iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** IX **Month of publication:** September 2025

DOI: https://doi.org/10.22214/ijraset.2025.74159

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue IX Sep 2025- Available at www.ijraset.com

Handwritten Digit Recognition Using Multilayer Neural Networks with Keras and Tensorflow

D G Sindhu[1], Dr. G. C. Manjunatha[2]
[1]*Master of Digital Electronics and Communication, Proudhadevaraya Institute of Technology, Hosapete Karnataka*
[2]*Guide, Department of Electronics and Communication, Proudhadevaraya Institute of Technology, Hosapete, Karnataka*

***Abstract*: Handwritten digit recognition is an important problem in the areas of pattern recognition and computer vision, with several practical applications such as automated postal mail sorting, bank check processing, and digitization of handwritten documents. Accurately recognizing digits written by different individuals remains a challenge due to variations in writing styles, shapes, and orientations. Addressing this challenge requires advanced techniques capable of learning and generalizing patterns from image data.**

This project aims to develop a highly accurate handwritten digit recognition system using deep learning, specifically Convolutional Neural Networks (CNNs). The system is implemented using Python with the help of Keras and TensorFlow libraries. It is trained and evaluated on the MNIST dataset, which consists of 60,000 labeled training images and 10,000 test images of handwritten digits ranging from 0 to 9. The CNN architecture is designed to effectively capture spatial hierarchies in image data, using layers such as convolutional, pooling, and fully connected layers for feature extraction and classification.

Experimental results demonstrate that the proposed model achieves high accuracy in recognizing handwritten digits, confirming the strength of CNNs in handling image classification tasks. The success of this system highlights the capability of deep learning models to perform reliably in real-world scenarios. Overall, this project showcases the potential of artificial intelligence and deep learning in automating tasks that require image interpretation and has broad implications for use in banking, logistics, education, and other industries that rely on digit recognition systems.

I. INTRODUCTION

The automatic recognition of handwritten digits is a fundamental task. It involves teaching machines to identify and interpret digits written by humans, despite variations in style, thickness, orientation, and writing tools. As simple as it may seem, this problem presents many challenges due to the high variability in human handwriting. Task is complex when real-time performance and high accuracy are required in practical applications. Hence it will be significantly enhance the automation of several processes in the real world by cracking the issue, making it both an academically and industrially valuable area of research.

As a result, the demand for efficient and intelligent digit recognition systems has increased rapidly in recent years.a result, the demand for efficient and intelligent digit recognition systems has increased rapidly in recent years.

The pattern recognition to learn spatial hierarchies of features. Project, a CNN picture is coached with the MNIST dataset—a benchmark dataset in the machine learning community consisting of 70,000 labelled images of handwritten digits. Each Picture is resized to a 28x28 pixels. 60,000 snaps are included for training, and 10,000 picture used for examining the performance of the model.The wide variety of handwriting styles captured in this dataset makes it ideal for training a robust and generalizable recognition system.

The code written in Python, Keras and TensorFlow tools are used, two widely used open-source frameworks for deep learning. Keras offers a simple and intuitive interface that makes it easier to design and test neural network models, while TensorFlow takes care of the complex computations in the background, ensuring speed and efficiency.

In a CNN, each layer is designed to identify important features of an image, such as edges, corners, curves, and shapes that distinguish one digit from another. Before the training process begins, the dataset goes through preprocessing steps like normalizing pixel values to the 0–1 range, reshaping images to match the CNN input format, and converting class labels into one-hot vectors for classification.

Model is compiled which takes the rates automatically to achieve faster and more stable training. ReLU is applied in hidden layers to introduce non-linearity, while the Softmax function in the results probability scores across all ten-digit classes.

During training, the network fixes the weights through backpropagation to minimize errors. Dropout layers used to stop the out of the box fittings, which randomly deactivate a portion of neurons during training, forcing the network to generalize better instead of

memorizing the dataset. Training is carried out in batches, which makes the process more efficient and helps improve overall performance.

After several epochs (full passes through the training data), model is evaluated using the test set to assess its accuracy. CNN demonstrates high correctness in analysing the written single digit number, even when handwriting styles vary significantly. Errors, when it occurs, usually digits that look very similar, such as 3 and 8 or 4 and 9. By analyzing these misclassifications, the model can be fine-tuned further. Data augmentation techniques—like slight rotations, shifts, or zooming—can also be applied to the training images to increase variety and strengthen the model's generalization ability.

Once fully trained and validated, the system can be integrated into practical applications. It may be deployed in web apps, desktop tools, or mobile applications, where users can write digits using a touchscreen or stylus, and the system instantly recognizes and digitizes the input. It can also be embedded into OCR pipelines to automate document digitization, creating smarter and more interactive interfaces.

Because of its lightweight structure and fast processing, the model is suitable not only for online platforms but also for offline or embedded environments, such as Raspberry Pi or smartphones.

In summary, this project illustrates how deep learning, particularly CNNs, can solve real-world digit identification functions with enhanced reliability and efficiency. By leveraging Python along with TensorFlow, Keras, and the MNIST dataset, the system is built in a way that is both academically rigorous and practically relevant.

II. LITERATURE REVIEW

Handwritten digit identification has been a widely studied problem in the fields of Artificial intelligence techniques, image interpretation and pattern recognition. Traditional approaches to this problem relied heavily on handcrafted features and classical machine learning algorithms, that is: k-Nearest Neighbors (k-NN), Support Vector Machines (SVM), and Decision Trees. Although these techniques offered justifiable accuracy, they were frequently restricted through their inability to generalize well to varied handwriting styles and noisy data. A significant advancement came with the introduction of the MNIST dataset by Yann LeCun et al., which standardized the evaluation of handwritten digit recognition systems. The dataset contains 60,000 training and 10,000 test grayscale images of handwritten digits (0–9), and has since become a benchmark for testing classification algorithms.

With the emergence of deep learning, especially Convolutional-based Neural Networks (CNNs), the correctness and robustness of digit recognition systems have dramatically improved. LeCun's pioneering work on LeNet-5 demonstrated that CNNs are extremely efficient in capturing spatial hierarchies in image data, rendering them particularly fitted for image identification methods. CNNs eliminate the requirement for manually selected feature extraction by learning structured in levels depiction straight from pixel data.

Subsequent studies have refined CNN architectures using methods such as Rectified Linear Unit (ReLU) activation, dropout-based regularization, data augmentation, and batch normalization, further improving performance. Research by Cireşan et al. (2012) showcased deep neural networks enhanced through the GPU acceleration, achieving state-of-the-art results on MNIST at that time.

Recent advancements in frameworks like TensorFlow and Keras have simplified the implementation of deep learning models, enabling rapid prototyping and experimentation. These tools provide powerful APIs and GPU support, making them widely adopted in both academia and industry.

In summary, the development in differentiation with typical machine learning techniques to deep learning has significantly enhanced the performance of handwritten digit identification systems. This project based upon this progress by implementing a CNN-based model using TensorFlow and Keras, trained and evaluated on the MNIST dataset.

III. METHODOLOGY

A. Data Collection

The MNIST dataset is used as the primary dataset for this project. It contains 70,000 grayscale images of handwritten digits (0–9), with the training set comprises 60,000 images, while the test set contains 10,000 samples. for testing. Each image is 28x28 pixels in size and labelled with the correct digit.

B. Data Reprocessing

To prepare the dataset for training, the following preprocessing steps are applied:

1) Normalization: Pixel values are scaled the values from 0-255 to a normalized range of 0-1.
2) Reshaping: Data is reshaped to fit the shape of the input data expected by CNNs (28x28x1).
3) One-Hot Encoding: Digit labels are transformed into categorical format for multi-class classification.

C. Model Design

A Convolutional Neural Network (CNN) is designed to extract spatial features from the images and classify digits. The architecture typically includes:

1) CNN layers with Rectified Linear Unit (ReLU) activation used for feature learning.
2) Max Pooling layers for spatial dimensionality reduction
3) Dropout layers for regularization to prevent overfitting
4) Fully connected (Dense) layers for classification
5) Softmax output layer for predicting digit probabilities

D. Model Implementation

The framework is implemented using Keras with a TensorFlow backend. Key implementation steps include:

1) Defining the CNN architecture
2) Compiling the model with categorical cross-entropy loss and an optimizer (e.g., Adam)
3) The model is trained using suitable batch size and
4) Monitoring accuracy and loss using validation data

E. Model Evaluation

After training, the model is evaluated on the test dataset using:

1) Accuracy score
2) Confusion matrix
3) Loss and accuracy plots over training epochs

F. Optimization and Tuning

To improve performance, hyperparameters including learning rate, number of filters, kernel size, batch size, and dropout rate may be tuned. Furthermore, data augmentation method can be used to expand model robustness.

IV. FLOWCHART

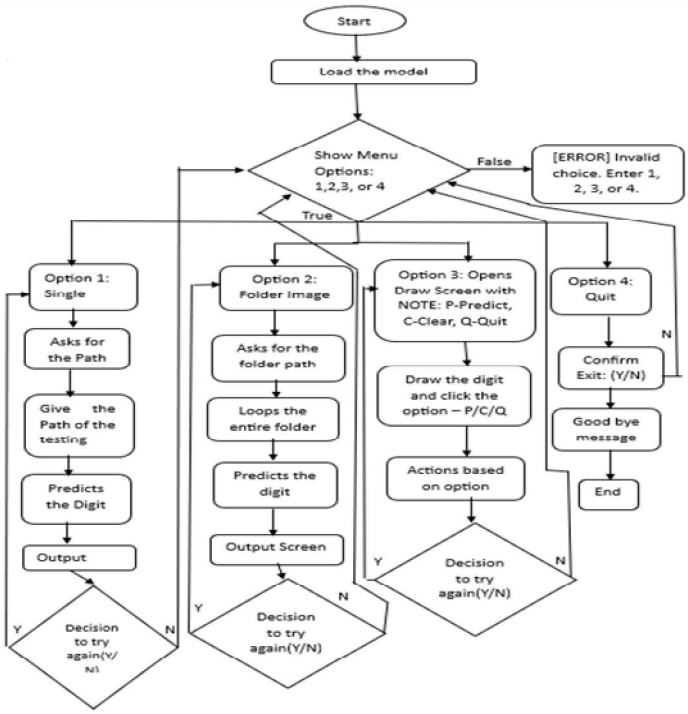


Figure 1: Flow Chart

The flowchart illustrates the overall workflow of the digit recognition system, describing how the model interacts with the user and processes inputs to generate outputs. It provides a clear representation of the decision-making process to recognise the task. It is easy to understand and easy to handle, offering different modes of input and ensuring smooth execution through error handling and retry options.

The process begins with the initialization stage, where the trained model is loaded into memory. This step is essential, as the accuracy of predictions depends on the pre-trained neural network model being successfully available. Once the model is loaded, the program presents the user with a menu containing four distinct options. These options are designed to provide flexibility in how the user wishes to test the digit recognition system: a single image input, a folder of images, a drawing interface, or a safe exit from the program.

A. Option 1: Single Image Input

When the user selects Option 1, the system prompts for the path of a single image file that is to be tested. The given image is preprocessed and then fed into the model for prediction. The model outputs the predicted digit, which is displayed to the user. After the result is shown, the program offers the user the opportunity to continue by processing another image or to exit this mode. This retry mechanism ensures flexibility and efficiency when testing multiple images one by one.

B. Option 2: Folder of Images

In Option 2, here it will process a collection of images not the individual picture/Digit. The user is required to provide the folder path containing the test images. The program automatically loops through the entire folder, applying the same preprocessing and prediction steps to each file.

The predictions are displayed sequentially on the output screen, providing the user with bulk testing functionality. Similar to Option 1, once the results are displayed, again it will ask to repeat the process or not to the user. This option is especially useful when evaluating the model's performance on a dataset of multiple images.

C. Option 3: Draw Screen Interface

Option 3 provides an interactive approach, where the user can handwrite a digit on a drawing screen. This mode gives current handwriting input and is examine the digit written by the performer. The Screen shows the 3 Parameter as NOTE.:

P if for the Predict: Grasps the drawn digit and predicts the the drawned digit on the drawn screen window.

C if for the Clear: It allow the user to redraw after clearing the Draw Screen

Q if for Quit: IT comes out from the draw screen mode and goes to the main menu window.

This option enhances user engagement by enabling practical testing without the need to prepare image files in advance. Once an action is performed, the system again prompts whether the user wishes to continue.

D. Option 4: Quit

If the user selects Option 4, the system requests confirmation before termination. This ensures that the program does not exit accidentally. Upon confirmation, a goodbye message is displayed, and the program ends gracefully. If the user chooses not to quit, control is returned to the main menu.

Error Handling

The flowchart also accounts for invalid inputs. If the operator inputs a choice outside the range of 1–4, the system generates an error message prompting them to provide a valid option. This feature ensures robustness and prevents unexpected crashes or undefined behaviour during execution.

Decision to Retry

Across all options, a common element is the decision point where there will be getting confirmation to the user which asks whether wanted to try again. This loop provides flexibility, allowing continuous testing without restarting the program. It also makes the system more interactive and convenient for repeated usage.

V. BLOCK DIAGRAM

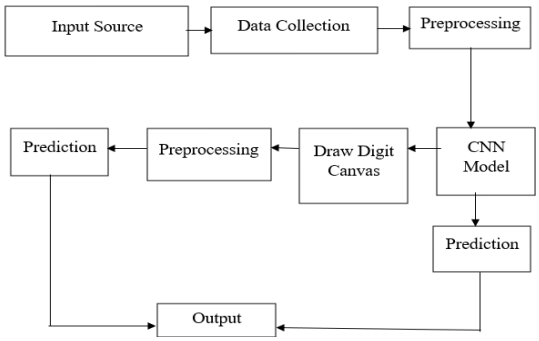


Figure 2: Block Diagram

The digit recognition - a powerful application of computer vision and deep learning techniques that enables automated recognition of handwritten or digitally created numerical digits. The block diagram shows a visual representation, illustrating how input data is transformed into meaningful output. Each block in the diagram represents a specific step in the processing pipeline, ensuring clarity and logical flow from image to final output. Designed to be robust, flexible, and user-friendly, making it suitable for both pre-existing datasets and real-time inputs.

A. Input Image Acquisition

The first and fundamental block in the digit recognition workflow is Input Image Acquisition. This block gives the raw data and moves to the system to continue the process. This is structured to maintain the picture from three multiple sources:

1) Single Image File: A user can specify the path of a single digit image. This is helpful for testing specific cases or validating the model with chosen examples.

2) Folder Containing Multiple Images: A directory of picture provided, allowing to process them sequentially. This method mainly adapted in folder wise checking, dataset evaluation, or automated testing.

3) Live Drawing Interface: Users can draw digits in real time using a graphical interface. The interface includes options like P (Predict), C (Clear), and Q (Quit) to support interactive digit recognition. This feature makes the system more versatile by demonstrating its ability to handle dynamic user input.

By supporting multiple input modes, the system achieves flexibility, making it suitable for both pre-collected datasets and live user-generated samples.

B. Image Preprocessing

Once Raw images often contain chaos, variations in size, all of which may affect neural network behaviour. Preprocessing takes care of that the input is standardized and optimized for analysis by the Convolutional Neural Network (CNN).

Typical preprocessing steps include:

1) Grayscale Conversion: Images are migrated from red, green, blue color to grayscale. Since digit recognition depends mainly on shapes and edges rather than color, computational complexity will be reduced by doing this step while preserving essential details.

2) Resizing: The image is resized to a 28×28-pixel format, which is the standard input dimension used for MNIST-trained CNN models. Uniform image size is crucial to ensure consistent processing across different inputs.

3) Normalization: Pixel values are scaled to a range between 0 and 1. Normalization improves the stability of the neural network during prediction and results that training dataset's image intensity is accurate.

This preprocessing step gives high accuracy and robustness, as it aligns the input data with the conditions under which the model was trained.

C. CNN Model Prediction

The heart of the digit recognition system is the Convolutional Neural Network (CNN) Prediction Block. This block takes care of the inputs given in the picture format. The following is the steps it takes care:

1) Convolutional Layers: This block remove the disturbance in the input image. Each filter gives the current patterns in the picture, and gives the individual characteristics of the digit
2) Pooling Layers: This is the second step where it will decrease the size of the feature maps without removing the important information.
3) Fully Connected Layers: After the Pooling Layers performance the extraction, the result from the first layer and pooling layers are flattened and moved to the fully connected layers.
4) Softmax Activation: This is the last Layer where the where it will give the accurate probability distribution of all the Ten digits.

The CNN model is treated a the most important of the Digiti Recognition systems. It is Structured layer which enables to capture the patterns given and gives very effective generalized result at all the different written styles of the Digit

A. *Output Display*

The final block in the workflow is Output Display and Optional Text-to-Speech (TTS). Post completion of the CNN prediction, the system gives the output to the performer in the clear and interactive manner:

1) Display: This is used to Show the predicted digit on the interface to view the result by the user. For the folder inputs, The output products one after the other.
2) Optional TTS: It announces the predicted digit loudly. Engine pyttsx3 is used for the announcements.

Digit recognition system's block diagram illustrates this structured flow from the current image input to final prediction. The work begins with image acquisition, which supports multiple input modes. Next, preprocessing standardizes the image to improve recognition accuracy. The CNN prediction block then extracts features and classifies the digit where the technique used is deep learning. Finally, the result is communicated back to the performer visually and, if enabled, through speech.

Each block contributes meaningfully. The smooth integration of all components improves reliability and makes the system easier to understand, implement, troubleshoot, and upgrade in the future.

VI. SOFTWARE AND TOOLS/LIBRARIES USED

A. *Software:*

1) *Visual Studio Code (IDE)*

- Description: The IDE used for coding, debugging, and running the project. A simple but feature-rich source code editor
- Role in Project: Used as the development environment to write, debug, and execute Python scripts
- Features Used: Code editing, terminal integration for running Python, extensions for Python support.

B. *Tools Used*

The following software tools and libraries are included for the construction and operation of the handwritten digit recognition system:

1) *Programming Language: Python 3.5+*

- Description: Python is a widely used interpreted language, recognized for its high-level structure and simplicity. and extensive library support.
- Reason for Use: Python provides a broad range of repositories for machine learning (ML) and deep learning. It is also supported by major frameworks like TensorFlow and Keras, making it ideal for rapid prototyping and development.
- Version Used: Python 3.5 or later

2) *TensorFlow*

- Description: An open-source environment for deep learning research framework for mathematical analysis and building machine learning (ML) models.
- Role in Project: Used as the backend framework to build and train deep learning models.
- Version: For Python 3.5: TensorFlow 2.1.0 or earlier and for Python 3.8+: TensorFlow 2.13.0 or later

3) *Keras*

- Description: A high-level neural network API written in Python, capable of running on top of TensorFlow.
- Role in Project: Simplifies the creation and training of deep learning models, especially CNNs for image classification.
- Version: For Python 3.5: Keras 2.2.4

4) NumPy

- Description: A primary toolkit for scientific computing with support for large, multi-dimensional arrays and matrices.
- Role in Project: Used for efficient numerical operations, especially during data preprocessing.
- Version: Compatible with Python 3.5+

5) OpenCV (cv2)

- Description: An open-source vision computing and image analysis library.
- Role in Project: Used for reading digit images, converting them to grayscale, resizing, preprocessing (thresholding, normalization), and handling user-drawn digits.
- Version: Advanced stable version compatible with Python 3.8+ (opencv-python package).

6) pyttsx3

- Description: A Python library for Text-to-Speech (TTS) conversion that works offline.
- Role in Project: Used for announcing predicted digits and giving user instructions through voice feedback in the digit recognition system.

VII. ADVANTAGES and APPLICATIONS

A. Advantages

The digit recognition system offers several notable advantages, making it highly effective and widely applicable across various domains:

1) High Accuracy: The system leverages a Convolutional Neural Network (CNN), which is capable of learning complex patterns in handwritten digits. This ensures accurate recognition even in cases of varied handwriting styles, distortions, or noise in the input images.

2) Scalability: The system can handle both single images and large datasets efficiently. It can process multiple images in batches or work with real-time inputs from users, making it suitable for small-scale experiments as well as large-scale industrial applications.

3) Applications in Postal Systems: Automated digit recognition can be used for reading postal codes on envelopes and packages, significantly speeding up mail sorting and delivery processes. This reduces manual effort and minimizes human errors.

4) Applications in Banking: Banks can use digit recognition systems for processing handwritten cheques (only for single digit – can considered for Sr.No), or other financial documents. Improves efficiency, processing time will be reduced, and ensures accuracy in financial transactions.

B. Apllications

The digit recognition system has a wide range of potential applications across various domains. Its usability can extend beyond traditional scenarios, providing innovative solutions in everyday life and educational settings:

1) Elevators – Hygiene-Conscious Input: In public or commercial buildings, the system could be integrated with elevators to allow users to input floor numbers using a smartphone or touchless device. This reduces physical contact with buttons, promoting hygiene in environments where minimizing touch is important.

2) Voting Systems: The system can be adapted for small-scale voting applications, such as in classrooms, colleges, or local organizations. By writing a digit corresponding to their choice (1–9) the user can put their votes, and the system automatically records and counts the votes, ensuring efficiency and accuracy

3) Toy Applications and Educational Apps: Digit recognition can be integrated into educational software or smart toys for children. Kids can practice writing digits (0–9) on a tablet or device, and the system provides instant feedback on correctness. This makes learning interactive, fun, and helps in developing handwriting skills at an early age.

VIII. CONCLUSIONS

This project effectively exhibits the significant potential of deep learning techniques, particularly convolutional neural networks (CNNs), in addressing real-world problems such as handwritten digit recognition.

Handwritten digit recognition is a core challenges in computer vision and has a key function in tasks that demand both automation and correctness. In this implementation, the well-known MNIST dataset, which consists of 70,000 grayscale images of digits (0–9), served as the foundation for model development and evaluation the recognition system.

A convolutional-based neural network (CNN) was built using TensorFlow and Keras framework, and the system achieved strong performance, confirming the effectiveness of deep learning techniques in solving image classification problems.

The practical significance of digit recognition is considerable. Postal departments can speed up mail handling by automatically reading zip codes, banks can process checks more efficiently, and organizations can digitize handwritten records to minimize manual work and errors. Beyond these, digit recognition technology can also be applied in educational apps, electronic voting systems, and other platforms that require interpretation of numerical input from handwritten data.

The success of this project demonstrates that deep neural network techniques can reliably handle complex pattern recognition tasks. CNNs, in particular, are effective due to the fact that they can retrieve significant patterns immediately from raw images without the need for handcrafted features. Traditional approaches required domain-specific feature engineering, which was often tedious and less adaptable. By contrast, the CNN model used here automatically learned multiple layers of representation. Early convolutional layers detected simple shapes like edges and curves, while deeper layers combined these features to identify entire digit structures. This hierarchical learning enabled the system to recognize a wide variety of handwriting styles with high accuracy.

Additionally, the project emphasized the importance of good preprocessing and thoughtful model design. Normalizing images and encoding labels were crucial for stable and efficient training, while dropout regularization helped reduce overfitting, making the model more reliable on unseen test data. Collectively, such techniques validated in which the system was both accurate and robust.

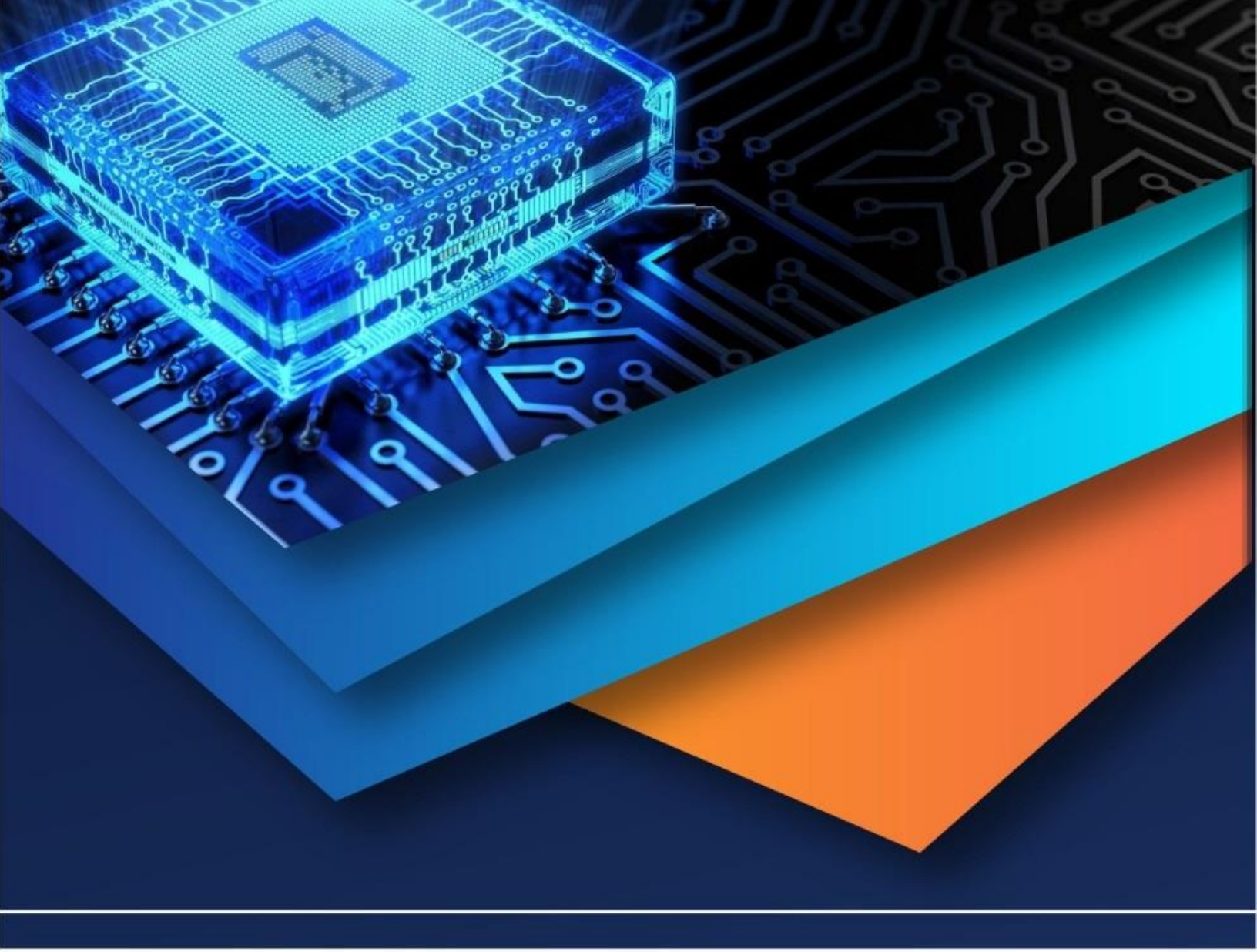
Additionally, the use of Keras and TensorFlow allowed for efficient construction, training, and evaluation of the CNN architecture, illustrating how modern deep learning frameworks can accelerate the development of intelligent systems. These tools also facilitated visualization of training progress, monitoring of accuracy and loss metrics, and experimentation with different hyperparameters to optimize model performance.

Testing results confirmed that the CNN model generalized well to unseen data, maintaining high classification accuracy that created the fitting for operationalization in real-time automated systems. The model's performance underscores the practical viability of using deep neural networks for image categorization tasks, offering a reliable alternative to manual or traditional computational methods. The success of current technique not only checks the efficacy of CNNs in pattern recognition but also introduces the opportunities for extending similar techniques to more complex tasks, such as multi-class object recognition, handwritten text recognition, and other areas where pattern variability is significant.

In conclusion, this project illustrates the practical implementation and benefits of a deep learning-based handwritten digit recognition system. Beyond its immediate technical accomplishments, it emphasizes the transformative potential of AI technologies in automating and enhancing activities that were typically executed manually. The methodologies employed—including data preprocessing, CNN architecture design, and model training—provide a strong foundation for future exploration. Prospective task would be able to include evaluating using additional modern neural frameworks systems such as residual networks (ResNets) or recurrent CNNs, integrating the model into live real-time applications, or extending its capabilities to recognize a wider variety of handwritten or printed characters. Ultimately, this project reinforces the role of deep learning as a powerful tool in intelligent automation, demonstrating its capacity to improve efficiency, accuracy, and reliability across diverse practical domains.

REFERENCES

[1] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324. https://doi.org/10.1109/5.726791
[2] Deng, L. (2012). The MNIST database of handwritten digit images for machine learning research. IEEE Signal Processing Magazine, 29(6), 141–142. https://doi.org/10.1109/MSP.2012.2211477
[3] Chollet, F. (2015). Keras: The Python Deep Learning library. https://keras.io
[4] Abadi, M., Barham, P., Chen, J., et al. (2016). TensorFlow: Large-scale machine learning on heterogeneous systems. arXiv preprint arXiv:1603.04467. https://tensorflow.org
[5] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. http://www.deeplearningbook.org
[6] Chollet, F. (2017). Deep Learning with Python. Manning Publications.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY