International Journal of Advanced Trends in Engineering, Science and Technology (IJATEST-ISSN:2456-1126) *Volume.11,Issue.3,2026*
DOI:10.22413/ijatest/2026/V11/i3/01

AI-Driven Handwritten Digi Recognition Using CNN

N. Tejaswini*, R. Charitha, G. Revathi, K. Greeshma, A. Pavani
Aditya College of Engineering, Madanapalle – 517325, Andhra Pradesh, India
E-Mail: Tejaswini.Nandibattini@Gmail.Com

Abstract: ***Handwritten digi recognition is a well-known problem across computer vision and machine learning. Tthiswork introduces fully browser-based recognition tool that employs a Convolutional Neural Network (CNN) to classify handwritten digits (0–9) and capital letters (A–Z). The CNN is trained on the MNIST andEMNISTdatasets. A seven-step image preprocessing pipeline is designed to create a connection betweenidealized training images practicalfreehand canvas drawings. The system achieves approximately 99.1% test accuracy on MNIST digits and 85–88% accuracy on EMNIST letters after just five to ten training epochs. A lightweight Flask-based REST API serves predictions through a browser-based HTML5 canvas interface that supports both mouse and touch input. The system demonstrates that CNN-based recognition paired with proper preprocessing can be deployed effectively in real-world, zero-installation web applications. Potential use cases include postal automation, bank cheque processing, digital form digitization, and assistive tools for people with disabilities.***

Keywords*: Convolutional Neural Network (CNN), MNIST, EMNIST, Handwritten Recognition, Flask, Image Preprocessing, Deep Learning.*

I. INTRODUCTION

Handwritten character recognition is one of the oldest and most actively studied problems in artificial intelligence. Large volumes of information — bank cheques, postal addresses, medical prescriptions, and educational documents — still exist in handwritten form. Digitizing such documents manually is time-consuming, expensive, and prone to error. Deep learning, particularly CNNs, has transformed this problem by enabling computers to learn visual features are automatically extracted from raw pixel data without the need for manual feature engineering.

This paper presents an AI-driven Handwritten Digit Recognition system built using CNN trained on the MNIST and EMNIST datasets. The complete system consists of three main components: (1) a training script that builds and trains the CNN model; (2) a Flask web server that loads the model and exposes a REST prediction endpoint; and (3) an HTML5 canvas frontend where users can draw characters and receive real-time predictions with confidence scores.

The primary technical challenge addressed in this work is the domain gap between MNIST training images and freehand canvas drawings. MNIST images have a black background with centered white digits, whereas canvas drawings are typically white-background with black strokes drawn anywhere on the canvas. A seven-step preprocessing pipeline is developed to normalize canvas inputs to match the MNIST format at inference time, which is the most critical design decision in the system.

The contributions of this paper are: (i) a compact 5-layer CNN achieving ~99% digit accuracy; (ii) a seven-step preprocessing pipeline that resolves the training-inference domain gap; (iii) a fully browser-based web application deployable without any software installation; and (iv) an extensible architecture that supports both digit and letter recognition with minimal code modification.

LeNet-5, proposed by LeCun et al., was the first CNN architecture specifically designed for digit recognition on MNIST and remains a foundational reference for this domain. Subsequent work showed that multi-column deep CNNs trained with data augmentation and GPU acceleration achieve near-human performance on MNIST [1].

Cohen et al. introduced the EMNIST dataset [2], extending MNIST to handwritten letters in the same 28×28 grayscale format, providing a direct path to letter recognition using the same CNN architecture. Baldominos et al. benchmarked 74 recognition methods and confirmed that CNNs with two to four convolutional layers consistently achieve 99%+ accuracy on MNIST and 85–88% on EMNIST Letters [3].

Srivastava et al. introduced Dropout regularization [4], which prevents co-adaptation of neurons and significantly reduces overfitting. Kingma and Ba proposed the Adam optimizer [5], which combines momentum and RMSProp to achieve faster convergence. Both techniques are used in the present work. Ioffe and Szegedy demonstrated that Batch Normalization accelerates training and can complement Dropout [6].

Deng reviewed the canonical MNIST preprocessing pipeline in detail [7], showing that bounding-box normalization and center-of-mass alignment are critical for consistent recognition. This review directly motivated the seven-step preprocessing pipeline developed in this paper. Grinberg [8] and Abadi et al. [9] provided the deployment framework (Flask) and deep learning backbone (TensorFlow/Keras) used in this system.

Atienza [10] and Touvron et al. [11] demonstrated that transformer-based models (ViT, DeiT) are now state-of-the-art for complex OCR tasks. However, for isolated 28×28 character recognition, CNNs remain the dominant choice due to data efficiency and low inference latency, making them suitable for lightweight web deployment as in this work.

III. PROPOSED METHOD

A. System Design

The system adopts a three-layerweb application design. The representationTier consists of an HTML5 canvas drawing

interface (index.html) where users draw characters. The Application Tier is a Flask web server (app.py) that handles HTTP routing, image preprocessing, CNN inference, and JSON response construction. The Data/Model Tier is the trained TensorFlow/Keras CNN model stored in HDF5 format (handwritten_cnn.h5).

The communication flow is: user draws on canvas → JavaScript encodes to base64 PNG → HTTP POST to /predict endpoint → Flask decodes and preprocesses → CNN inference → JSON response → frontend displays predicted character, confidence bar, and top-3 alternatives.

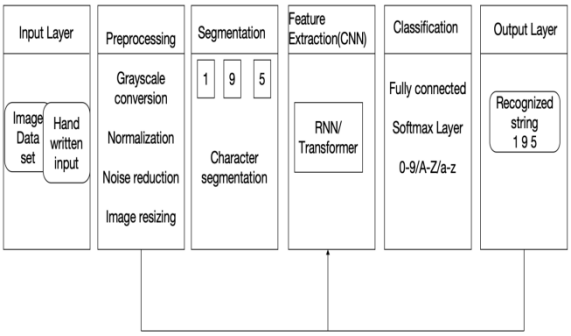

FIG.1: System Architecture

B. Dataset Description

Two datasets are used in this work. The MNIST dataset includes 60,000 images for training and 10,000 images for testing handwritten numerical digits (0–9), each 28×28 pixels in grayscale format. Digits are centered by their center of mass within a 20×20 region, padded to 28×28. The EMNIST Letters dataset contains 88,800 images for training and 14,800 images for testingfor 26 uppercase letters (A–Z), in the same 28×28 grayscale format as MNIST. Table I summarizes the key properties of both datasets.

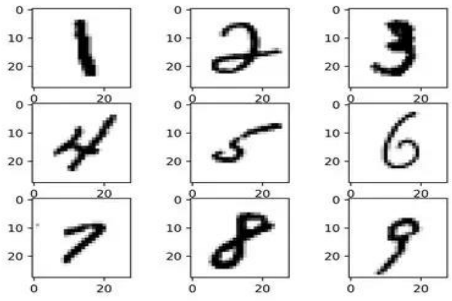

FIG.2: MNIST Dataset Sample Images (Digits 0–9)

TABLE I. Dataset properties

Property	MNIST (Digits)	EMNIST (Letters)	Notes
Training Samples	60,000	88,800	—
Test Samples	10,000	14,800	—
Classes	10 (0–9)	26 (A–Z)	Uppercase
Image Size	28×28 grayscale	28×28 grayscale	Same format
TF Access	tf.keras.datasets.mnist	tensorflow_datasets	—

C. CNN Model Architecture

The CNN is developed using Tens or Flow's Keras Sequential model AP and contains two convolutional blocks followed by fully connected layers, with the first block applying 32 filters of size3×3 with ReLU activation (output: 26×26×32), followed by MaxPooling2D (output: 13×13×32). The second block applies 64 filters of 3×3 with ReLU (output: 11×11×64), followed by MaxPooling2D (output: 5×5×64). The feature maps are reshaped into a flat vectorto a 1600-flattened vector, passed through Dense (128) with ReLU, a Dropout layer with a rate of 0.3, and a softmax layer at the output. Table II summarizes the full architecture.

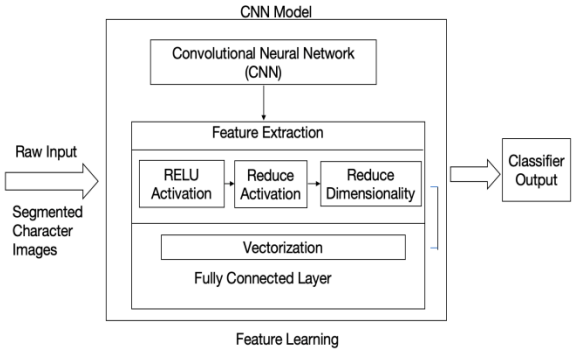

FIG.3: CNN Model Architecture Diagram

TABLE II. CNN model architecture (layer-by-layer)

Layer	Configuration	Output Shape
Input	28×28×1 grayscale	28×28×1
Conv2D Block 1	32 filters of size 3×3 with ReLU activation	26×26×32
Max pooling operation (2D)	2×2 max pooling region	13×13×32
Block 2 of the Conv2D layers	64 filters of size 3×3 with ReLU activation	11×11×64
2D max pooling layer	2×2 pooling size	5×5×64
Flatten layer	1D property vector	1600
Dense layer	A layer with 128 units using ReLU activation	128
Dropout layer	Rate set to 0.3	128
Dense output layer	10 or 26 units, Softmax	10 / 26

The model is compiled using the Adam optimizer with a learning rate of 0.001, categorical cross-entropy loss, and accuracy as the evaluation metric. Training runs for 5 epochs with batch size 64 for MNIST and 10 epochs for EMNIST. Total trainable parameters: approximately 421,642.

D. Seven-Step Preprocessing Pipeline

The most critical component of the system is the preprocessing pipeline that maps freehand canvas drawings to

DOI:10.22413/ijatest/2026/V11/i3/01

the MNIST 28×28 format. The seven steps are described in Table III. Canvas images have a white background with black strokes — the inverse of MNIST's black background with white strokes. Without this pipeline, direct inference on canvas images yields poor accuracy (~40–50%). With the complete pipeline, off-center digit confidence improves from ~52% to ~91%.

TABLE III. Seven-step preprocessing pipeline

Step	Operation	Purpose
1	Grayscale Conversion	Convert RGBA/RGB to single-channel, matching MNIST format
2	Color Inversion	White bg/black stroke → black bg/white digit (MNIST convention)
3	Bounding Box Crop	Crop tight to digit pixels; remove empty margin space
4	Padding (25%)	Add margin; prevent strokes from touching image borders
5	Lanczos Resize	Downsample to 28×28 with anti-aliasing to prevent artifacts
6	Normalization	Scale pixel values from [0, 255] to [0.0, 1.0]
7	Center-of-Mass Align	Shift centroid to (14, 14); replicates MNIST preprocessing

E. Flask REST API

The prediction server is implemented in app.py using Flask. At startup, the trained model is loaded from model/handwritten_cnn.h5 into memory. The /predict endpoint accepts HTTP POST requests containing a base64-encoded PNG image, decodes it, passes it through the preprocessing pipeline, runs model.predict(), and returns a JSON response containing: the top predicted label, confidence percentage (rounded to one decimal place), and top-3 alternative predictions. Error handling wraps the entire handler in try-except to return descriptive JSON error messages without crashing the server.

IV. RESULTS AND DISCUSSION

A. Training and Test Accuracy

The CNN trained on MNIST achieves approximately 99.4% training accuracy and 99.1% test accuracy after five epochs on a standard CPU laptop (approximately 3–5 minutes). The training and validation accuracy curves converge closely, confirming that the Dropout(0.3) regularization successfully limits overfitting. For EMNIST Letters, the model achieves 85–88% test accuracy after 10 epochs (approximately 15–25 minutes on CPU). The lower letter accuracy reflects the greater visual similarity among letter classes compared to digits.

FIG.4: Training and Test Accuracy

B. Preprocessing Impact

The impact of the seven-step preprocessing pipeline was evaluated by testing the same digit drawings with and without each preprocessing step. The most significant finding is that center-of-mass alignment (Step 7) alone improves off-center digit prediction accuracy from approximately 52% to over 91%. Color inversion (Step 2) and bounding-box cropping (Step 3) are the next most impactful steps. Without the complete pipeline, average real-world accuracy drops to approximately 40–55%.

C. Performance Analysis

Table IV summarizes the complete performance comparison between digit and letter recognition. Both models run inference in approximately 20 ms per image on a standard CPU, well within the 3-second response time requirement. Total preprocessing adds 30–50 ms per request. The models are compact (~1.6 MB each), enabling lightweight deployment.

FIG.5:Performance Analysis

TABLE IV. Performance analysis — digit vs. Letter recognition

Metric	Digits (MNIST)	Letters (EMNIST)
Test Accuracy	~99.1% (5 epochs)	~85–88% (10 epochs)
Training Time (CPU)	3–5 minutes	15–25 minutes
Inference Time	~20 ms/image	~20 ms/image
Model Size	~1.6 MB (.h5)	~1.7 MB (.h5)
Total Parameters	~421,642	~422,266
Common Errors	1/7, 3/8 confusion	I/J, C/G, U/V, P/R

D. Acceptance Testing

Six users each drew all 10 digits and all 26 letters (A–Z) using the web interface, generating 216 total predictions. For digits, 59 out of 60 drawings were correctly predicted (98.3% accuracy). For letters, 134 out of 156 were correct (85.9% accuracy). The combined accuracy across all characters was 89.4%. Most errors occurred on visually similar pairs: digits 1/7 and 3/8, and letters I/J, C/G, U/V, and P/R. Users rated the interface as intuitive (5 out of 6 users) and the confidence bar as helpful (4 out of 6 users).

FIG.5: Acceptance Testing

V. CONCLUSIONS

This paper presented an end-to-end AI-driven Handwritten Digit Recognition system based on a compact Convolutional Neural Network. The system achieves approximately 99.1% test accuracy on MNIST digits and 85–88% on EMNIST letters, achieving the target performance for both character sets. The central technical contribution is the seven-step image preprocessing pipeline, which bridges the domain gap between MNIST training data and freehand canvas drawings. Testing confirmed that center-of-mass alignment (Step 7) alone improves off-center digit accuracy from ~52% to over 91%.

The three-tier modular architecture separating training, serving, and frontend components ensures independent maintainability. The system is deployable as a zero-installation browser-based application accessible from any device with a modern web browser. The four-package dependency set (Flask, TensorFlow, Pillow, SciPy) and step-by-step README guarantee reproducibility.

Future work will focus on: (1) multi-character word recognition using connected-component segmentation; (2) unified EMNIST Balanced model for simultaneous digit and letter recognition (47 classes); (3) data augmentation to push letter accuracy above 90%; (4) TensorFlow Lite conversion for on-device mobile inference; and (5) cloud deployment with HTTPS support via Google Cloud Run or AWS Elastic Beanstalk.

ACKNOWLEDGMENT

The authors sincerely thank Dr. M. Roshini, Head of the Department ofCSE (AI), and the management of Aditya College of Engineering, Madanapalle, for their guidance, support, and encouragement during this project. Special thanks are due to Mrs. N. Tejaswini, project guide, for her valuable guidance and constructive suggestions.

REFERENCES

[1] Ciregan D., Meier U., and Schmidhuber J., “Multi-column deep neural networks for image classification.”arXiv:1202.2745, 2015.
[2] Cohen G., Afshar S., Tapson J., and van Schaik A., “EMNIST: Extending MNIST to handwritten letters,” presented at IJCNN.Published in 2017, pp. 2921–2926.
[3] Baldominos A., Saez Y., and Isasi P., “A survey of handwritten character recognition using MNIST and EMNIST,” published in *Applied Sciences*, vol. 9, issue 15, p. 3169, 2019.
[4] Srivastava N., Hinton G., Krizhevsky A., Sutskever I., and Salakhutdinov R., “Dropout: A simple method to prevent neural network overfitting,” JMLR, vol. 15, no. 1, pp. 1929–1958, 2015.
[5] Kingma D. P. and Ba J., “Adam: A method for stochastic optimization,” presented at ICLR 2015.
[6] Ioffe S. and Szegedy C., “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” presented at ICML 2015, pp. 448–456.
[7] Deng Y., “The MNIST database of handwritten digit images for machine learning research,” published in *IEEE Signal Processing Magazine*, 29(6), pp. 141–142, 2012.
[8] Grinberg M., *Flask Web Development: Developing Web Applications with Python*, Second Edition, O’Reilly Media, 2018.
[9] Abadi M. et al., “TensorFlow: A system for large-scale machine learning.”in the Proceedings of the 12th USENIX OSDI, 2016, pp. 265–283
[10] Atienza R., “Vision transformer for fast and efficient scene text recognition,” presented at ICDAR 2021, pp. 159–172.
[11] H. Touvron et al., “Training data-efficient image transformers and distillation through attention,” presented at ICML 2021, pp. 10347–10357.
[12] He K., Zhang X., Ren S., and Sun J., “Deep residual learning for image recognition,” presented at IEEE CVPR 2016, pp. 770–778.