten numbers. The action begins when the user uploads a digit image via the user interface. The system then moves on to the image preprocessing phase, when tasks such as resizing, grayscale conversion, and noise reduction are undertaken to prepare the image for analysis. After preprocessing, control is passed to the feature extraction step, which uses the CNN module to extract essential visual patterns such as curves, edges, and strokes. The retrieved features are then sent to the Capsule Network stage, which captures spatial hierarchies and encodes the orientation and interactions between various portions of the finger.

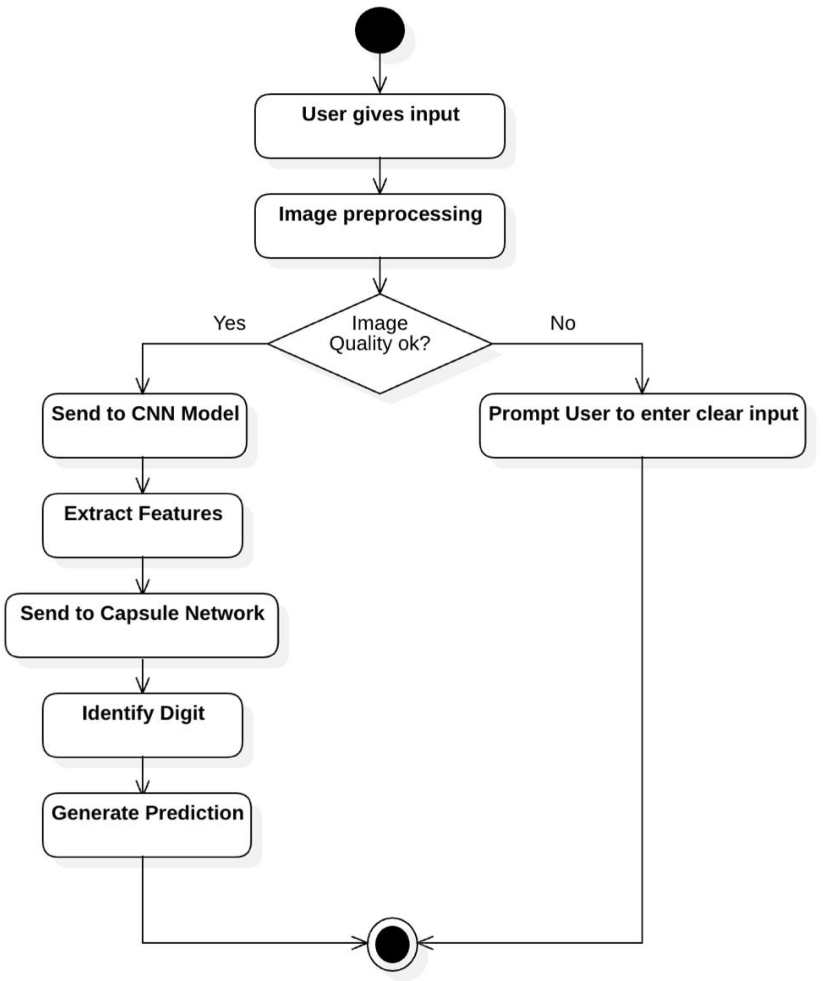


Fig. 5.4.5 Activity Diagram

5.4.6 State Chart Diagram:

The State Chart Diagram for the handwritten number recognition system depicts the numerous stages that the system goes through during the digit recognition process, as well as the transitions caused by user actions or internal events. System Idle is the application's initial state, in which it waits for user interaction. When a user uploads a handwritten digit image, the system moves to the Image Received state. It next enters the Preprocessing stage, where the image is resized, grayscale converted, and normalized. Following successful preprocessing, the system enters the Feature Extraction state, which activates the CNN module to detect visual patterns. The next step is Capsule Encoding, which captures spatial hierarchies and orientations using the Capsule Network.

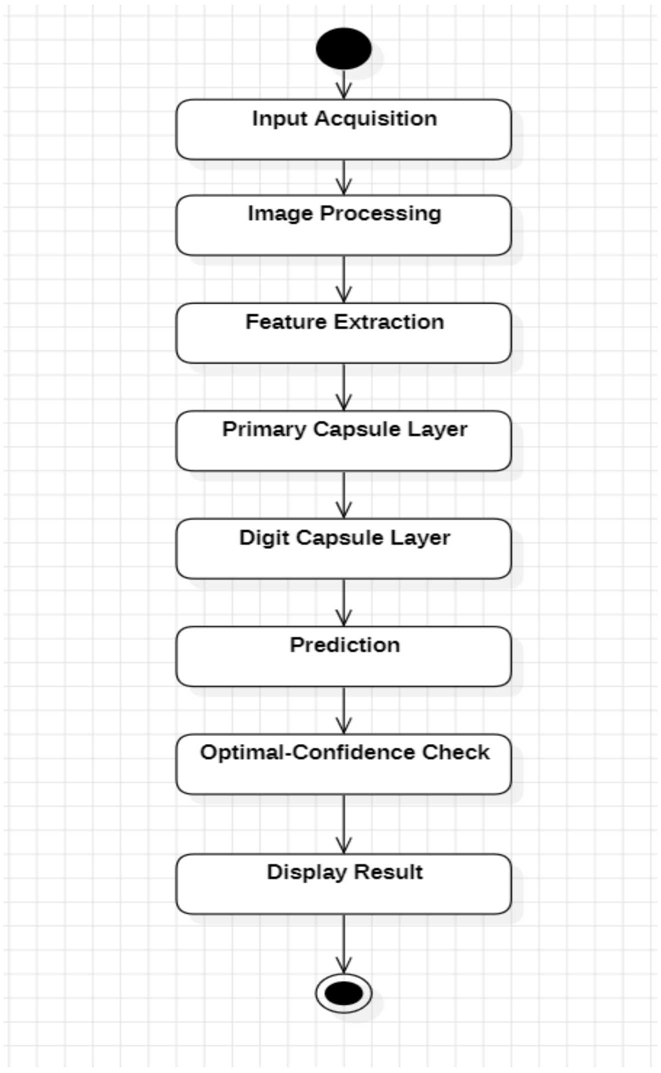


Fig. 5.4.6 State Chart Diagram

5.4.7 Component Diagram:

The Component Diagram for the handwritten numeral recognition system depicts the system's primary components, internal roles, and interdependencies, providing a high-level overview of the software architecture. The system is built around numerous interrelated components, including the User Interface, Preprocessing, CNN Feature Extraction, Capsule Network, Digit Classification, and an optional Database component for storing and logging. The User Interface enables users to upload digit photos and obtain recognition results. This component connects with the Preprocessing module, which uses scaling, grayscale conversion, and normalization to clean and standardize the input. The result is sent to the CNN component, which uses convolutional filters and pooling layers to extract important visual information.

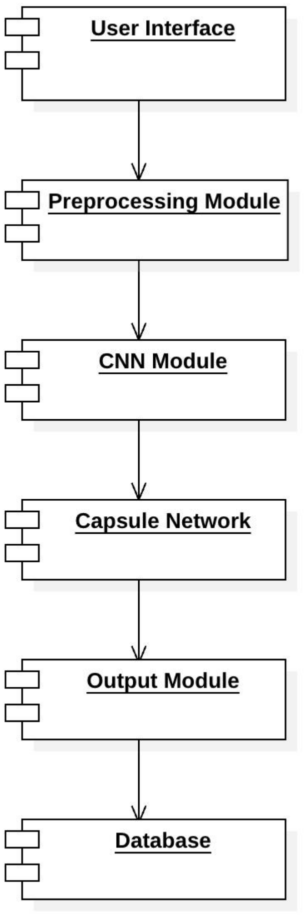


Fig. 5.4.7 Component Diagram

5.4.8 Deployment Diagram:

The Deployment Diagram for the handwritten numeral recognition system depicts the system's physical architecture by mapping software components to the hardware nodes that handle their execution. The system is often distributed among numerous nodes, such as a Client Device, a Web/Application Server, and a Model Server or Machine Learning Backend. The Client Device node, such as a user's browser or mobile app, houses the User Interface component, which allows users to upload images and view recognition results. This client connects to the Web/Application Server, which handles API calls, manages user interactions, and routes image data for processing.

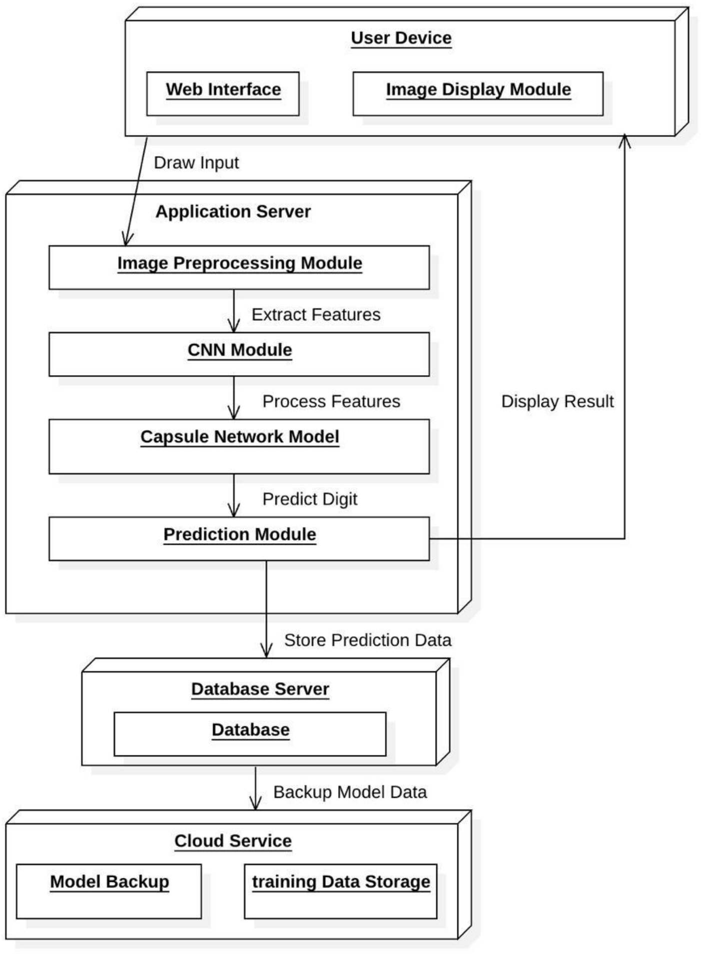


Fig. 5.4.8 Deployment Diagram

CHAPTER 6

IMPLEMENTATION

6.1 ALGORITHMS USED:

- **Convolutional Neural Network (CNN) Algorithm**: CNNs are deep learning models that handle visual input. They use a series of convolution operations to scan the input image using filters (kernels) and discover patterns such as edges and curves.

  - **Convolution layers** for feature extraction
  - **ReLU (Rectified Linear Unit)** as the activation function
  - **Pooling layers** (e.g., Max Pooling) for downsampling
  - **Fully connected layers** for classification

- **Capsule Network (CapsNet) Algorithm:** Capsule Networks are an advanced neural architecture introduced to overcome limitations of CNNs, especially in recognizing rotated or overlapping digits.

  - **Capsules**: Groups of neurons outputting vectors instead of scalars. These vectors represent features along with spatial information (such as orientation and pose).
  - **Dynamic Routing Algorithm**: Unlike fixed pooling in CNNs, CapsNets use a routing-by-agreement process that determines which higher-level capsule should receive the output of a lower-level capsule.

- **ReLU (Rectified Linear Unit) Activation Function:** Used in both CNNs and CapsNets, ReLU introduces non-linearity by setting negative values to zero and retaining positive values. It accelerates training and avoids vanishing gradients.

- **Softmax Classifier :** The final layer in most CNN and CapsNet architectures is a Softmax function, which converts the network output into a probability distribution across digit classes (0–9), selecting the most likely digit as the prediction.

6.2 SAMPLE CODE :

1. Model Setup and Loading:

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = MnistCapsuleModel().to(device)

model.load_state_dict(torch.load("mnist_capsule1.pt", map_location=device))

model.eval()
```

2. Image Preprocessing Function:

```
def preprocess_image(image_data):

    try:

        image_bytes = base64.b64decode(image_data.split(',')[1])

        image = Image.open(io.BytesIO(image_bytes)).convert('L')

        image = image.resize((128, 128), Image.Resampling.LANCZOS)

        image = transforms.ToTensor()(image)

        image = (image < 0.5).float()

        non_zero = image.squeeze().nonzero(as_tuple=False)

        if non_zero.size(0) == 0:

            return None

        top_left = non_zero.min(0)[0]

        bottom_right = non_zero.max(0)[0]

        bbox = image[:, top_left[0]:bottom_right[0]+1, top_left[1]:bottom_right[1]+1]

        bbox = transforms.Resize((20, 20))(bbox)
        padded = torch.zeros(1, 28, 28)

        padded[:, 4:24, 4:24] = bbox

        padded = (padded - 0.1307) / 0.3081

        return padded.unsqueeze(0).to(device)

    except:

return None
```

3. Prediction Route (Flask):

```
@app.route("/predict", methods=["POST"])

def predict():

    image_data = request.form["imageData"]

    input_tensor = preprocess_image(image_data)

    if input_tensor is None:

        return render_template("result.html", prediction="Please enter a digit", image_data=None)

    output = model(input_tensor)

    prediction = output.argmax(dim=1).item()

    return render_template("result.html", prediction=prediction, image_data=image_data)
```

4. CapsNet Logic:

```
def squash(x):

    return (x.norm(dim=-1, keepdim=True)*2 / (1 + x.norm(dim=-1, keepdim=True)*2)) * (x / x.norm(dim=-1, keepdim=True))

class PrimaryCaps(nn.Module):
    def forward(self, x):

        x = conv(x)  # Conv to create capsules

        x = x.view(batch, -1, dim)  # Shape: [batch, capsules, dim]

        return squash(x)

class DigitCaps(nn.Module):

    def forward(self, x):

        u_hat = transform(x)  # Predict outputs

        for _ in range(routing_iters):

            c = softmax(b)  # Routing weights

            s = (c * u_hat).sum(dim=1)

            v = squash(s)

            b += (u_hat * v).sum(-1)

        return v
```

CHAPTER 7

SYSTEM TESTING

7.1 INTRODUCTION TO TESTING:

Testing is an important step in developing a handwritten numeral recognition system, especially one that uses Convolutional Neural Networks (CNNs) and Capsule Networks. It entails methodically assessing the performance, accuracy, and robustness of trained models against unseen data. In this case, testing guarantees that the system can accurately generalize to new handwritten numbers, regardless if they differ in style, orientation, size, or stroke quality. Testing often use a different test dataset, such as the MNIST or EMNIST datasets, which contain a wide range of handwritten digits. Performance indicators like as accuracy, precision, recall, F1-score, and confusion matrix are used to evaluate how well the CNN and CapsNet models detect and categorize digits.

Furthermore, robustness testing may include rotated, shifted, or noisy digits to assess the system's resistance to real-world fluctuations. In this system, CNNs are assessed for their capacity to extract significant information, whereas CapsNets are examined for their superior spatial awareness and orientation management. Overall, testing validates the efficiency of the combined deep learning technique and guarantees that the model meets application accuracy and reliability requirements.

7.2 TESTING STRATEGIES:

The following are the common testing strategies:

System Testing: System testing involves evaluating the **entire numeral recognition system** as a whole to verify that it meets the specified requirements. It checks the complete interaction between components—input image processing, feature extraction (CNN), spatial analysis (CapsNet), and final classification. This end-to-end testing ensures that the user can upload a handwritten digit image and receive an accurate prediction with a seamless user experience.

Black-Box Testing: In black-box testing, the internal workings of the system are not considered. The tester provides inputs (digit images) and observes outputs (predicted digits) to ensure correctness. This is particularly useful for validating the recognition results without knowing how CNNs and CapsNets operate internally. Testers check whether different handwritten digits, in various styles, are classified correctly.

Robustness Testing: Robustness testing evaluates how the system behaves under **unusual or degraded conditions**—such as noisy, rotated, blurred, or incomplete digit images. Since CapsNets are designed to better handle such challenges compared to CNNs alone, this testing ensures the model maintains high accuracy even when faced with poor-quality inputs or variations in writing style.

Boundary Testing: Boundary testing involves checking the system's performance at the edges of acceptable input. Examples include: Very small or very large digit images, Very thick or very thin strokes, Digits written at the edge or corner of the input canvas.

7.3 TEST CASES:

Table 7.2 Test Cases

S.no.	Test Cases	Input	Expected Outcomes	Status
1	Clear Digit	"Digit 3 drawn clearly"	3	Pass
2	Blank Canvas	"No Digit Drawn"	Please enter a Digit	Pass
3	Faint Scribbble	"Unclear random strokes"	Please enter a Digit	Fail

4	Thick Digit Edge	“Digit 8 Near Borders”	8	Pass
5	Digit with Noise	“Digit 1 with extra marks”	1	Pass

7.3 RESULTS AND DISCUSSION:

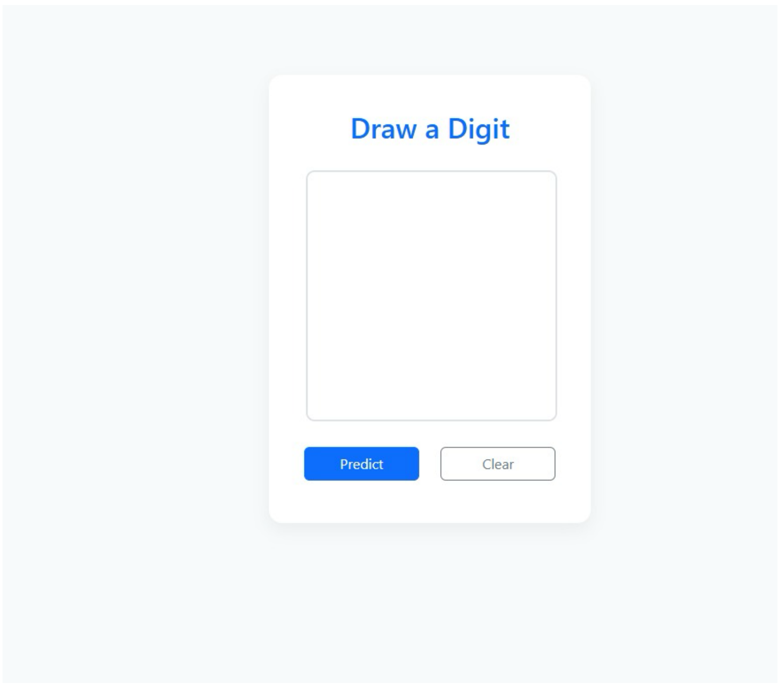


Fig. 7.3.1 Draw Digit

This page shows a sketch pad in which we can draw a digit contains two button “Predict” and “Clear”.

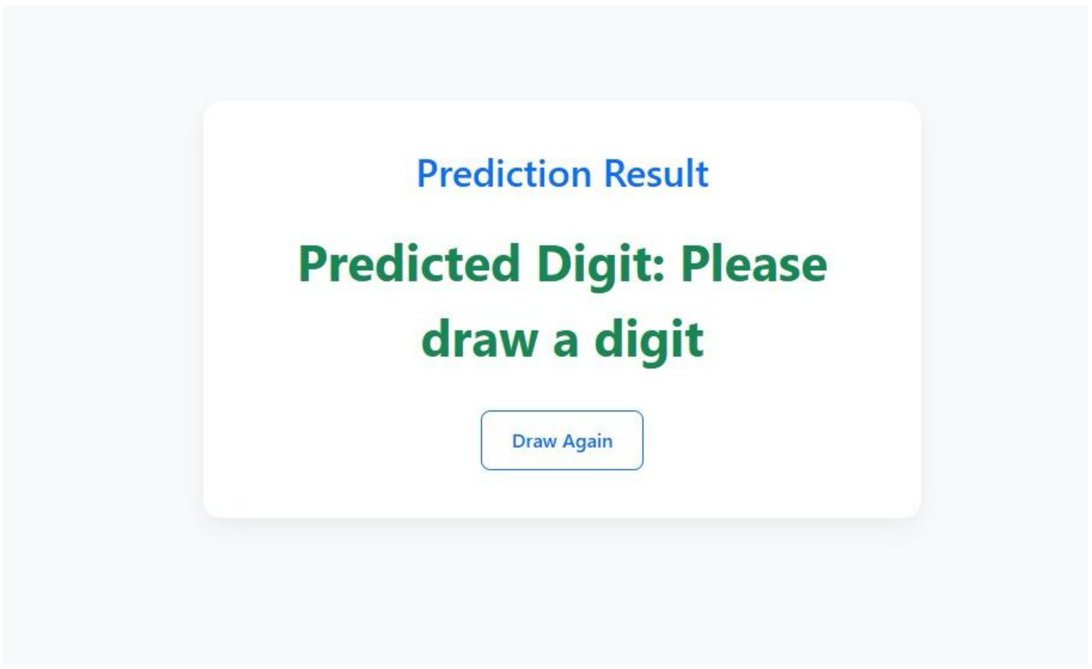


Fig. 7.3.2 Predicting Result

This page shows that if no digit is drawn it gives as "Please draw a digit" and it contains a button "Draw again".

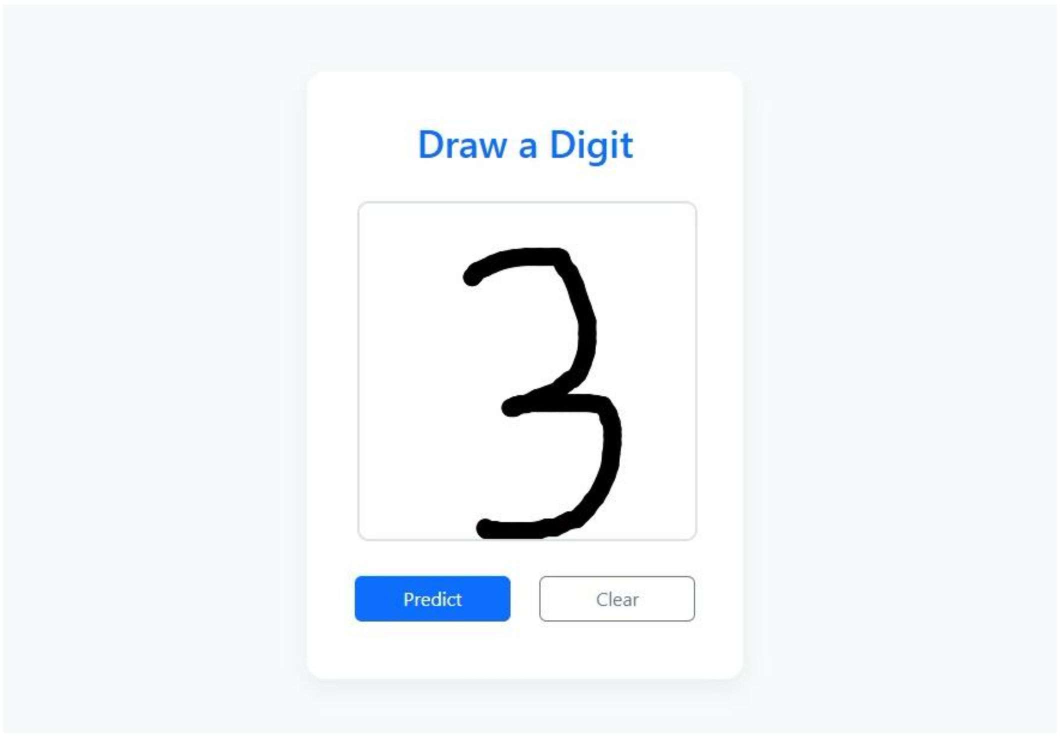


Fig. 7.3.3 Digit "3" drawn clearly

This is a page where we drawn a digit "3" on the sketch pad.

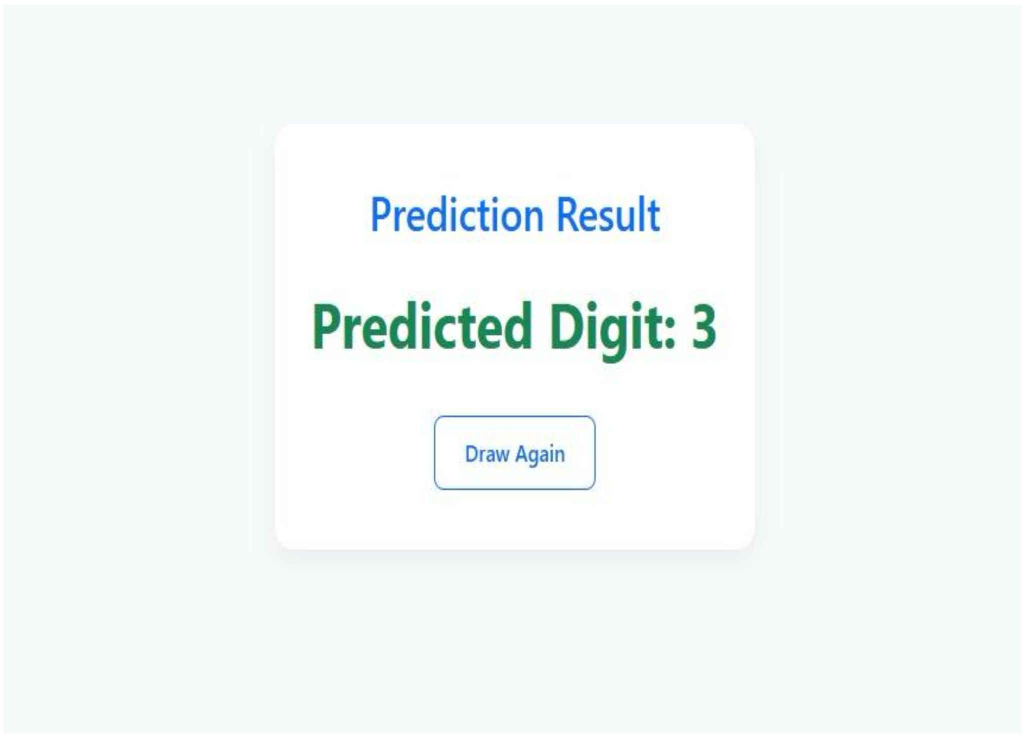


Fig. 7.3.4 Predicted digit as “3”.

This is a predicting page allows users to know the number which is drawn.

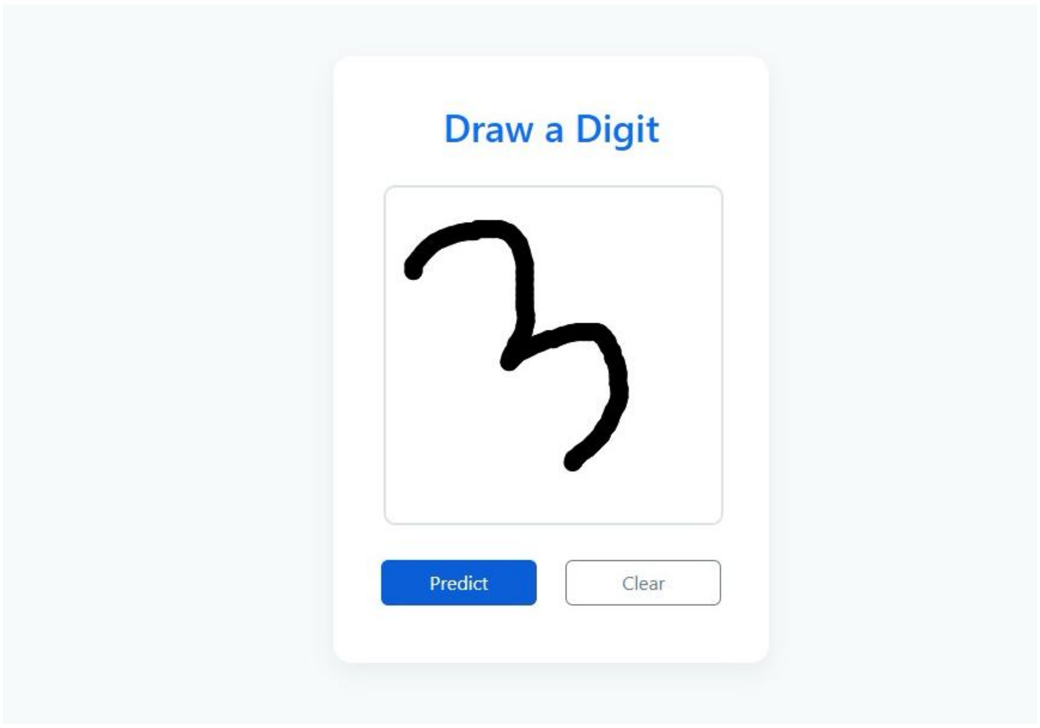


Fig. 7.3.5 Digit “3” tilted to left

In the above figure, the user can also draw the digit in titled manner.

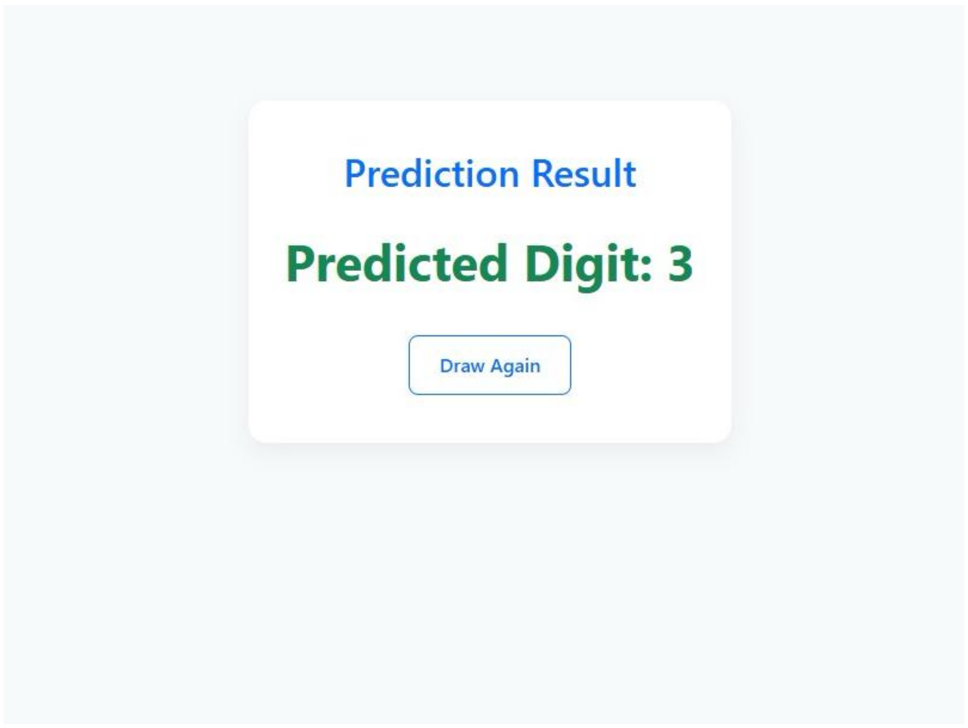


Fig. 7.3.6 Predicted tilted digit as “3”.

In the above figure, it predicted the titled digit and shown as “predicted digit:3”

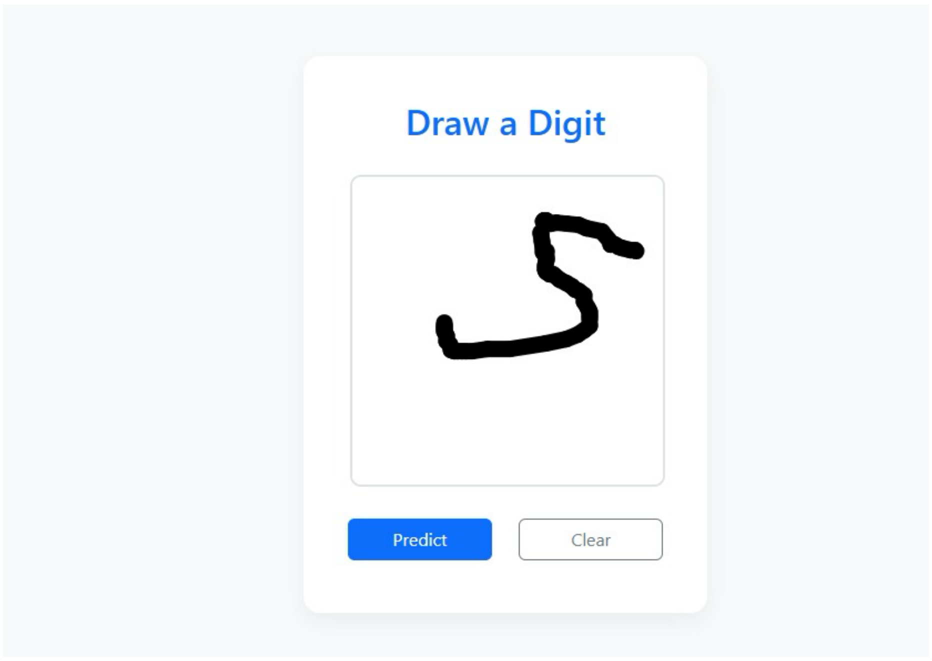
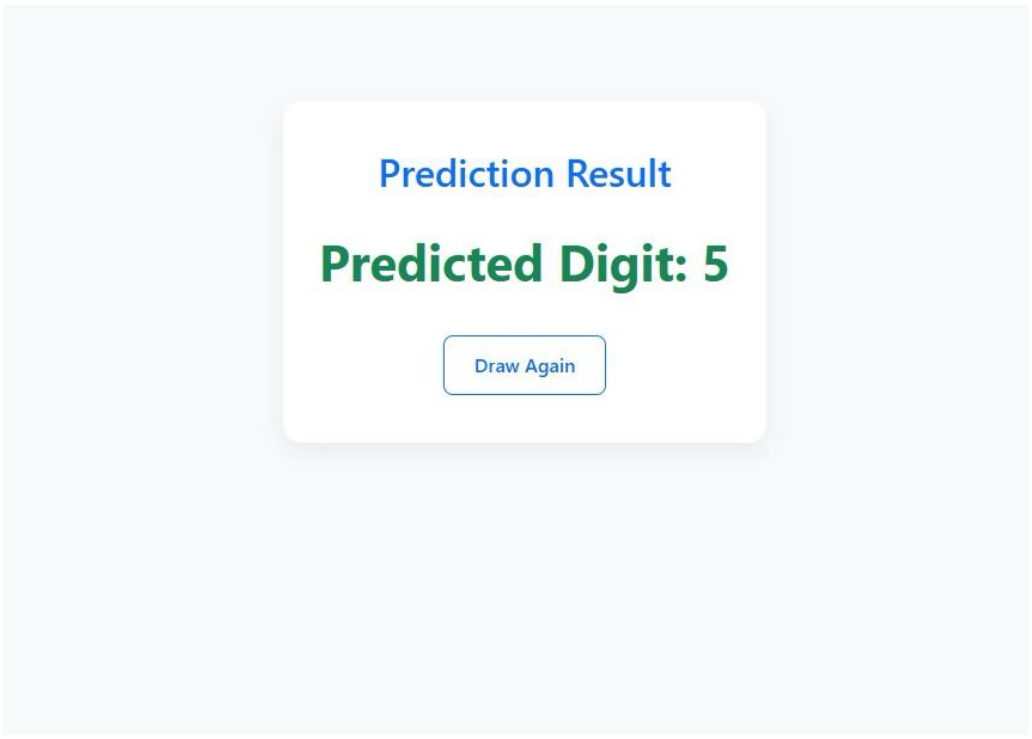


Fig. 7.3.7 Digit “5” tilted to right

In the above figure, the user can also draw the digit in titled manner.


Prediction Result

Predicted Digit: 5

Fig. 7.3.8 Predicted tilted digit as “5”.

In the above figure, it predicted the titled digit and shown as “predicted digit:5”

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENT

8.1 CONCLUSION:

In conclusion, handwritten number recognition utilizing Convolutional Neural Networks (CNN) and Capsule Networks (CapsNet) is a huge step forward in the field of image-based pattern recognition. CNNs have long been the industry standard for image classification tasks due to its capacity to automatically learn and extract local features via convolutional and pooling layers. They are computationally efficient and work well with large-scale datasets like MNIST. However, due to pooling processes, CNNs frequently lose spatial hierarchies, reducing their capacity to distinguish objects when rotated, scaled, or deformed. Capsule Networks, designed to address these limitations, preserve spatial links between features by grouping neurons into capsules and employing dynamic routing algorithms to maintain the hierarchical structure of parts and wholes. This makes CapsNet more adaptable to changes in orientation and perspective, resulting in a more accurate depiction of the digit structures. When compared, CNNs tend to generalize well given enough data, whereas CapsNet performs better with fewer samples and can handle complex transformations more well. Combining the two technologies, either in simultaneously or through hybrid architectures, can take advantage of their respective strengths—CNN's feature extraction efficiency and CapsNet's spatial awareness. This integration improves the system's robustness and identification accuracy, especially in noisy or uncontrolled contexts. Overall, number identification combining CNN and CapsNet is a promising technique for real-world applications such as automated data entry, bank check processing, postal sorting, and educational technology, opening the door to more sophisticated and context-aware visual recognition systems.

8.2 FUTURE ENHANCEMENT:

Future improvements to handwritten numeral recognition using CNN and CapsNet can concentrate on increasing accuracy, robustness, and real-world applicability. One possible approach is to include attention mechanisms or transformer-based modules into CNN or CapsNet structures, allowing the model to focus more effectively on relevant regions of the input image, particularly in the presence of noisy or overlapping digits. Furthermore, the creation of hybrid models that combine CNNs' rapid feature extraction with CapsNet's ability to preserve spatial hierarchies can result in more accurate and interpretable results. Another area for improvement is to increase the system's ability to generalize across different handwriting styles and writing instruments by incorporating larger and more diversified datasets, such as multilingual and multi-script numeric datasets. Techniques like as data augmentation, transfer learning, and few-shot learning could help to increase performance in low-data circumstances. Edge computing and model compression approaches may make these models more suited for deployment on mobile or embedded devices, allowing offline numeral identification in real time. Finally, incorporating feedback mechanisms for online learning, in which the model improves over time based on user corrections, as well as extending the system to handle complete handwritten text (alphanumeric or symbolic), would significantly broaden its capabilities and impact in educational tools, digital forms, and accessibility technologies.

CHAPTER 9

REFERENCES

[1] **Chen, F., Luo, Z., Chen, N., Mao, H., Hu, H., Jiang, Y., Pan, X., & Zhang, H.** (2024). *Assessing Four Neural Networks on Handwritten Digit Recognition Dataset (MNIST)*. Journal of Computer Science Research, 6(3), 17–22.

[2] **Tarade, S., Tanpure, S., Pawar, S., & Puntambekar, S.** (2024). *Innovative CNN Strategies for Superior Handwritten Digit Recognition*. International Journal of VLSI Circuit Design & Technology, 2(1), 27–34.

[3] **Ullah, S. S., Gang, L., Riaz, M., Ashfaq, A., Khan, S., & Khan, S.** (2025). *Handwritten Digit Recognition: An Ensemble-Based Approach for Superior Performance*. arXiv preprint arXiv:2503.06104.

[4] **Du, Z., Chen, P., & Miao, H.** (2022). *Design and Application of Handwritten Numeral Recognizer Based on Convolutional Neural Network*. In G. A. Tsihrintzis, S. J. Wang, & I. C. Lin (Eds.), *2021 International Conference on Security and Information Technologies with AI, Internet Computing and Big-data Applications* (pp. 267–275). Springer.

[5] **Roy, P., Ghosh, S., & Pal, U.** (2023). *A CNN Based Framework for Unistroke Numeral Recognition in Air-Writing*. arXiv preprint arXiv:2303.07989.

[6] **Kushwaha, K., Rahul, S., Eliyaz, S., Reddy, C., & K, A.** (2024). *CNN-Based Image Classification for Handwritten Digit Recognition*. International Journal of Intelligent Systems and Applications in Engineering, 12(14s), 91–97.

[7] **Islam, M., Shuvo, S. A., Nipun, M. S., Sulaiman, R. B., Nayeem, J., Haque, Z., Shaikh, M. M., & Sourav, M. S. U.** (2022). *Efficient approach of using CNN based pretrained model in Bangla handwritten digit recognition*. arXiv preprint arXiv:2209.13005.

[8] **Authors** (2024). *Advanced OCR for Digits Exploring CNN for Optimal Performance*. Procedia Computer Science.