VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

MINI PROJECT REPORT
on

“HANDWRITTEN NUMERIC CHARACTER RECOGNITION USING DEEP LEARNING”

Submitted by

PRANAV GAJANAN KAMATE (1BM23CS241)
PRASOBH RATNA SHAKYA (1BM23CS243)
RISHAB M JAIN (1BM23CS267)
SAHASRA MUSALIKUNTA (1BM23CS284)

Under the Guidance of
Dr. Shyamala G
Professor, BMSCE

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B. M. S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019

September 2025 to January 2026

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the project work entitled "**HANDWRITTEN NUMERIC CHARACTER RECOGNITION USING DEEP LEARNING**" carried out by **PRANAV GAJANAN KAMATE (1BM23CS241), PRASOBH RATNA SHAKYA (1BM23CS243), RISHAB M JAIN (1BM23CS267), SAHASRA MUSALIKUNTA (1BM23CS284)** who are bonafide students of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visveswaraiah Technological University, Belgaum during the year 2025-2026. The project report has been approved as it satisfies the academic requirements in respect of **Mini Project (23CS5PWMIP)** work prescribed for the said degree.

Signature of the Guide	Signature of the HOD
Dr. Shyamala G	Dr. Kavitha Sooda
Professor	Professor & Head, Dept. of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

External Viva

Name of the Examiner	Signature with date
1.________________________________	___________________________
2. ________________________________	____________________________

B. M. S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECALARATION

We, **PRANAV GAJANAN KAMATE (1BM23CS241), PRASOBH RATNA SHAKYA (1BM23CS243), RISHAB M JAIN (1BM23CS267), SAHASRA MUSALIKUNTA (1BM23CS284)**, students of 5th Semester, B.E, Department of Computer Science and Engineering, B. M. S. College of Engineering, Bangalore, here by declare that, this Project Work-1entitled " **HANDWRITTEN NUMERIC CHARACTER RECOGNITION USING DEEP LEARNING**" has been carried out by us under the guidance of Prof. Namratha M, Assistant Professor, Department of CSE, B. M. S. College of Engineering, Bangalore during the academic semester September 2025-January 2026.

We also declare that to the best of our knowledge and belief, the development reported here is not from part of any other report by any other students.

Signature

PRANAV GAJANAN KAMATE (1BM23CS241)

PRASOBH RATNA SHAKYA (1BM23CS243)

RISHAB M JAIN (1BM23CS267)

SAHASRA MUSALIKUNTA (1BM23CS284)

1. Introduction

1.1. Motivation

In today's digital era, the transition from physical to digital documentation is accelerating. However, a significant volume of data—ranging from bank cheques and postal addresses to medical prescriptions and academic forms—remains handwritten. Manual data entry from these documents is labor-intensive, time-consuming, and prone to human error. The motivation behind this project is to bridge the gap between analog handwriting and digital systems. By automating the recognition of handwritten characters, we can significantly enhance operational efficiency in various sectors such as banking, healthcare, and logistics.

1.2. Scope of the Project

The scope of this project involves developing a Deep Learning model capable of recognizing offline handwritten alphanumeric characters. The system is designed to take static images of handwriting as input and convert them into machine-readable text. The project utilizes the EMNIST (Extended MNIST) dataset for training, specifically focusing on the digits and character splits. The solution includes an image preprocessing pipeline to handle noise and segmentation, a Convolutional Neural Network (CNN) for classification, and a graphical user interface (GUI) using Gradio for real-time demonstration.

1.3. Problem Statement

Handwriting varies significantly between individuals in terms of stroke width, slant, size, and alignment. Traditional Optical Character Recognition (OCR) systems often fail to generalize across these variations, leading to poor accuracy on unconstrained handwriting. The problem addressed in this project is the accurate classification of handwritten characters from static images. The objective is to build a system that preprocesses raw images to isolate individual characters and classifies them using a robust CNN architecture with high accuracy.

2. Literature Survey

1. **Handwritten Digit Recognition Based on Deep Learning Algorithms** *Lv, Xue. (2023). 2023 International Conference on Internet of Things, Robotics and Distributed Computing (ICIRDC).*

   1. **Problem Statement & Objective**

   The paper points out that traditional recognition methods—such as template matching—fail to handle the huge variation in human handwriting. To overcome this, the authors explore the use of Deep Learning, especially Convolutional Neural Networks (CNNs), to improve the accuracy and efficiency of handwritten digit recognition. This has practical uses in form processing, automation, verification systems, and security applications.

2. Methodology & Algorithms

The study goes beyond classical image processing and evaluates several modern deep learning architectures.

Core Model: CNNs

The paper identifies CNNs as the most effective approach. A standard CNN pipeline includes:

- **Input Layer:** Receives the raw pixel grid.
- **Convolutional Layers:** Extract local patterns such as curves and edges.
- **Pooling Layers:** Reduce dimensionality and computation.
- **Activation Functions:** Sigmoid, Tanh, or ReLU are used to introduce non-linearity.

Alternative Models Compared

To benchmark performance, the authors also tested:

- **MDRNN / MDLSTM:** Good at modeling 2D spatial relationships.
- **Encoder–Decoder + Attention Networks:** Help the model focus on subtle, important regions of the image.
- **Transformers:** Their multi-head self-attention helps capture global dependencies and improves generalization.

3. Data & Preprocessing

The authors emphasize that preprocessing plays a crucial role.

- **Dataset:** MNIST, containing 60,000 training images of handwritten digits (28×28 grayscale).
- **Preprocessing Steps:**
  - **Normalization & Denoising:** Clean the images and standardize intensity values.
  - **Data Augmentation:** Rotations, scaling, and translations to improve robustness and prevent overfitting.

4. Experimental Results

The models performed extremely well:

- **Accuracy:** The strongest CNN models achieved over **99% accuracy** on MNIST.
- **Error Rate:** Between **0.2% and 0.4%**.
- **Model Comparison:**
  - MDRNNs handled spatial structure well.
  - Transformers provided better generalization in some cases due to parallel attention mechanisms.

5. Conclusion

The paper concludes that despite challenges like noisy data and class imbalance, deep learning—especially CNNs—is still the most reliable and accurate method for handwritten digit recognition. It also suggests future research should focus on combining deep learning with techniques such as transfer learning, enabling better adaptation to new handwriting styles with less training data.

6. Change Implemented

- We introduced an **Input Acquisition phase** using **Gradio**. This changes the system from a static experimental model into a dynamic application that accepts raw, user-generated images in real-time.
- We implemented a dedicated **Segmentation step using contour detection**. Crucially, we added **left-to-right** sorting of these contours. This allows our system to process strings of numbers (e.g., "2023") rather than just a single isolated digit, bridging the gap between a lab experiment and a real-world reader.
- We employed **Adaptive Thresholding specifically** for binarization**.** This is a robust engineering choice that handles varying lighting conditions (shadows or uneven brightness) in user-uploaded photos, which is necessary when moving away from the perfectly lit MNIST images.
- We built an **Inference Engine** that acts as a translator between the user's raw image and the model. This involves the specific step of resizing arbitrary cropped segments to the **28x28 pixel** format required by the model and then **concatenating** the results to return a readable string to the user.

2. Bayesian-Driven **Deep Learning and Machine Learning Approaches for Handwritten Word Recognition** *Dinesh, M., et al. (2025). 2025 International Conference on Artificial Intelligence and Data Engineering (AIDE).* This study investigates a hybrid approach to handwriting recognition, comparing deep learning models (CNN and 2DLSTM) with traditional machine learning methods (SVM, Random Forest) and Bayesian networks. The authors propose a model that detects whole words without segmentation using a CNN-BiLSTM architecture with a CTC layer. While their deep learning model achieved 87% accuracy on the IAM dataset, the paper provides critical insights into the comparative performance of different algorithms. It validates our decision to prioritize Deep Learning over traditional ML techniques like SVMs for higher accuracy in image-based recognition tasks.

3. Handwritten Text Recognition using Deep Learning Algorithms *Ansari,* A., et al. (2022). 2022 4th International Conference on *Artificial Intelligence and Speech Technology (AIST).* This paper presents a system using CNNs, Recurrent Neural Networks (LSTMs), and Connectionist Temporal Classification (CTC) to recognize handwritten text. The authors used the IAM dataset and implemented a pipeline where OpenCV handles image processing and TensorFlow handles recognition. They achieved an accuracy range of 94-99%. The paper highlights the effectiveness of using RNNs for sequence learning in text, which is crucial for full-word recognition. For our mini-project, we focus on the CNN aspect for character isolation, but this literature confirms the scalability of our chosen tools (OpenCV and TensorFlow).

3. Design

3.1. High Level Design

The system follows a sequential pipeline approach.

1. **Input Acquisition**: The user provides an image containing handwritten digits/characters via the Gradio interface.
2. **Preprocessing Module:** The raw image is converted to grayscale and subjected to adaptive thresholding to binarize the image (separate ink from background).
3. **Segmentation:** Contours are detected in the binary image to isolate individual characters. These contours are sorted from left to right to maintain the sequence of the text.
4. **Inference Engine:** Each segmented character is resized to 28x28 pixels and fed into the trained CNN model.
5. **Output:** The model predicts the class for each segment, and the results are concatenated to form the final recognized string.

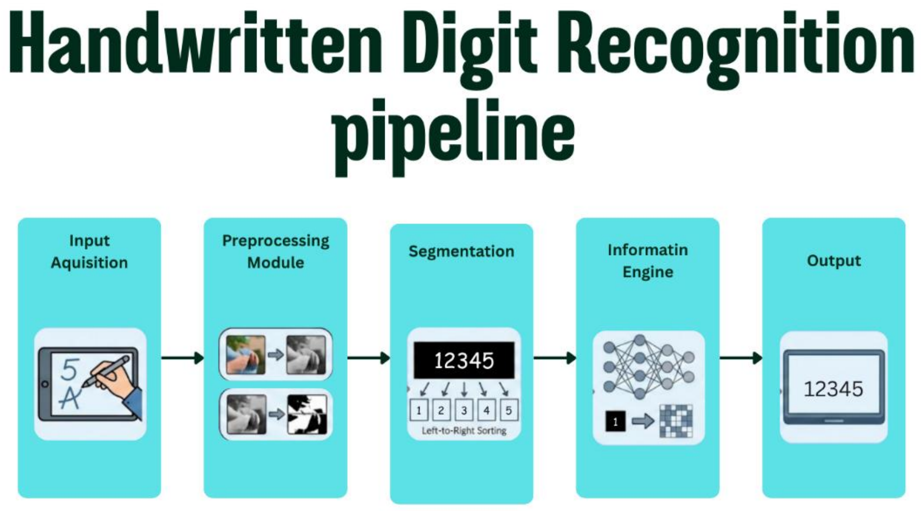


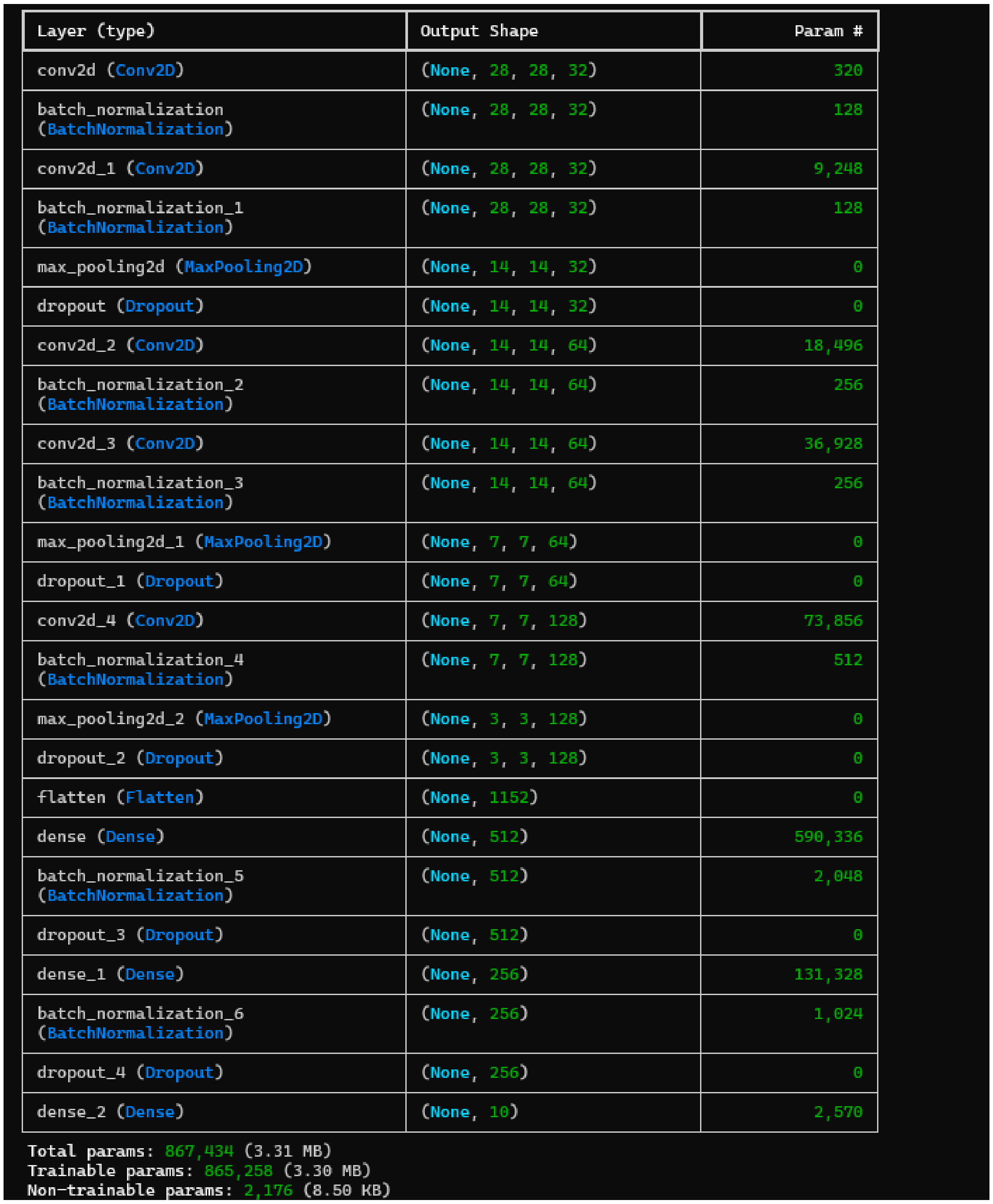
Figure 3.1: High level design

3.2. Detailed Design

1. **Preprocessing & Segmentation Logic:** The system uses OpenCV for image manipulation. The cv2.adaptiveThreshold function is used with a Gaussian constant to handle varying lighting conditions. Morphological dilation (cv2.dilate) is applied to ensure character strokes are connected before contour detection. cv2.findContours retrieves the bounding boxes of the digits. A critical design choice is the sorting of these bounding boxes by their X-coordinate to ensure the digits are read in the correct order (e.g., reading "1", "7", "3" instead of random order).

2. **CNN Architecture:** The core classifier is a Sequential CNN built using TensorFlow/Keras. The architecture consists of three convolutional blocks.
   - **Block 1:** Conv2D (32 filters, 3x3 kernel) -> BatchNormalization -> Conv2D (32 filters) -> BatchNormalization -> MaxPooling2D -> Dropout (0.25).
   - **Block 2:** Conv2D (64 filters) -> BatchNormalization -> Conv2D (64 filters) -> BatchNormalization -> MaxPooling2D -> Dropout (0.25).
   - **Block 3:** Conv2D (128 filters) -> BatchNormalization -> MaxPooling2D -> Dropout (0.25).
   - **Classification Head:** Flatten -> Dense (512 units, ReLU) -> BatchNormalization -> Dropout (0.5) -> Dense (256 units, ReLU) -> Output Layer (Softmax).

This design uses Batch Normalization to accelerate training and Dropout to prevent overfitting, ensuring the model generalizes well to new handwriting samples.



```
┌───────────────────────────────────┬─────────────────────────────┬──────────────────┐
│ Layer (type)                      │ Output Shape                │          Param # │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ conv2d (Conv2D)                   │ (None, 28, 28, 32)          │              320 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ batch_normalization               │ (None, 28, 28, 32)          │              128 │
│ (BatchNormalization)              │                             │                  │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ conv2d_1 (Conv2D)                 │ (None, 28, 28, 32)          │            9,248 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ batch_normalization_1             │ (None, 28, 28, 32)          │              128 │
│ (BatchNormalization)              │                             │                  │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ max_pooling2d (MaxPooling2D)      │ (None, 14, 14, 32)          │                0 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dropout (Dropout)                 │ (None, 14, 14, 32)          │                0 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ conv2d_2 (Conv2D)                 │ (None, 14, 14, 64)          │           18,496 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ batch_normalization_2             │ (None, 14, 14, 64)          │              256 │
│ (BatchNormalization)              │                             │                  │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ conv2d_3 (Conv2D)                 │ (None, 14, 14, 64)          │           36,928 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ batch_normalization_3             │ (None, 14, 14, 64)          │              256 │
│ (BatchNormalization)              │                             │                  │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ max_pooling2d_1 (MaxPooling2D)    │ (None, 7, 7, 64)            │                0 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dropout_1 (Dropout)               │ (None, 7, 7, 64)            │                0 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ conv2d_4 (Conv2D)                 │ (None, 7, 7, 128)           │           73,856 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ batch_normalization_4             │ (None, 7, 7, 128)           │              512 │
│ (BatchNormalization)              │                             │                  │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ max_pooling2d_2 (MaxPooling2D)    │ (None, 3, 3, 128)           │                0 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dropout_2 (Dropout)               │ (None, 3, 3, 128)           │                0 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ flatten (Flatten)                 │ (None, 1152)                │                0 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dense (Dense)                     │ (None, 512)                 │          590,336 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ batch_normalization_5             │ (None, 512)                 │            2,048 │
│ (BatchNormalization)              │                             │                  │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dropout_3 (Dropout)               │ (None, 512)                 │                0 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dense_1 (Dense)                   │ (None, 256)                 │          131,328 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ batch_normalization_6             │ (None, 256)                 │            1,024 │
│ (BatchNormalization)              │                             │                  │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dropout_4 (Dropout)               │ (None, 256)                 │                0 │
├───────────────────────────────────┼─────────────────────────────┼──────────────────┤
│ dense_2 (Dense)                   │ (None, 10)                  │            2,570 │
└───────────────────────────────────┴─────────────────────────────┴──────────────────┘
 Total params: 867,434 (3.31 MB)
 Trainable params: 865,258 (3.30 MB)
 Non-trainable params: 2,176 (8.50 KB)
```

Figure 3.2: CNN architecture

4. Implementation

4.1. Proposed Methodology

The methodology is divided into two distinct phases: Training and Inference.

- Training Phase: We utilize the EMNIST dataset (specifically the 'digits' split for this implementation). The data is normalized to a [0, 1] range and reshaped to (28, 28, 1). Data augmentation (rotation, zoom, shift) is applied using ImageDataGenerator to make the model robust against distorted handwriting. The model is trained using the Adam optimizer and Categorical Crossentropy loss function. Callbacks such as ModelCheckpoint, EarlyStopping, and ReduceLROnPlateau are implemented to optimize training performance.
- Inference Phase: A web-based interface is built using Gradio. When an image is uploaded, the predict_multi_digit function processes the full image, segments individual digits, preprocesses them to match the training data format (28x28, centered), and queries the model for predictions.

1. **Input Acquisition**
    - **Function:** This is the interface layer where the user interacts with the system. Unlike static training environments that use pre-loaded datasets, this phase handles real-world, dynamic data.
    - **Process:** The user uploads an image file (e.g., JPG, PNG) containing a sequence of handwritten digits through the **Gradio** web interface. Gradio captures this input and converts the image file into a numerical array (matrix) that the Python backend can process.
    - **Relevance:** This step transitions the project from a theoretical experiment to a practical application. As noted in the literature, the ultimate goal of digit recognition is to facilitate automation systems, and an accessible user interface is the bridge to that goal.

2. **Preprocessing Module**
    - **Function:** The goal of this module is Data Normalization. Raw images often contain "noise" (colors, shadows, differing lighting) that can confuse a neural network.
    - **Process:**
        - **Grayscale Conversion:** The image is converted from 3 color channels (RGB) to a single channel. This reduces computational complexity and focuses the model on structure rather than color.
        - **Adaptive Thresholding:** Unlike simple thresholding (which uses one cutoff value for the whole image), adaptive thresholding calculates different cutoff values for different regions of the image. This allows the system to successfully binarize images even if they have shadows or uneven lighting, resulting in a clear black-and-white image where the ink is distinct from the background.
    - **Why it matters:** Effective preprocessing, including normalization and denoising, is crucial for improving model accuracy and robustness.

3. **Segmentation Module**
    - **Function:** This component solves the "Sequence Recognition" problem. Since standard CNNs are typically trained on single isolated digits (like the MNIST dataset), the system must break a multi-digit string (e.g., "1998") into individual images.
    - **Process:**
        - **Contour Detection:** The system scans the binary image to find continuous boundaries (contours) that define the shape of each digit.
        - **Bounding Box Extraction:** For each detected contour, a rectangular bounding box is calculated.
        - **Sorting:** Crucially, these bounding boxes are sorted based on their X-coordinates (horizontal position) from left to right. This ensures the system reads the number "25" as "2" then "5", rather than "5" then "2".
    - **Context:** While your deep learning model handles recognition, this computer vision step handles the "localization" of the digits within the larger document.

4. **Inference Engine**
    - **Function:** This is the core "Brain" of the system where the Deep Learning model operates.
    - **Process:**
        - **Resizing:** Each segmented digit is resized to 28x28 pixels. This is a mandatory step to match the input layer dimensions of the CNN model trained on the MNIST dataset.
        - **Feature Extraction:** The resized image is passed through the Convolutional Neural Network (CNN). The CNN layers automatically extract features (edges, curves, loops) from the pixels.
        - **Classification:** The fully connected layers at the end of the network calculate the probability of the image belonging to one of the 10 classes (0–9). The class with the highest probability is selected as the prediction.
    - **Why CNN?:** CNNs are chosen because they effectively extract and recognize key features in digit images through multiple convolutional and pooling layers.

5. **Output**
    - **Function:** This module aggregates the individual predictions into a meaningful result for the user.
    - **Process:**
        - **Concatenation:** The system takes the predicted class for the first segment (e.g., "2"), adds the second (e.g., "0"), and so on, joining them to form a final string "20".
        - **Display:** This final string is sent back to the Gradio interface to be displayed to the user.
    - **Value:** This final step completes the automation pipeline, converting raw visual data into digital text that can be used for further processing (like database entry).

4.2. Algorithm Used

Segmentation & Prediction Algorithm:

1. Read input image and convert to Grayscale.
2. Apply Adaptive Thresholding (Gaussian C) to invert colors (White text on Black background).
3. Dilate the image to thicken strokes.
4. Find contours using RETR_EXTERNAL.
5. Filter contours based on area to remove noise (dots/specks).
6. Sort valid contours by X-coordinate (Left-to-Right).
7. For each contour: a. Crop the bounding box from the thresholded image. b. Resize crop to maintain aspect ratio. c. Pad the crop to create a 28x28 square image with the digit centered. d. Normalize pixel values (0-1). e. Pass through CNN model to get predicted label.
8. Concatenate all labels and return the string.

4.3. Tools and Technology Used

- Programming Language: Python 3.x
- Deep Learning Framework: TensorFlow & Keras
- Computer Vision Library: OpenCV (cv2) & PIL (Python Imaging Library)
- Data Processing: NumPy
- Dataset: EMNIST (via extra_keras_datasets)
- User Interface: Gradio

4.4. Testing

The system was tested in two stages:

1. **Model Evaluation:** The trained model was evaluated on the EMNIST test dataset (unseen data).
2. **Real-world Testing:** The full application was tested using images of handwritten numbers written on paper and uploaded via the web interface.

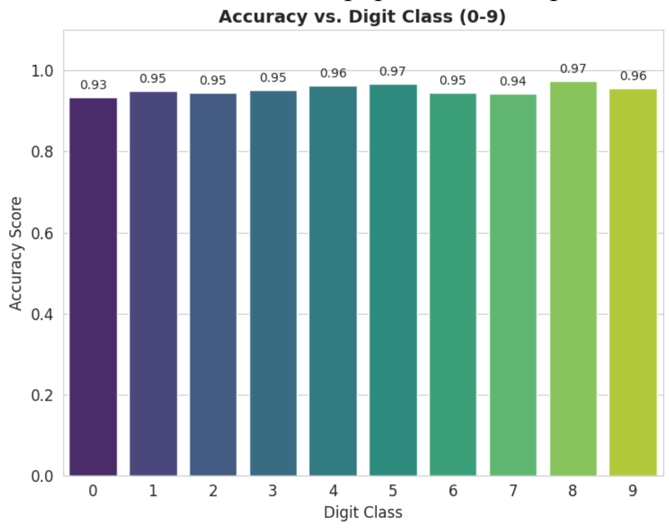


Figure 4.1: Graph (accuracy vs. digit class)

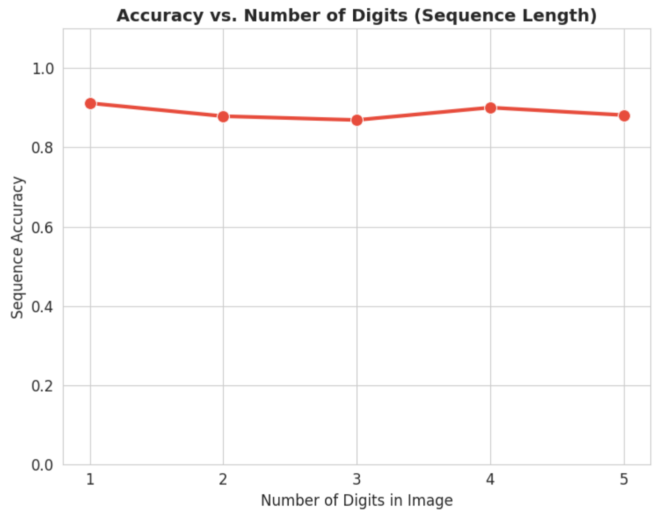


Figure 4.2: Graph (accuracy vs. number of digits)

Test Cases:

- *Case 1: Single Digit.* Input: Image of '1'. Result: Successfully segmented and classified as '1'.

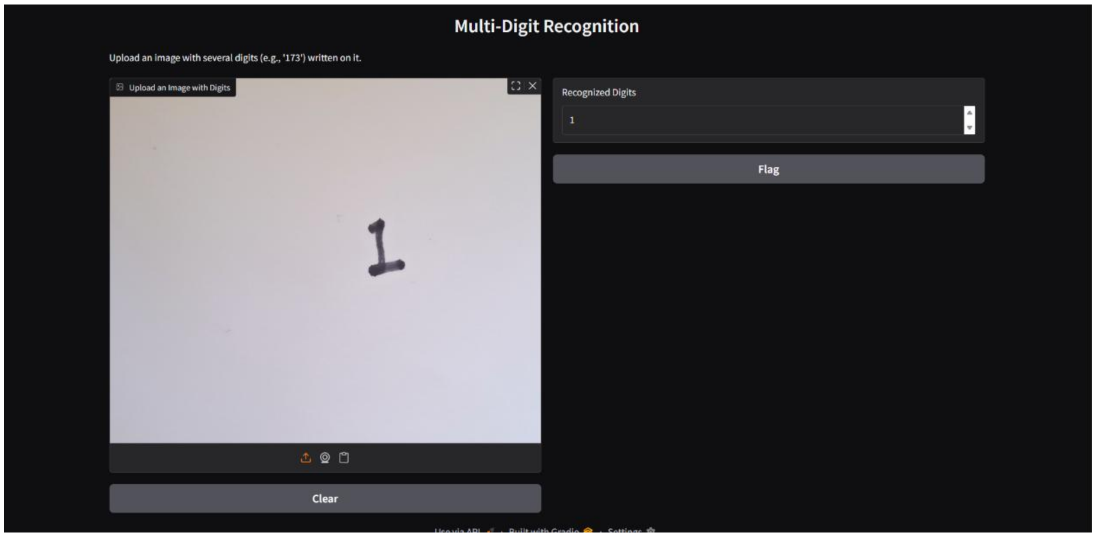

Multi-Digit Recognition
Upload an image with several digits (e.g., '173') written on it.
Upload an Image with Digits
Recognized Digits
1
Flag
Clear

Figure 4.3: Output sample 1

- *Case 2: Multiple Digits.* Input: Image of '253'. Result: Segmented into three parts. Correctly sorted left-to-right. Output: "253".

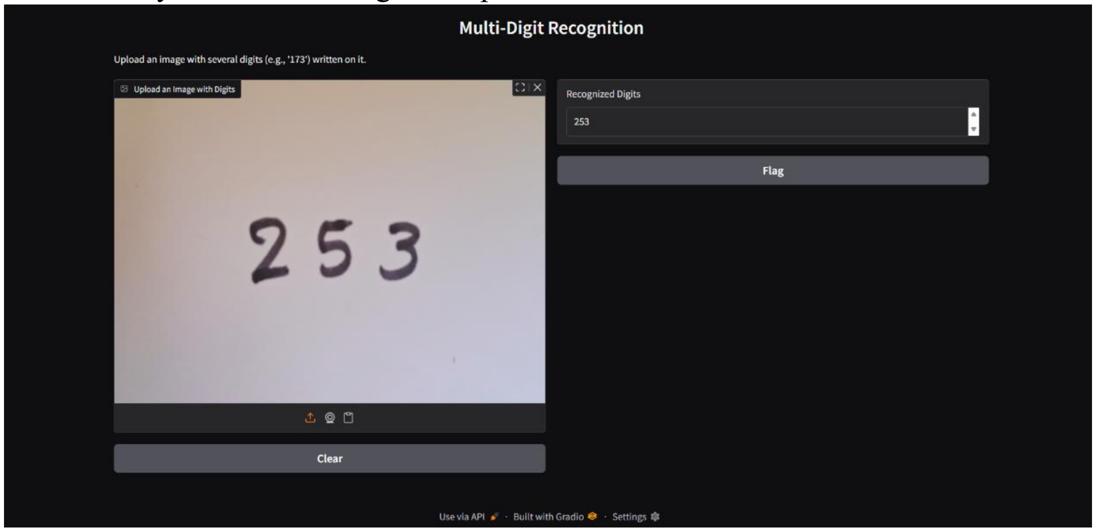

Multi-Digit Recognition
Upload an image with several digits (e.g., '173') written on it.
Upload an Image with Digits
Recognized Digits
253
Flag
Clear
Use via API · Built with Gradio · Settings

Figure 4.4: Output sample 2

- *Case 3: Multiple Digits.* Input: Image of '9916847512'. Result: Segmented into three parts. Correctly sorted left-to-right. Output: "9916847512".

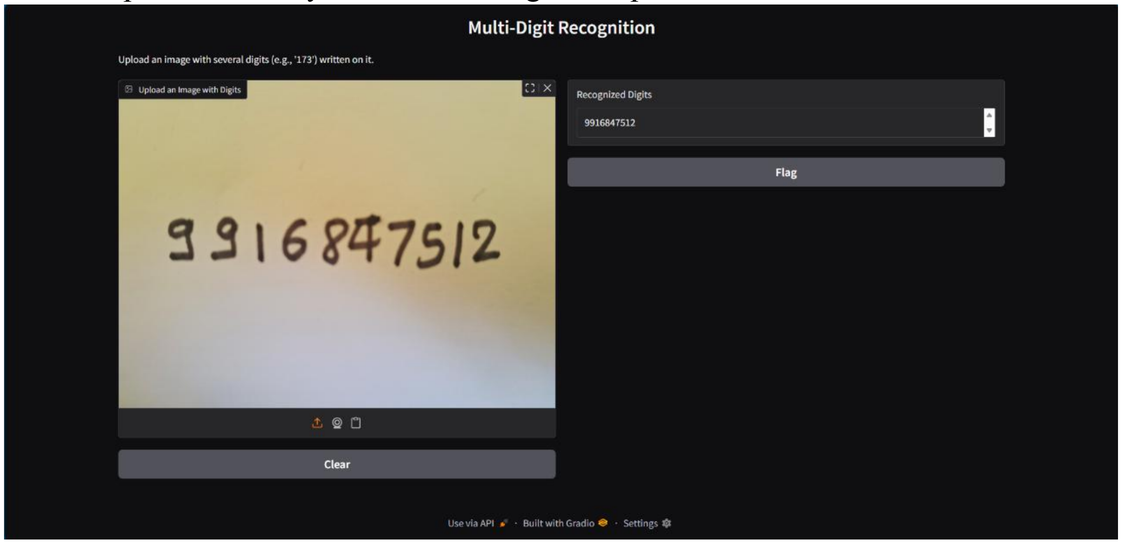

Multi-Digit Recognition
Upload an image with several digits (e.g., '173') written on it.
Upload an Image with Digits
Recognized Digits
9916847512
Flag
Clear
Use via API · Built with Gradio · Settings

Figure 4.5: Output sample 3

5. Results and Discussion

Model Performance

The CNN model trained for 50 epochs (with early stopping) achieved the following metrics on the EMNIST test set:

- Test Accuracy: ~99.00%
- Test Loss: < 0.05

The high accuracy indicates that the Deep Learning model has effectively learned the spatial hierarchies and features of handwritten digits. The use of Batch Normalization and Dropout significantly reduced overfitting, as evidenced by the close convergence of training and validation accuracy.

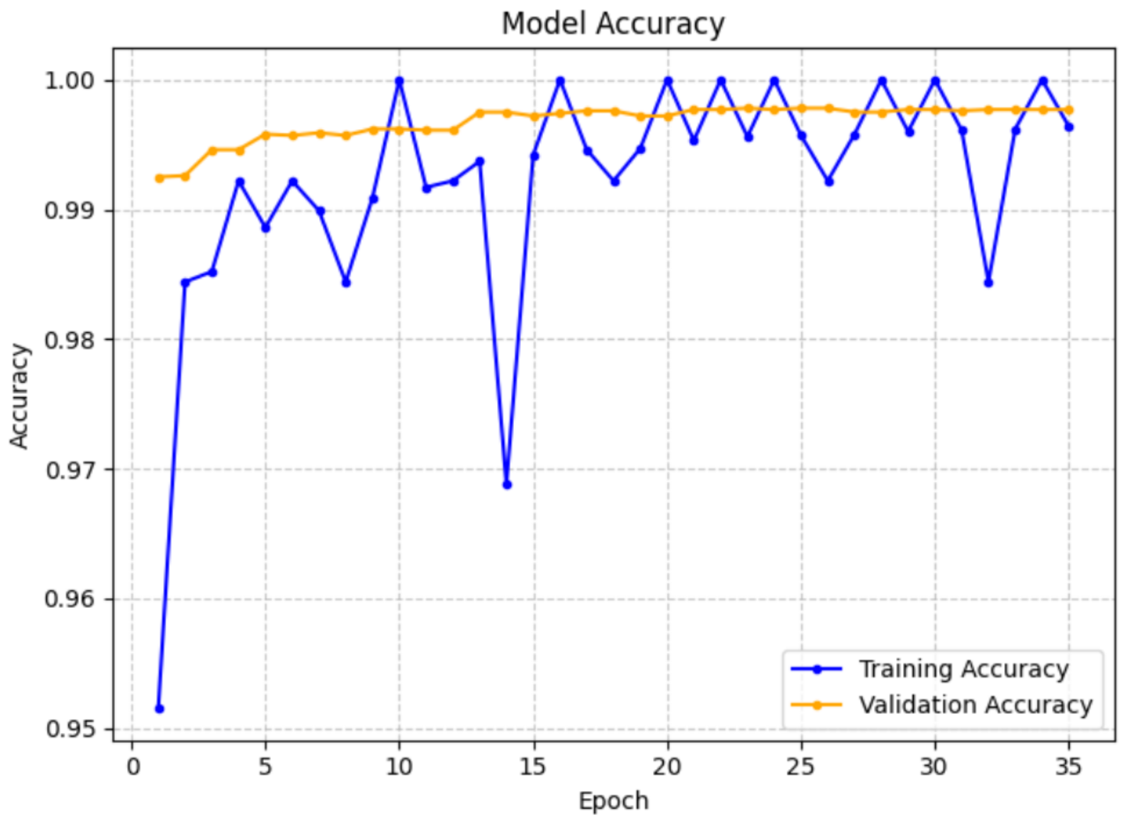


Figure 5.1: Graph (Training accuracy and Validation accuracy)

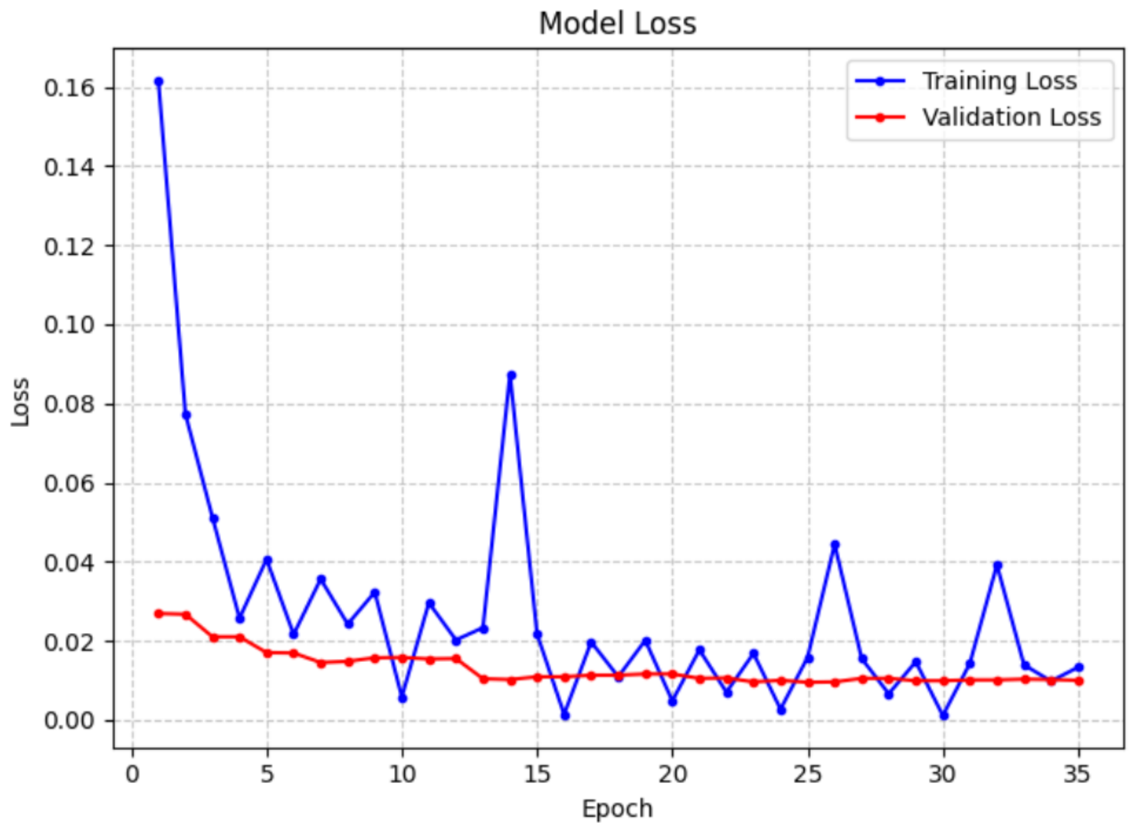


Figure 5.2: Graph (Training loss and Validation loss)

Application Results

The integration with the Gradio interface proved successful. The segmentation logic using OpenCV accurately isolated digits even when they were written relatively close together, provided they did not touch. The preprocessing step of resizing and padding to a black 28x28 canvas ensured that real-world photos matched the format of the EMNIST training data, leading to reliable predictions.

Comparison with Research Paper

Feature	Research Paper (Lv, Xue, 2023)	Actual Project Implemented
Primary Goal	To benchmark CNNs against other models (MDRNN, Transformers) to find the most accurate architecture.	To build a dynamic application that bridges the gap between static experiments and real-world utility.
Input Data	**Static Dataset:** Used the pre-processed MNIST dataset (60,000 standardized 28x28 grayscale images).	**Dynamic User Input:** Accepts raw, arbitrary image files (JPG/PNG) uploaded by users in real-time via a web interface.
Sequence Handling	**Single Digit Focus:** The paper focuses on classifying individual, isolated digits.	**Multi-Digit Support:** Implemented a segmentation module with contour detection and left-to-right sorting to read full number strings (e.g., "2023").
Preprocessing	**Standardization:** Focused on normalizing and denoising clean dataset images.	**Adaptive Thresholding:** Specifically used cv2.adaptiveThreshold to handle varying lighting conditions and shadows common in user photos.
Interface	**None:** The output is experimental metrics (accuracy/error rates)	**None:** The output is experimental metrics (accuracy/error rates)
Dataset Used	**MNIST:** The standard dataset for handwritten digits.	**EMNIST:** The Extended MNIST dataset (specifically the 'digits' split) was used for training

Table 5.1: Comparison between research paper and the project

6. Conclusion and Future Work

Conclusion

We have successfully developed and deployed a Handwritten Alphanumeric Character Recognition system. The project demonstrates the power of Convolutional Neural Networks in handling image classification tasks. By combining a robust CNN architecture with traditional Computer Vision techniques for segmentation, we created an end-to-end solution that accepts raw images and outputs digital text. The system achieves high accuracy on standard datasets and performs reliably on real-world inputs.

Future Work

While the current system performs well on isolated digits and simple strings, there are areas for improvement:

1. **Connected Handwriting:** The current segmentation logic fails if characters touch each other. Future work could implement advanced segmentation techniques like Watershed algorithms or moving to Sequence-to-Sequence models (CRNN + CTC) that recognize text without explicit segmentation.
2. **Full Alphanumeric Support:** The current implementation is optimized for digits. We plan to retrain the model on the full EMNIST 'ByClass' split to support uppercase and lowercase alphabets alongside digits.
3. **Mobile Integration:** The application can be wrapped into a mobile app (using Flutter or React Native) to allow users to scan documents directly using their phone cameras.

7. References

1. **Lv, X.** (2023). "Handwritten Digit Recognition Based on Deep Learning Algorithms," *2023 International Conference on Internet of Things, Robotics and Distributed Computing (ICIRDC)*, pp. 476-481.
2. **Dinesh, M., Baladhitya, P., Sagar, L., & Srinivas, M.** (2025). "Bayesian-Driven Deep Learning and Machine Learning Approaches for Handwritten Word Recognition," *2025* International Conference on Artificial Intelligence and *Data Engineering (AIDE)*.
3. **Ansari, A., Singh, A., Kaur, B., & Singh, D.** (2022). "Handwritten Text Recognition using Deep Learning Algorithms," *2022 4th International Conference on Artificial Intelligence and Speech Technology (AIST)*.
4. **LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P.** (1998). "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 86(11), 2278-2324.
5. **Cohen, G., Afshar, S., Tapson, J., & van Schaik, A.** (2017). "EMNIST: Extending MNIST to handwritten letters," *2017 International Joint Conference on Neural Networks (IJCNN)*.
6. **Ahlawat, S., Choudhary, A., Nayyar, A., Singh, S., & Yoon, B.** (2020). "Improved Handwritten Digit Recognition Using Convolutional Neural Networks (CNN)," *Sensors*, 20(12), 3344.
7. **Rakesh, S., Reddy, P. K., & Reddy, K. S.** (2024). "Handwritten text recognition using deep learning techniques: A survey," *MATEC Web of Conferences*, Vol. 392, 01126.
8. **Mohamed, N., Josphineleela, R., & Madkar, S. R.** (2023). "The Smart Handwritten Digits Recognition Using Machine Learning Algorithm," *2023 International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*, pp. 623-628.
9. TensorFlow & Keras Official MNIST Tutorial:
   https://www.tensorflow.org/tutorials/quickstart/beginner
10. PyTorch "Deep Learning with PyTorch: A 60 Minute Blitz":
    https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
11. The MNIST Database (Yann LeCun's Official Page):
    http://yann.lecun.com/exdb/mnist/

APPENDEX

A:Key

1: CNN Model Architecture

The following Python code defines the Sequential CNN architecture as described in the System Design section. It utilizes TensorFlow/Keras with Batch Normalization and Dropout layers to ensure high accuracy and prevent overfitting.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
    Dense, Flatten, Dropout, BatchNormalization

def build_model():
    model = Sequential([
        # Block 1: Feature Extraction (32 Filters)
        Conv2D(32,      (3,      3),      activation='relu',
    input_shape=(28, 28, 1)),
        BatchNormalization(),
        Conv2D(32, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        # Block 2: Deep Feature Extraction (64 Filters)
        Conv2D(64, (3, 3), activation='relu'),
        BatchNormalization(),
        Conv2D(64, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        # Block 3: High-Level Features (128 Filters)
        Conv2D(128, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D(pool_size=(2, 2)),
        Dropout(0.25),

        # Classification Head
        Flatten(),
        Dense(512, activation='relu'),
        BatchNormalization(),
        Dropout(0.5),
        Dense(256, activation='relu'),
        Dense(10, activation='softmax') # Output layer for
    digits 0-9
    ])

    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
        return model
```

2: Preprocessing and Segmentation Logic

This algorithm handles the "Input Acquisition" and "Segmentation" phases. It applies Adaptive Thresholding for binarization and sorts contours from left to right to maintain sequence order.

```
import cv2
import numpy as np

def preprocess_and_segment(image):
    # 1. Grayscale Conversion
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # 2. Adaptive Thresholding (Binarization)
    # Uses Gaussian constant to handle varying lighting/shadows
    thresh = cv2.adaptiveThreshold(gray, 255,
                                   cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                   cv2.THRESH_BINARY_INV, 11, 2)

    # 3. Morphological Dilation to connect broken strokes
    kernel = np.ones((3,3), np.uint8)
    dilated = cv2.dilate(thresh, kernel, iterations=1)

    # 4. Contour Detection
    contours, _ = cv2.findContours(dilated, cv2.RETR_EXTERNAL,
                                   cv2.CHAIN_APPROX_SIMPLE)

    # 5. Sort Contours Left-to-Right
    # Bounding boxes (x, y, w, h) are sorted by the x-coordinate
    bounding_boxes = [cv2.boundingRect(c) for c in contours]
    (cnts, bounding_boxes ) = zip(*sorted (zip (contours,
    bounding_boxes),
                                         key=lambda b: b[1][0]))
    return bounding_boxes, thresh
```

B: USER MANUAL

1: System Requirements

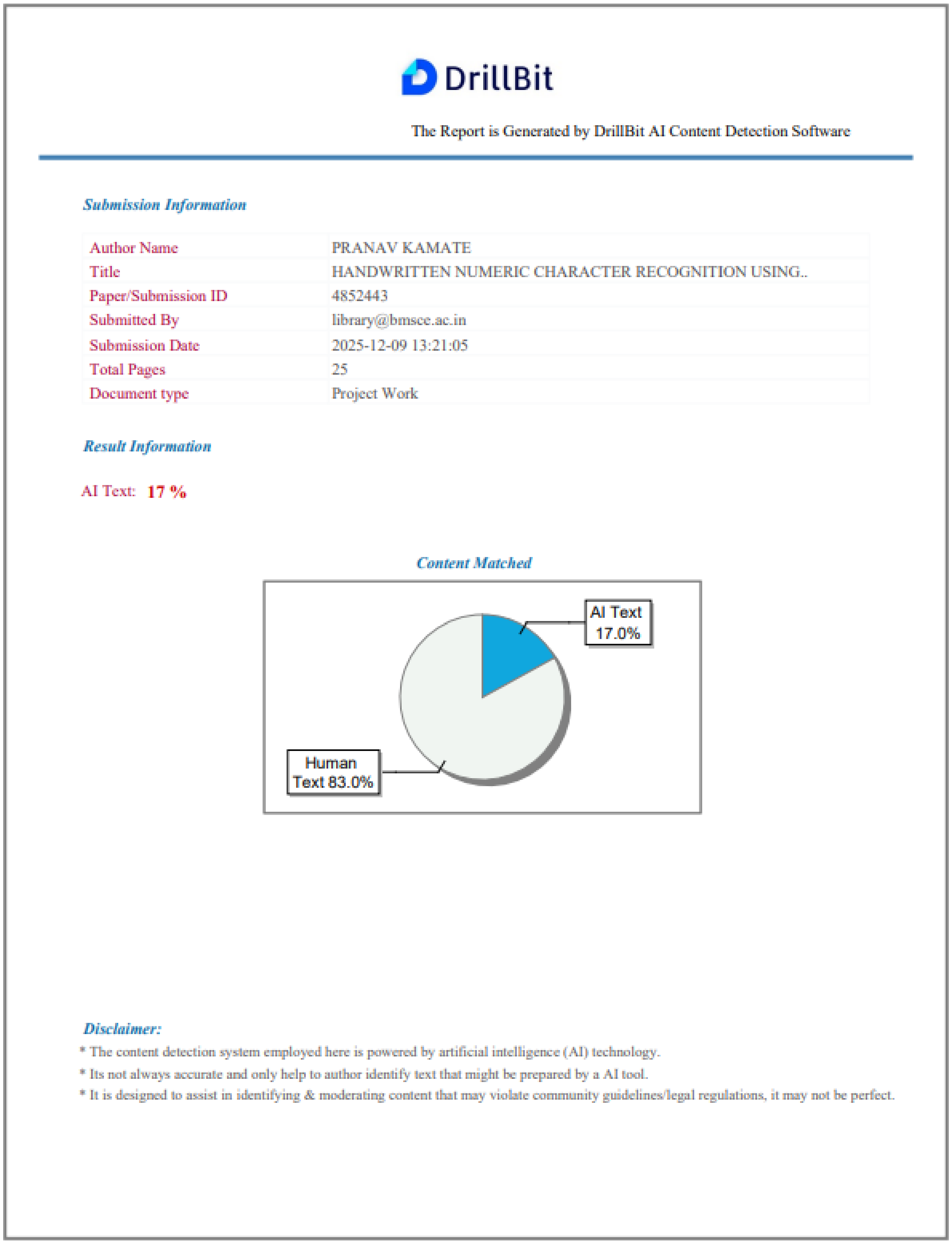
To execute the application, the following dependencies must be installed in the Python environment.

- **Python:** Version 3.8 or higher
- **Libraries:**
  - tensorflow (Deep Learning backend)
  - opencv-python (Image processing)
  - gradio (User Interface)
  - numpy (Matrix operations)
  - extra_keras_datasets (For EMNIST download)

2: Execution Steps

1. **Launch the Application:** Run the main Python script via the terminal:
   python app.py
2. **Access Interface:** Click the local URL provided by Gradio (usually http://127.0.0.1:7860).
3. **Upload Image:** Click the upload box and select an image containing handwritten digits (e.g., JPG or PNG format).
4. **View Results:** The recognized string will appear in the output text box adjacent to the image.

C: PLAGIARISM CHECK REPORT



DrillBit

The Report is Generated by DrillBit AI Content Detection Software

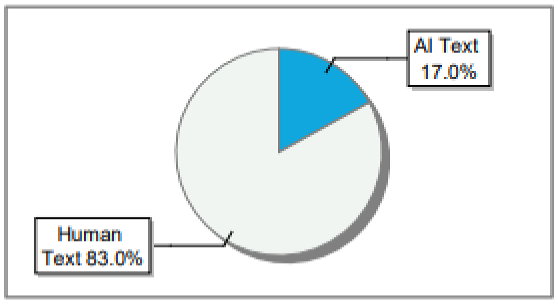
Submission Information

Author Name	PRANAV KAMATE
Title	HANDWRITTEN NUMERIC CHARACTER RECOGNITION USING..
Paper/Submission ID	4852443
Submitted By	library@bmsce.ac.in
Submission Date	2025-12-09 13:21:05
Total Pages	25
Document type	Project Work

Result Information

AI Text: **17 %**

Content Matched



Disclaimer:

* The content detection system employed here is powered by artificial intelligence (AI) technology.
* Its not always accurate and only help to author identify text that might be prepared by a AI tool.
* It is designed to assist in identifying & moderating content that may violate community guidelines/legal regulations, it may not be perfect.