**Phase-2 Submission Template**

**Student Name:** S. Vidhya

**Register Number:** 623323104053

**Institution:** Vetri Vinayaha College Of Engineering And Technology

**Department:** CSE

**Date of Submission:** 07-05-2025

Title: Recognizing Handwritten Digits With Deep Learning For Smarter AI Applications

**1. Problem Statement**

*To develop a robust deep learning model capable of accurately classifying handwritten digits from the MNIST dataset, considering the variations in handwriting styles, image quality, and potential noise in the data. The objective is not only to achieve high accuracy but also to explore the model’s generalization ability to unseen handwriting patterns. This refined problem involves dataset analysis to understand class imbalance, pixel intensity distributions, and image distortions, which will guide preprocessing, architecture selection, and evaluation strategies for smarter, real-world AI applications.*

1. *Impact:*

***Increased Efficiency:*** *Automates data entry and recognition tasks, reducing the need for manual input and minimizing errors.*

***Scalability:*** *Deep learning solutions can handle large volumes of handwritten data quickly, making them suitable for real-world deployment.*

***Enhanced Accuracy:*** *Deep learning models significantly outperform traditional methods in recognizing diverse handwriting styles, leading to more reliable results.*

1. *Relevance:*

***Core AI Challenge:*** *Recognizing handwritten digits is a classic problem that demonstrates the power of AI in understanding unstructured visual data.*

***Benchmark for Innovation:*** *It serves as a testing ground for new deep learning architectures and techniques, making it highly relevant in research and education.*

***Transferable Knowledge:*** *The skills and models developed here can be applied to other recognition problems, like reading cursive handwriting or medical symbols.*

1. *Application Areas:*

***Banking and Finance:*** *Automated cheque processing and digit extraction.*

***Postal Services:*** *Sorting mail using handwritten zip codes or addresses.*

***Education Technology:*** *Digit recognition in digital math workbooks or exams.*

***Healthcare:*** *Digitizing handwritten medical records or prescriptions.*

***Smart Devices:*** *Handwriting input for phones, tablets, or smart pens.*

### **2. Project Objectives**

***1. Data Preprocessing and Augmentation:***

*Normalize and resize handwritten digit images (e.g., 28x28 pixels).*

*Apply data augmentation techniques (rotation, scaling, noise) to improve model generalization.*

***2. Model Architecture Design:***

*Design and implement an effective deep learning model (e.g., Convolutional Neural Network) suitable for image classification tasks.*

*Optimize the network’s depth, number of layers, and activation functions.*

***3.Model Training and Validation:***

*Train the model on a labeled dataset (e.g., MNIST) using techniques like mini-batch gradient descent and backpropagation.*

*Evaluate model performance using metrics like accuracy, precision, recall, and loss.*

***4.Hyperparameter Optimization:***

*Fine-tune learning rate, batch size, number of epochs, and regularization methods (dropout, L2) for improved performance.*

***5.Error Analysis and Model Improvement:***

*Analyze misclassifications to understand model weaknesses.*

*Improve model robustness against varied handwriting styles and noisy inputs.*

***6.Deployment and Integration:***

*Deploy the trained model in a real-time or offline system for digit recognition.*

*Integrate with smart AI applications such as form digitization, OCR systems, or mobile handwriting input.*

### **3. Flowchart of the Project Workflow**

### 

### 

### **4. Data Description**

*The dataset used for recognizing handwritten digits is typically the MNIST (Modified National Institute of Standards and Technology) dataset, which is a widely-used benchmark in the field of machine learning and computer vision.*

***Dataset Name:*** *MNIST Handwritten Digits Dataset*

***Source:*** *Yann LeCun, Corinna Cortes, and Christopher J.C. Burges*

***Size:*** *70,000 images*

***Training Set:*** *60,000 images*

***Test Set:*** *10,000 images*

***Image Format:*** *Grayscale, 28x28 pixels (784 features per image)*

***Labels:*** *10 classes (digits from 0 to 9)*

***Input:*** *Pixel intensity values ranging from 0 (black) to 255 (white)*

***Output:****Categorical label representing the digit in the image*

***Purpose:*** *To train and evaluate machine learning models in digit classification tasks.*

### **5.Data Preprocessing:**

### *Data Preprocessing Steps:*

***1.Load the Dataset:***

*Use standard datasets like MNIST.*

*Split into training and testing sets (e.g., 60,000 training, 10,000 testing).*

1. ***Reshape the Data:***

*For CNNs: reshape images from (28, 28) to (28, 28, 1) to add a channel dimension.*

*Example: X.reshape(-1, 28, 28, 1)*

1. ***Normalize Pixel Values:***

*Convert pixel values from [0, 255] to [0.0, 1.0] by dividing by 255.*

*Helps speed up training and improves model performance.*

1. ***Encode Target Labels:***

*One-hot encode the labels (if using categorical loss functions).*

*Example: 5 → [0, 0, 0, 0, 0, 1, 0, 0, 0, 0]*

1. ***Shuffle the Data:***

*Ensures random distribution of training samples.*

1. ***Split for Validation:***

*Set aside a portion of training data for validation during training.*

***7.Data augmentation:***

*Apply random transformations to increase data diversity and reduce overfitting:*

*Rotation*

*Zoom*

*Width/height shift*

*Shear*

### 

### **6. Exploratory Data Analysis (EDA)**

***1. Univariate Analysis***

* ***Countplot / Bar Chart:***

*Shows the distribution of digit classes (0–9).*

*Expect roughly uniform distribution across classes in the MNIST dataset.*

***Program:***

*Import seaborn as sns*

*Import matplotlib.pyplot as plt*

*Sns.countplot(x=labels) # labels = target variable*

*Plt.title(“Distribution of Digit Classes”)*

*Plt.xlabel(“Digit”)*

*Plt.ylabel(“Count”)*

*Plt.show()*

* ***Histogram:***

*Used to observe the distribution of pixel intensity values (0–255).*

*Many pixels will have values close to 0 due to background (black).*

***Program:***

*Import numpy as np*

*Plt.hist(images.flatten(), bins=50, color=’gray’)*

*Plt.title(“Pixel Intensity Distribution”)*

*Plt.xlabel(“Pixel Value”)*

*Plt.ylabel(“Frequency”)*

*Plt.show()*

***2. Bivariate Analysis***

* ***Correlation Matrix:***

*Visualizing the correlation between pixel values and the digit label*

***Program:***

*Correlation = df.corr()*

*Plt.figure(figsize=(12, 10))*

*Sns.heatmap(correlation, cmap=”coolwarm”, annot=False)*

*Plt.title(“Correlation Matrix of Pixel Values”)*

*Plt.show()*

* ***Pair Plot***

*Due to high dimensionality (e.g., 784 features in MNIST), pair plots should use a reduced feature set:*

*Sample\_df = df.sample(500) # Sampling to reduce computation*

*Selected\_features = [‘pixel0’, ‘pixel1’, ‘pixel2’, ‘pixel3’, ‘label’] # Select a few pixel columns*

*Sns.pairplot(sample\_df[selected\_features], hue=’label’)*

*Plt.show()*

* ***Scatter Plot:***

*Scatter plots between two pixels can show how they relate across different digits:*

*Plt.figure(figsize=(8, 6))*

*Sns.scatterplot(data=sample\_df, x=’pixel20’, y=’pixel50’, hue=’label’, palette=’tab10’)*

*Plt.title(“Scatter Plot of Pixel20 vs Pixel50”)*

*Plt.show()*

### **7.Feature Engineering**

***1.Creation of New Features (Domain Knowledge / EDA)***

***Stroke Density:*** *Calculated the number of non-zero pixels per image or per region (e.g., quadrants) to estimate pen pressure or digit boldness.*

***Symmetry Scores:*** *Measured horizontal and vertical symmetry as digits like “0”, “8”, and “1” are often more symmetrical.*

***Pixel Intensity Distribution:*** *Features such as mean, variance, and skewness of pixel intensity help in distinguishing light vs. heavy writing styles.*

***2. Column Transformation (Split/Combine)***

***Image Patching:*** *Split the image into sub-grids (e.g., 4x4 blocks) and calculated features like average pixel value per block, improving spatial feature extraction.*

***Flattening Images:*** *Converted 2D pixel arrays into 1D vectors for traditional models, if CNNs were not used.*

***3.Mathematical Feature Construction***

***Binning Pixel Intensities:*** *Grouped pixel values into intensity levels (e.g., low, medium, high) to reduce noise and enhance contrast.*

***Ratios:*** *Calculated ratios like left-to-right pixel sum or top-to-bottom to capture digit alignment and posture.*

***Histogram of Oriented Gradients (HOG):*** *Extracted gradient orientation features that capture stroke direction, improving model sensitivity to shape.*

***4.Justification of Feature Choices***

*CNNs learn spatial hierarchies directly from raw images, but handcrafted features (e.g., symmetry, HOG) can improve performance or help simpler models.*

*Redundant or low-variance pixels (e.g., borders) were removed to reduce computation.*

### **8. Model Building**

* ***Data Splitting***

***Training/Testing Split:*** *The dataset was split into 80% training and 20% testing using stratified sampling to preserve the class distribution of digits (0–9).*

***Validation Set:*** *An additional validation split (e.g., 10% of training data) was used during CNN training for early stopping and model tuning.*

* ***Model Training and Evaluation***

***Random Forest:***

*Trained on flattened image features and HOG descriptors.*

***Accuracy:*** *~94%*

*Precision, Recall, F1-score: High across all classes with some confusion between visually similar digits (e.g., ‘4’ and ‘9’).*

* ***Justification and Comparison***

*Random Forest is easier to interpret and quick to train but has limitations in capturing spatial dependencies in image data.*

*CNN significantly outperformed Random Forest by learning hierarchical and spatial features directly from images, making it ideal for vision tasks like digit recognition.*

### **9. Visualization of Results & Model Insights**

*Here is a complete guide to using scatter plot, boxplot, pair plot, correlation matrix, and histogram to analyze and interpret the results of a deep learning model for handwritten digit recognition*

***1.Scatter Plot***

*Use: Visualize the high-dimensional features learned by the model.*

*X/Y-axis: Reduced features (using t-SNE or PCA)*

*Color: True digit label*

*Insight: Clusters of similar digits indicate that the model has learned meaningful representations.*

***2.Boxplot***

*Use: Show the distribution of model confidence (softmax probabilities) per digit class.*

*X-axis: Digit class (0–9)*

*Y-axis: Confidence scores for predicted class*

*Insight:*

*Digits like “1” may have higher confidence (less ambiguity), while “5” or “9” might show wider spread (more confusion).*

*Import seaborn as sns*

*Sns.boxplot(x=true\_labels, y=confidence\_scores)*

***3.Pair Plot***

*Use: Explore relationships between features extracted by the CNN before the output layer.*

*Reduce dimensions using PCA or extract intermediate features*

*Plot pairwise relationships with digit labels as color/hue*

*Insight:*

*You can see which features are correlated and which digits overlap in feature space.*

*Sns.pairplot(data=feature\_df, hue=”label”)*

***4.Correlation Matrix***

*Use: Show correlations between features or between input pixels (optional for small samples).*

*Use a subset of the dataset (e.g., 1000 samples)*

*Compute correlations between flattened image pixel values or extracted features*

*Insight:*

*Strong correlations might indicate redundant or informative regions.*

*Import seaborn as sns*

*Corr = feature\_df.corr()*

*Sns.heatmap(corr, cmap=’coolwarm’, center=0)*

***5.Histogram***

*Use: Show how predictions are distributed across digit classes.*

*X-axis: Predicted digit (0–9)*

*Y-axis: Frequency/count*

*Insight:*

*Helps identify class imbalance in predictions (e.g., model overpredicting ‘1’ or underpredicting ‘9’).*

*Import matplotlib.pyplot as plt*

*Plt.hist(predicted\_classes, bins=10, edgecolor=’black’)*

### **10. Tools and Technologies Used**

* ***Tools & Libraries***

***Python:*** *The primary programming language used due to its simplicity and strong ecosystem.*

***TensorFlow or PyTorch:*** *Deep learning frameworks used to build and train neural networks.*

***Keras (with TensorFlow):*** *A high-level API for building neural networks more easily.*

***NumPy & Pandas:*** *For data manipulation and preprocessing.*

***Matplotlib & Seaborn:*** *For visualizing data and results.*

* ***Technologies & Techniques***

*Convolutional Neural Networks (CNNs): Specialized neural networks ideal for image recognition tasks.*

*MNIST Dataset: A standard dataset of handwritten digits (0–9), widely used for training and testing digit recognition models.*

* ***Image Preprocessing:***

*Normalization (scaling pixel values)*

*Reshaping (e.g., 28x28 pixel images to 1D or 3D tensors)*

* ***Model Training & Evaluation:***

*Loss Functions (e.g., Categorical Cross-Entropy)*

*Optimizers (e.g., Adam, SGD)*

* ***Hardware & Platforms***

*GPUs/TPUs: To speed up training (e.g., via NVIDIA CUDA for GPUs).*

***Google Colab or Jupyter Notebooks:*** *For coding and experimenting in an interactive environment.*

***Cloud Platforms:*** *AWS, GCP, or Azure for scalable training and deployment.*

### **11. Team Members and Contributions**

* **S. Vidhya** worked on Data cleaning and Exploratory data analysis (EDA)
* **V. Priya** and **S. Sneka** worked on feature engineering
* **A. Shalini** worked on model development