Digit Classifier Using CNN

A THESIS

Submitted by

Kaushal Soundararajan – CB.EN.U4ECE23125

Nidhil Remila Shivakumaran – CB.EN.U4ECE23129

Sanjay Subramanyan S– CB.EN.U4ECE23147

Shriyaank R – CB.EN.U4ECE23151

Vivyn M – CB.EN.U4ECE23157

in partial fulfilment for the award of the degree of

BACHELORS OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

23AIE232M - PYTHON FOR AI

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE - 641 112 (INDIA)

APRIL – 2025

Contents

Abstract

A Convolutional Neural Network (CNN) - based digit classifier efficiently recognizes handwritten digits by learning spatial hierarchies of features. Using the MNIST dataset, the model is trained to distinguish digits (0-9) by leveraging convolutional layers for feature extraction and fully connected layers for classification. The methodology involves designing a convolutional neural network model that gives the best results for the classification task. The proposed approach outperforms traditional machine learning techniques by automatically detecting patterns in pixel intensities. This method has practical applications in digitized document processing, postal automation, and banking systems. The project is expected to yield a highly accurate digit classification model which will showcase the potential of CNNs in addressing image recognition tasks.

Introduction

Background and Motivation:

Handwritten digit recognition is a fundamental task in computer vision with widespread applications in document digitization, postal automation, and banking systems. Traditional machine learning techniques rely on handcrafted features, which often fail to generalize across different handwriting styles. Convolutional Neural Networks (CNNs) have revolutionized image classification by automatically learning spatial hierarchies of features, making them highly effective for digit recognition tasks.

Problem Statement:

Recognizing handwritten digits accurately remains a challenging problem due to variations in writing styles, distortions, and noise in images. Conventional methods require extensive preprocessing and feature engineering, making them less adaptable to diverse datasets. The goal of this project is to develop a robust CNN-based classifier that can efficiently recognize and classify handwritten digits from 0-9 with high accuracy.

Objectives and Scope:

The primary objective of this project is to design and implement a CNN model for handwritten digit classification using the MNIST dataset. The model should achieve high accuracy while maintaining computational efficiency. The scope includes data preprocessing, model training, evaluation, and testing. Additionally, the project explores potential real-world applications such as automated form processing and check verification systems.

Literature Review

Summary of Existing Work:

Handwritten digit classification has been widely studied using traditional machine learning techniques such as k-nearest neighbours (KNN), support vector machines (SVM), and artificial neural networks (ANN). KNN is a simple yet effective method but becomes computationally expensive as the dataset size increases. SVM has shown good performance in digit classification but requires careful tuning of kernel functions. ANN, though capable of learning complex patterns, struggles with high-dimensional image data without feature extraction. These methods rely on handcrafted features, making them less adaptable to diverse handwriting styles.

How This Project Improves Upon Existing Solutions:

Unlike traditional approaches, this project utilizes a Convolutional Neural Network (CNN) that automatically learns spatial features from digit images, eliminating the need for manual feature extraction. CNNs improve classification accuracy by effectively capturing local and global patterns in images. Additionally, the proposed model integrates techniques such as data normalization and dropout regularization to enhance generalization and robustness. By leveraging CNNs, this project provides a more efficient and scalable solution for handwritten digit recognition compared to basic machine learning techniques.

Paper 1: A Robust Model for Handwritten Digit Recognition using Machine and Deep Learning Technique

Importance and Challenges:

- HDR is a key application of pattern recognition with significant real-world implications, such as banking, postal services, and digitized document processing.
- Challenges arise from varying handwriting styles, sizes, and orientations, necessitating robust models for accurate recognition.

Dataset:

- The MNIST dataset, containing 70,000 grayscale images of handwritten digits (60,000 for training, 10,000 for testing), is a standard benchmark for HDR research.

Other Machine Learning Approaches used:

1. Support Vector Machines (SVM):
    - Widely used for classification tasks, SVM achieves recognition accuracies of up to 97.91% on MNIST, but struggles with scaling for large datasets and achieving higher accuracies needed in sensitive domains.
2. K-Nearest Neighbour (KNN):
    - KNN performs well for HDR with accuracies up to 96.8% but is computationally intensive, especially with large datasets.

3. Comparison to CNN:
    - SVM and KNN achieve moderate accuracy but lack the scalability and feature extraction efficiency provided by deep learning models.
4. Convolutional Neural Networks (CNN):
    - CNN has emerged as a superior model for HDR, leveraging convolutional and pooling layers to efficiently extract and classify features.
    - Researchers have achieved high accuracies using CNN:
        - 98.5% with extended minimal CNN.
        - 99.2% with CNN coupled with gradient descent.
        - 99.21% using DL4J-based CNN.

Comparative Studies:

- CNN consistently outperforms traditional approaches like SVM and KNN in accuracy and computational efficiency, with models achieving over 99% accuracy on MNIST.

Limitations:

- Despite high accuracies, challenges remain in handling more complex handwriting datasets with cursive styles or multilingual characters.
- Higher accuracies may require deeper architectures, but this increases computational cost.

Paper 2: A robust CNN model for handwritten digits recognition and classification

1. Historical Progress:
    - Early work in the 1980s and 1990s on neural networks for HDR faced limitations due to lack of large datasets and computational power.
    - The resurgence of neural networks in the 2010s, powered by faster computers and datasets like MNIST, catalysed advances in HDR.
2. Significance of HDR:
    - Applications in online/offline recognition, bank-check processing, and postal-address interpretation.
    - Challenges include variability in handwriting styles, orientations, and sizes.
3. Traditional Techniques:
    - Techniques such as KNN and decision trees demonstrated moderate accuracies but struggled with scalability and feature extraction for complex datasets.
4. Deep Learning and CNNs:
    - CNNs have become the preferred architecture for HDR due to their ability to handle large image datasets without extensive preprocessing.
    - Convolutional layers effectively extract features, while pooling layers reduce dimensionality and computational complexity.
5. Limitations of Current Models:
    - Accuracy can be affected by cursive or less structured handwriting styles.
    - Incremental hidden layers in CNNs can lead to diminishing returns in accuracy due to overfitting or computational constraints.

Technologies and Libraries Used

Libraries:

- NumPy
- Pandas
- Matplotlib
- Scikit-Learn
- TensorFlow
- Seaborn

Methodology

Data Collection:

- Dataset: Digit dataset (CSV format)
- Source: Provided training dataset (digitrain.csv)

Data Preprocessing:

- Normalize pixel values to [0,1].
- Reshape data for CNN input (28x28 images with 1 channel).
- One-hot encode labels for multi-class classification.
- Split dataset into 80% training and 20% validation.

Exploratory Data Analysis (EDA):

- Display sample images.
- Use histograms and scatterplots to analyse label distributions.
- Compute class frequencies to detect imbalances.

Feature Engineering:

- No manual feature extraction (CNNs automatically extract relevant features).
- Data augmentation to improve generalization.

Model Building:

- Architecture:
    - Two convolutional layers (32 and 64 filters, 3x3 kernel, ReLU activation).
    - Max-pooling layers (2x2 pool size) for down-sampling.

- Dropout layers (0.25-0.5) to prevent overfitting.
- Fully connected layers with so□max activation for classification.

- Compilation:
  - Loss Function: Categorical Cross entropy
  - Optimizer: Adam
  - Metrics: Accuracy

Training & Evaluation:

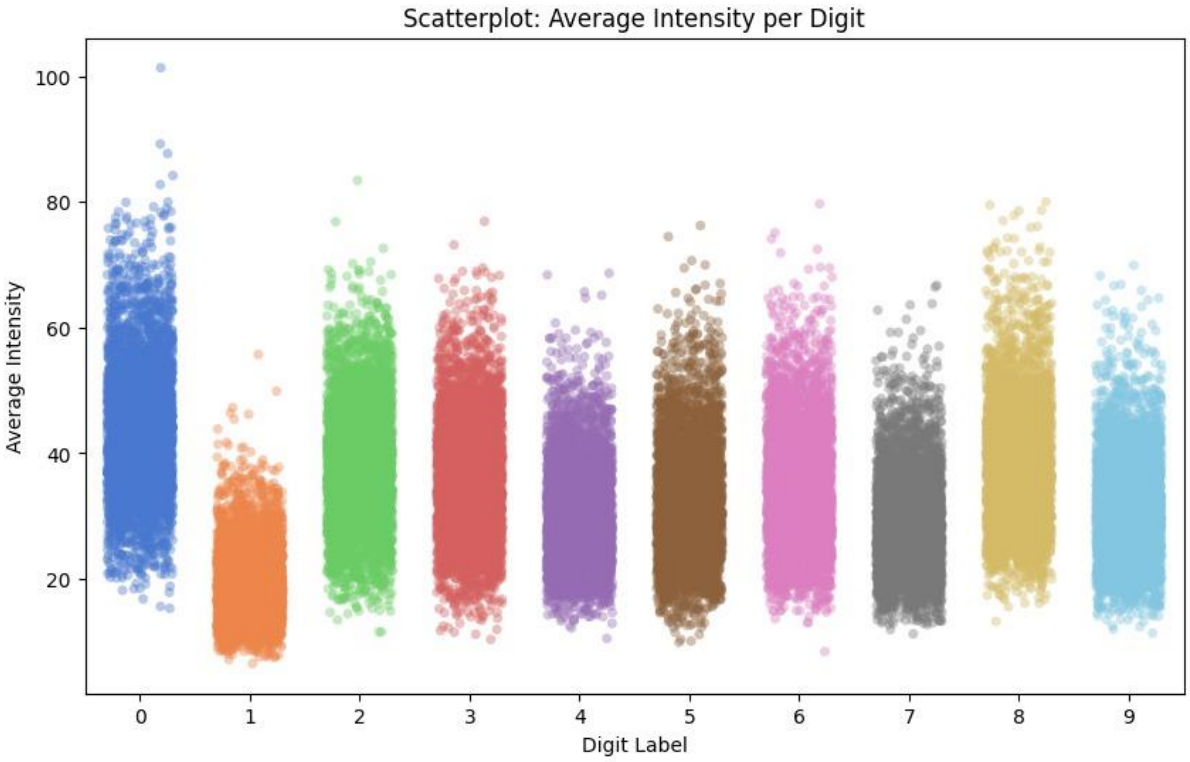
- Hyperparameters:
  - Epochs:10
  - Batch Size: 64
  - Learning Rate Scheduling: ReduceLROnPlateau
- Evaluation Metrics:
  - Accuracy, Loss
  - Confusion Matrix
  - Classification Report
- Feature Map Visualization:
  - Extract convolutional layer outputs to visualize feature maps for each digit.
  - Identify patterns learned by different layers.
- Accuracy & Loss Curves:
  - Training/validation accuracy and loss plotted over epochs.
  - CNN achieved 99% accuracy on validation data.

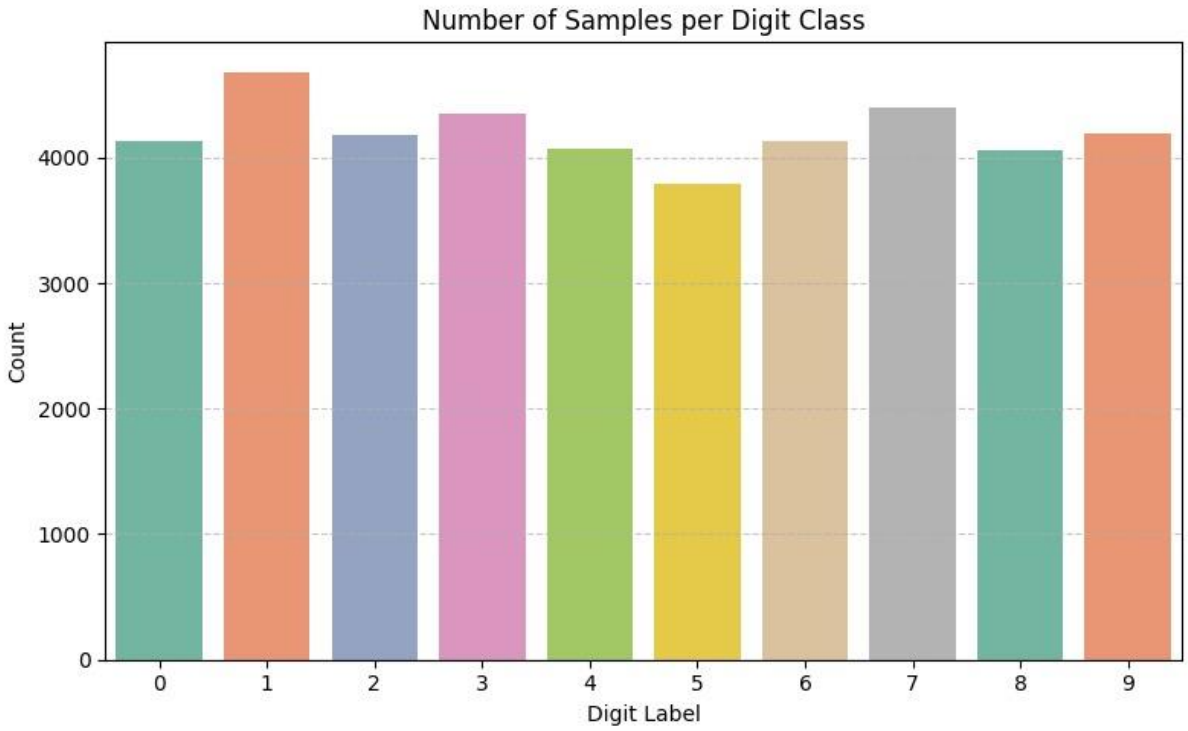
Proposed Work

- Successfully loaded and pre-processed dataset.
- Built and trained CNN model.
- Implemented EDA and feature visualization.
- Evaluated model performance.
- Generated confusion matrix and classification report.
- Further hyperparameter tuning.
- Testing with additional datasets for generalization.
- User interface to receive inputs has been implemented.

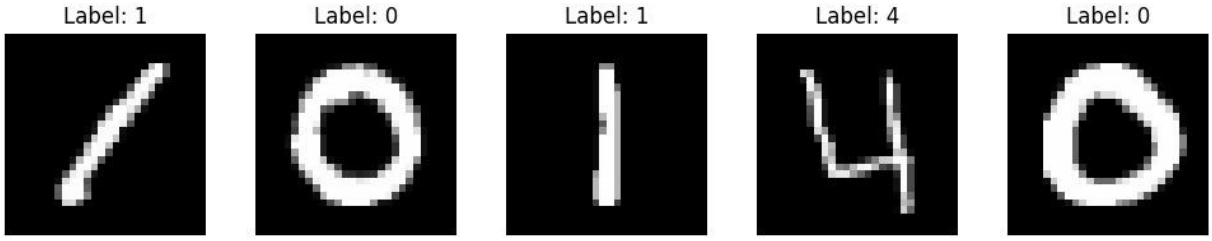
Results

EDA:


Scatterplot: Average Intensity per Digit
Average Intensity
Digit Label
100
80
60
40
20
0
1
2
3
4
5
6
7
8
9

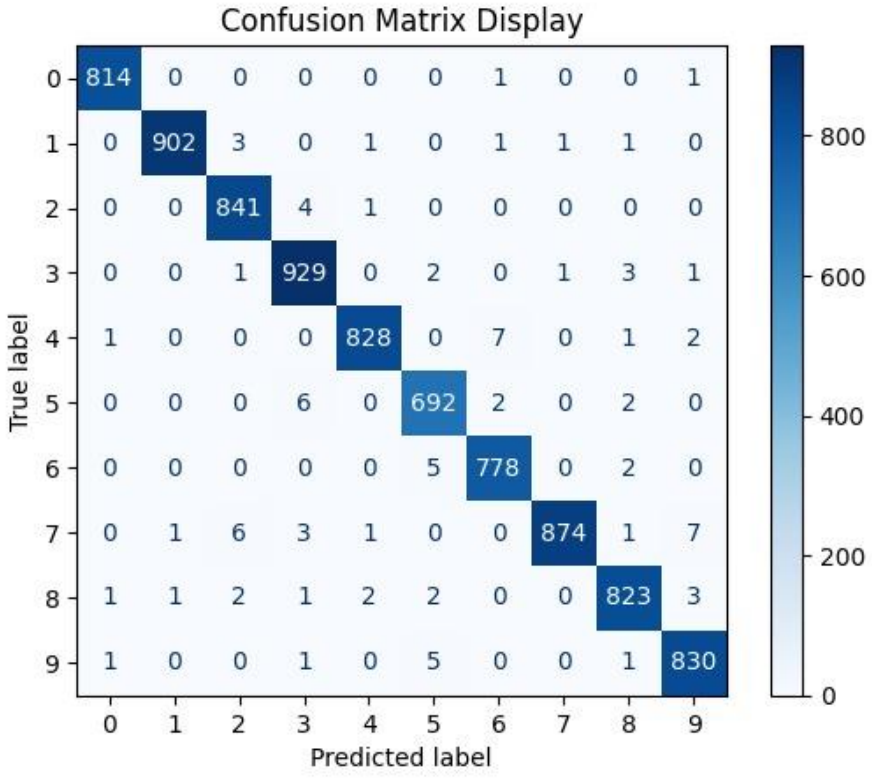

Number of Samples per Digit Class
Count
Digit Label
4000
3000
2000
1000
0
0
1
2
3
4
5
6
7
8
9

Input:

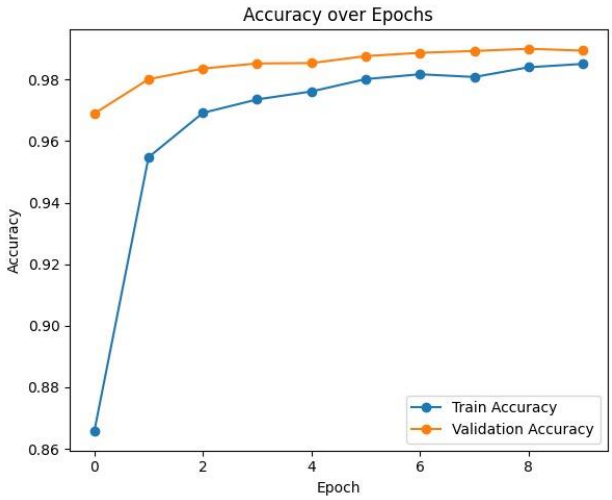


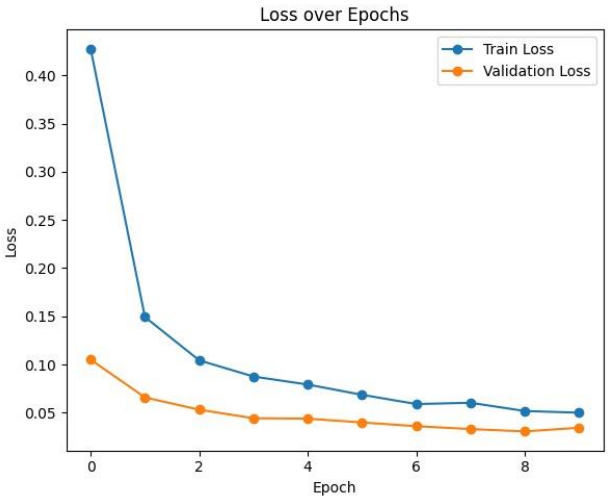
Outputs:

Confusion Matrix:



Loss and Accuracy Curve:



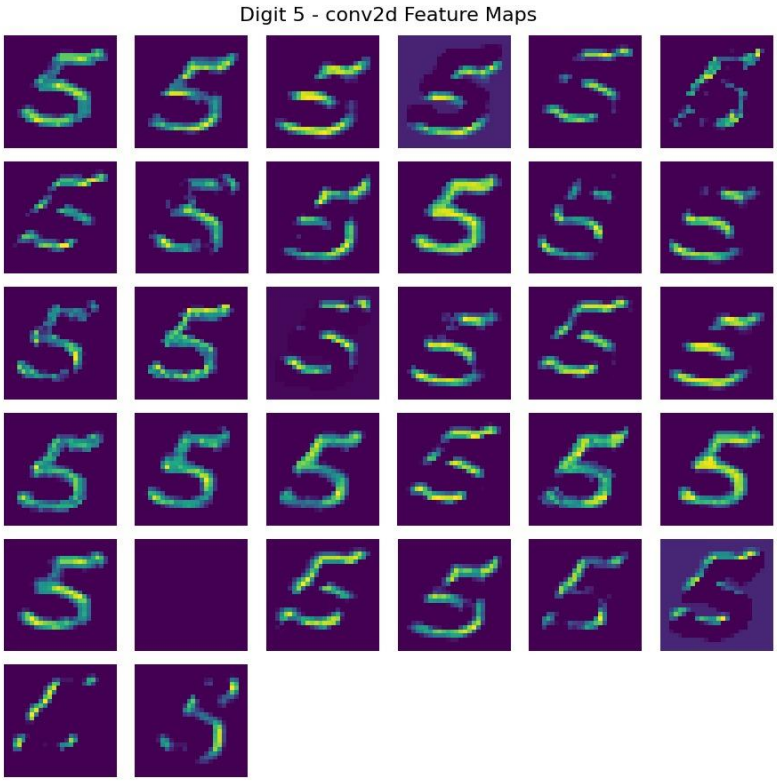


Model Parameters:

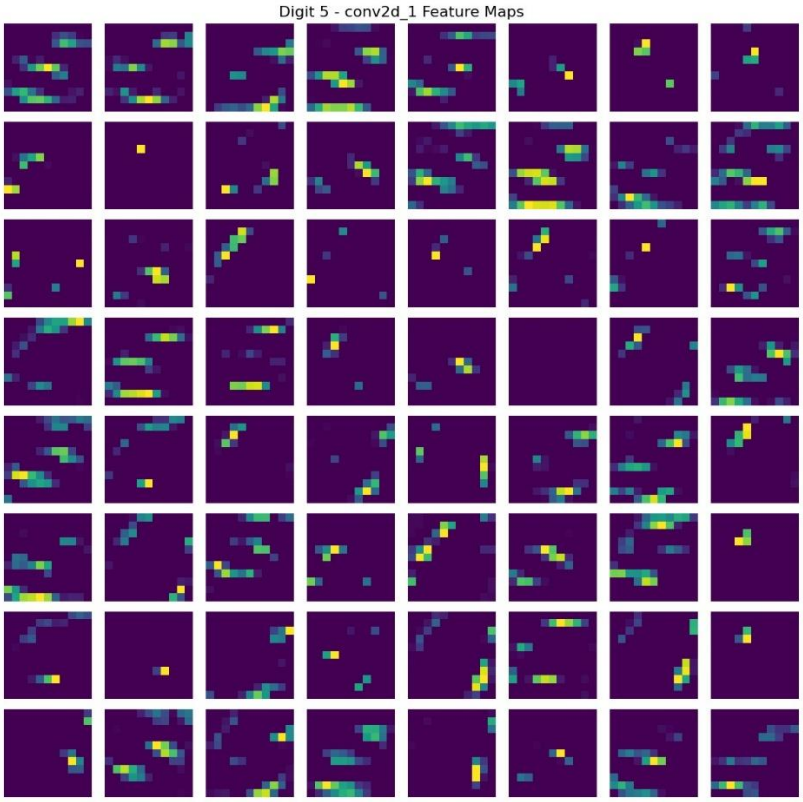
Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
dropout (Dropout)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
dropout_1 (Dropout)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 128)	204,928
dropout_2 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1,290

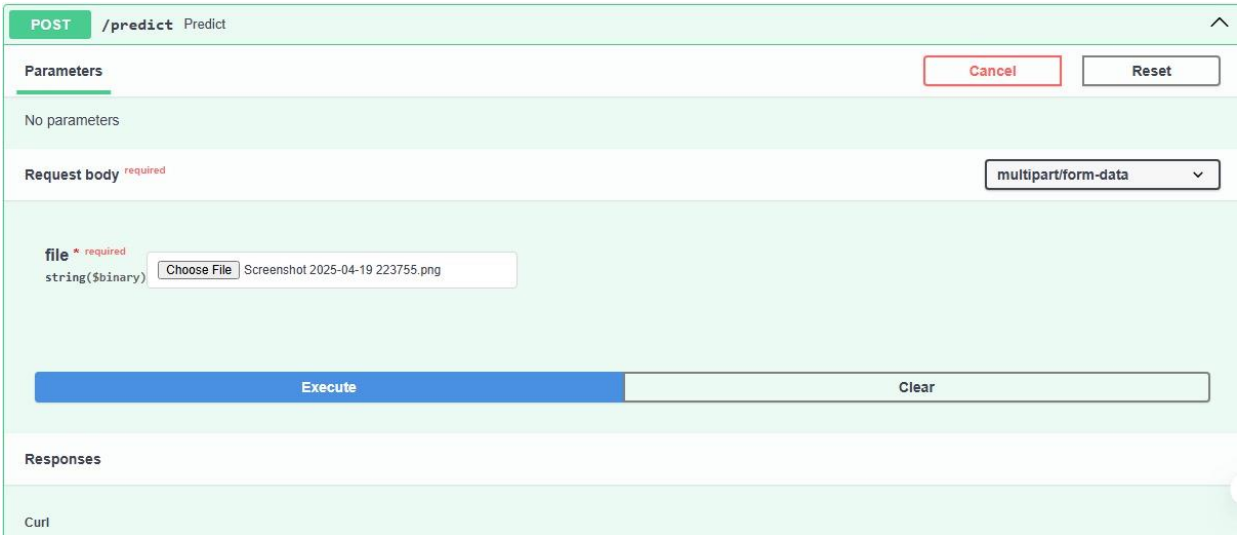
Convolution Layer 1:



Convolution Layer 2:



User Interface:



Input and Output:

Challenges & Risks

- Computational Limitations: Training deep networks requires high GPU resources.
- Overfitting: Managed using dropout and data augmentation.

Conclusion

The Convolutional Neural Network (CNN) model developed for digit classification demonstrates significant advantages over traditional machine learning models, particularly for image recognition tasks. Unlike conventional models such as Support Vector Machines, which rely heavily on manually engineered features, CNNs automatically extract and learn hierarchical spatial features directly from the raw image data. This capability allows CNNs to achieve superior performance and scalability for complex datasets like MNIST.

Through this project, the CNN model achieved high accuracy, leveraging techniques such as convolutional layers for feature extraction which are specifically designed for image processing. These advantages make CNNs better suited for tasks involving spatial hierarchies and local dependencies, which are difficult for traditional models to capture effectively.

References

Agrawal, Ayush Kumar, et al. “A Robust Model for Handwritten Digit Recognition Using Machine and Deep Learning Technique.” *2021 2nd International Conference for Emerging Technology (INCET)*, 21 May 2021, pp. 1–4, https://doi.org/10.1109/incet51464.2021.9456118. Accessed 19 Apr. 2025.

S. Ali, Z. Sakhawat, T. Mahmood, M. S. Aslam, Z. Shaukat and S. Sahiba, "A robust CNN model for handwritten digits recognition and classification," 2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications(AEECA), Dalian, China, 2020, pp. 261-265

Appendix

Code:

MODEL BUILDING:

1. Import Libraries:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

2. Data Preprocessing

```
# Normalize and reshape data

X = X / 255.0

X = X.reshape(-1, 28, 28, 1)

# One-hot encode labels

from tensorflow.keras.utils import to_categorical

y = to_categorical(y, num_classes=10)
```

3. Build CNN Model

```
model = Sequential([

    Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),

    MaxPooling2D(2,2),
```

```
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Dropout(0.25),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])
```

4. Compile and Train

```
model.compile(optimizer='adam',loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=10, batch_size=64, validation_split=0.2)
```

5. Evaluate Model

```
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy:.2f}")
```

6. Plot Accuracy and Loss

```
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.title('Model Accuracy')
plt.legend()
plt.show()
```

UI BUILIDING:

1. Loading the Model:

```
final_model = tf.keras.models.load_model(r"C:\Users\shriy\Downloads\CNN try 2\CNN try 2\model.h5")
```

2. Formatting the Image:

```
def format_image(img):
    img = img.resize((28, 28))                # Resize to 28x28
    img_array = np.array(img)                 # Convert to NumPy array
    return img_array
```

3. Image Prediction:

```
def predict_digit(model, img_array: np.ndarray) -> str:
    data = img_array.reshape(1, 28, 28, 1).astype('float32') / 255.0  # Reshape + normalize
    prediction = model.predict(data)                                  # Get prediction
    digit = np.argmax(prediction)                                     # Highest confidence class
    return str(digit)
```

4. FastAPI Endpoint for Prediction:

```
@app.post("/predict")
async def predict(file: UploadFile = File(...)):
    contents = await file.read()                                  # Read uploaded image file
    img = Image.open(io.BytesIO(contents)).convert('L')           # Convert to grayscale
    img_array = format_image(img)                                 # Format image
    digit = predict_digit(final_model, img_array)                 # Predict digit
```

```
    return {"Digit": digit}                    # Return response
```

5. Root Endpoint:

```
@app.get("/")
def read_root():
    return {"message": "MNIST digit recognition API is live!"}
```