**Handwritten Digit Recognition using Neural Network**

*A Project Based Learning Report Submitted in partial fulfilment of the requirements for the award of the degree*

*of*

**Bachelor of Technology in**

**Department of Electronics and Communication Engineering**

**22AIP3305A- DEEP LEARNING**

Submitted by

2210040002- Yadnya Vidhate

2210040076- Krishn Chand Kewat

2210040079-N.Srujan Kumar

2210040112-Shaik Shahna

2210040116-M.Gowthami

Under the guidance of

**Dr. Sumit Hazra**



Department of Electronics and Communication Engineering

Koneru Lakshmaiah Education Foundation, Aziz Nagar

Aziz Nagar – 500075

**Topic : Handwritten Digit Recognition using Neural Network**

**1. Project Overview:**

Handwritten digit recognition using MNIST dataset is a major project made with the help of Neural Network. It basically detects the scanned images of handwritten digits.

We have taken this a step further where our handwritten digit recognition system not only detects scanned images of handwritten digits but also allows writing digits on the screen with the help of an integrated GUI for recognition.

**2. Key Concepts:**

1. **Dataset**:

He most commonly used dataset for handwritten digit recognition. It contains **60,000 training** and **10,000 testing** grayscale images of digits (0-9) of size **28x28 pixels**.

1. **Data Preprocessing:**
   * **Normalization**: Pixel values (0-255) are scaled to a range of 0-1 or -1 to 1 to improve training efficiency.
   * **Reshaping:** Ensuring input shape is compatible with the neural network (e.g., from (28,28) to (28,28,1) for CNNs).
   * **One-Hot Encoding:** Labels (digits 0-9) are converted into a binary vector representation (e.g., digit 2 → [0,0,1,0,0,0,0,0,0,0]).
2. **Neural Network Architectures:**

* **Feedforward Neural Network (FNN):**

**-** Fully connected Multi-Layer Perceptron (MLP)

**-** Uses activation functions like ReLU, Softmax

**-** Suitable for basic digit recognition tasks

* **Convolutional Neural Network (CNN) (Preferred for Image Recognition)**
  + - Layers: Convolution → ReLU → Pooling → Fully Connected
    - Extracts spatial features automatically
    - Example architecture:
    - Conv2D → ReLU → MaxPooling → Flatten → Dense → Softmax
* **Recurrent Neural Network (RNN)**
  + - Rarely used but can be helpful if sequential dependencies exist (e.g., handwritten sequences)

1. **Training the Neural Network:**

* **Loss Function:**
  + - Categorical Crossentropy for multi-class classification
* **Optimizer:**
  + - Adam, SGD, RMSprop for weight updates
* **Backpropagation & Gradient Descent:**
  + - Adjusts weights using learning rate
* **Epochs & Batch Size:**
  + - Determines how many times the model sees the data
* **Regularization Techniques:**
  + - Dropout, L2 Regularization to prevent overfitting

1. **Model Evaluation:**
   * Accuracy, Precision, Recall, F1-score
   * Confusion Matrix to analyze misclassifications
   * Test on unseen data (Generalization)
2. **Deployment & Optimization:**
   * Convert model to ONNX, TensorFlow Lite, or Edge AI for deployment
   * Use GPU acceleration (CUDA, TensorRT) for faster inference
   * Optimize with Quantization, Pruning, or Knowledge Distillation

**3. Steps in Building the Project:**

1. **Setup Environment:**

* Install required libraries: pip install tensorflow keras numpy matplotlib
* Import necessary modules (tensorflow, keras, numpy, matplotlib).

1. **Load and Preprocess Data:**

* Use the **MNIST dataset** (built into TensorFlow/Keras).
* Normalize pixel values to **[0,1]** range.
* Convert labels to **one-hot encoding**.

1. **Build the Neural Network Model:**

* Choose architecture: MLP (Dense) or CNN.

Example CNN layers:

* Conv2D → ReLU → MaxPooling.

1. **Compile and Train the Model:**

* Choose loss function: categorical\_crossentropy.
* Use optimizer: Adam, SGD, or RMSprop.
* Train the model using epochs & batch size.

1. **Evaluate the Model:**

* Test the model on unseen test data.
* Check accuracy and confusion matrix.

1. **Make Predictions:**

* Use model.predict() on new digit images.
* Convert predicted probabilities to digit labels**.**

1. **Save and Deploy the Model:**

* Save model using model.save('digit\_model.h5').
* Deploy using Flask, TensorFlow Lite, or a web interface.

**4. Outcome of the Project:**

1. **Successful Recognition of Handwritten Digits:**

* The trained neural network can accurately recognize digits (0-9) from handwritten input images.
* High accuracy on the MNIST dataset (typically >98% for CNN models).

1. **Model Generalization & Performance Metrics:**

* Accuracy, Precision, Recall, F1-score for performance evaluation.
* A confusion matrix to analyze misclassified digits**.**

1. **Interactive Prediction System:**

* The system can take user input (e.g., drawing digits on a GUI/web app).
* Real-time predictions for handwritten digits.

1. **Model Deployment & Practical Applications:**

* The trained model can be deployed in various ways:

- Web app (Flask/Django, Streamlit).

- Mobile app (TensorFlow Lite).

- Embedded systems (Raspberry Pi, Edge AI).

**5. Challenges Faced:** One of the primary challenges faced in the Handwritten Digit Recognition project was dealing with the variability in handwriting. People write digits in many different styles, with varying slants, sizes, and orientations, which can make consistent recognition difficult. Additionally, noisy and incomplete data, such as smudged digits or poorly scanned images, further complicates the task, leading to inaccurate predictions. Overfitting was also a concern, as the model might perform well on the training dataset but fail to generalize effectively to new, unseen data. Finally, optimizing the neural network for deployment on edge devices or in real-time applications posed computational challenges, as training deep learning models requires significant processing power and memory.

**6. Future Enhancements:**

Future enhancements for the Handwritten Digit Recognition project could focus on improving model accuracy by experimenting with more advanced architectures such as hybrid CNN-RNN models or transformer-based models. Additionally, incorporating data augmentation techniques to generate more diverse training data can help the model generalize better. Another enhancement would be to optimize the model for faster inference by using techniques like quantization and pruning, allowing it to run efficiently on mobile or edge devices. Finally, the system could be expanded to recognize handwritten text or other languages, broadening its applicability in real-world scenarios.

**7. Conclusion:**

The Handwritten Digit Recognition project demonstrates the effectiveness of neural networks, particularly CNNs, in accurately classifying handwritten digits. By utilizing datasets like MNIST and applying various deep learning techniques, the model achieves high performance and can be deployed for practical applications such as automation and data entry. Despite challenges like handwriting variability and computational constraints, the project highlights the potential of deep learning in solving real-world problems, with room for further enhancement through model optimization and broader application expansion.