HANDWRITTEN DIGIT RECOGNITION USING PYTHON

P.V.NAGESWARA RAO (622237)

ABSTRACT

- This project explores handwritten recognition using PYTHON, focusing on image processing techniques. The goal is to develop a system capable of accurately identifying handwritten digits and characters. Leveraging datasets like MNIST, alongside techniques such CNNs, high accuracy in recognizing characters is achieved. Through iterative refinement, the project demonstrates the potential of image processing and machine learning in handwritten recognition tasks . Additionally, rigorous evaluation measures are employed to assess the performance of the recognition system, ensuring robustness and reliability. This project underscores the significance of inter disciplinary approaches in addressing complex real-world challenges.

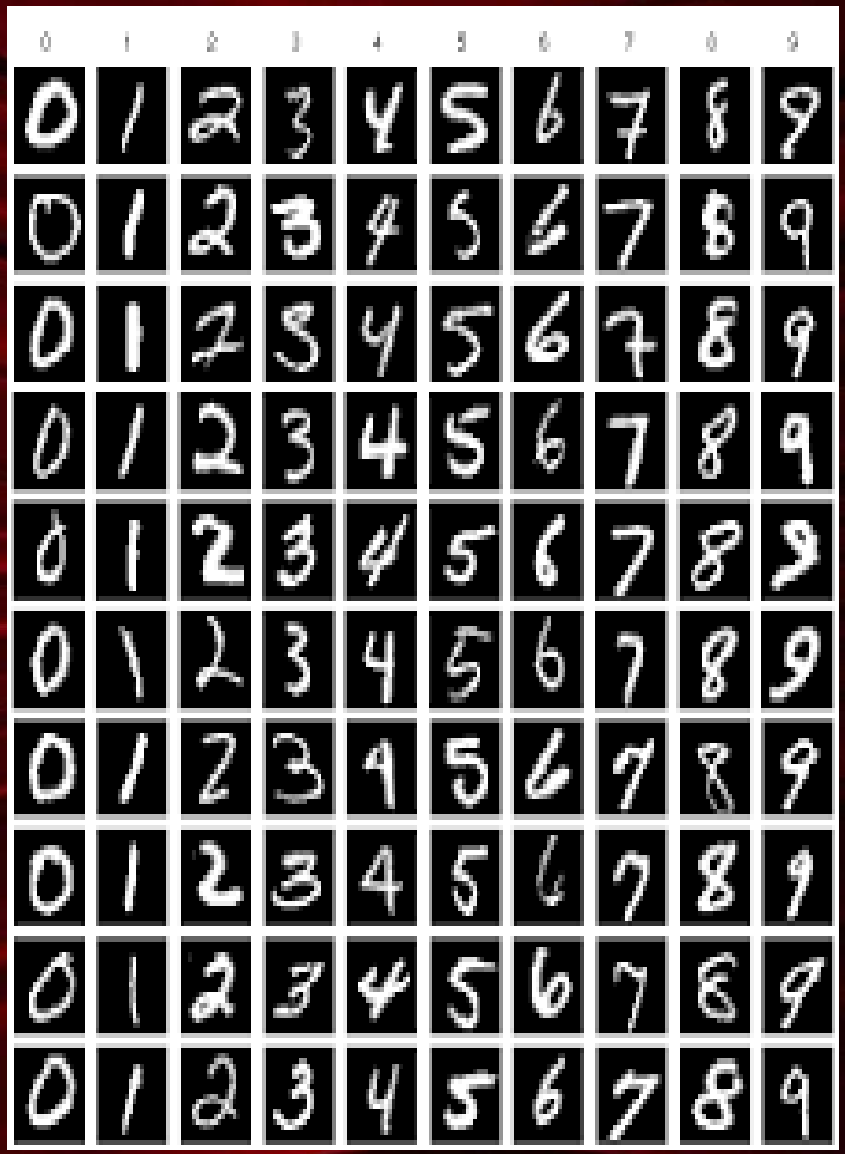
INTRODUCTION

In the realm of digital image processing and machine learning, the recognition of handwritten digits stands as a benchmark challenge that has garnered significant attention. Our project aims to address this challenge by employing a Convolutional Neural Network (CNN), a class of deep neural networks renowned for their prowess in analyzing visual imagery. Utilizing a Convolutional Neural Network (CNN), our project aims to accurately classify handwritten digits from the MNIST dataset.

In addition to processing this pre-defined dataset, our model extends its capabilities to recognize digits drawn in Microsoft Paint. This feature simulates a real-world scenario where the model interprets and classifies user-generated content, thereby demonstrating its adaptability and potential for practical applications. Through rigorous training and optimization, our CNN model endeavors to achieve high accuracy in digit recognition, paving the way for further research and development.

DATASET

- The MNIST dataset is a widely used benchmark dataset for handwritten digit recognition tasks. It consists of 28 x 28 pixel grayscale images of handwritten digits (0 - 9) collected f rom various sources.
- The dataset contains 60 , 000 training images and 10 ,000 test images, making it a standard dataset for training and evaluating machine learning models, especially neural networks.
- Due to i ts simplicity and accessibility, the MNIST dataset has become a benchmark for evaluating the performance of new machine learning algorithms and neural network architectures.



METHODOLOGY

1. Data Collection and Preprocessing:

- Data Collection:
  - Gather a dataset of handwritten digits. Popular choices include the MNIST dataset (containing 28x28 grayscale images of digits) or custom datasets.
  - Ensure a balanced distribution of digits (0 to 9) to prevent bias.
- Data Preprocessing:
  - Normalize pixel values to the range [0, 1].
  - Split the dataset into training, validation, and test sets

2. MODELSELECTION AND ARCHITECTURE DESIGN:

CNN Architecture:

- Design a CNN architecture suitable for image classification

  Common layers include : Convolutional layers (with filters/kernels)

  Pooling layers (e.g., max-pooling)

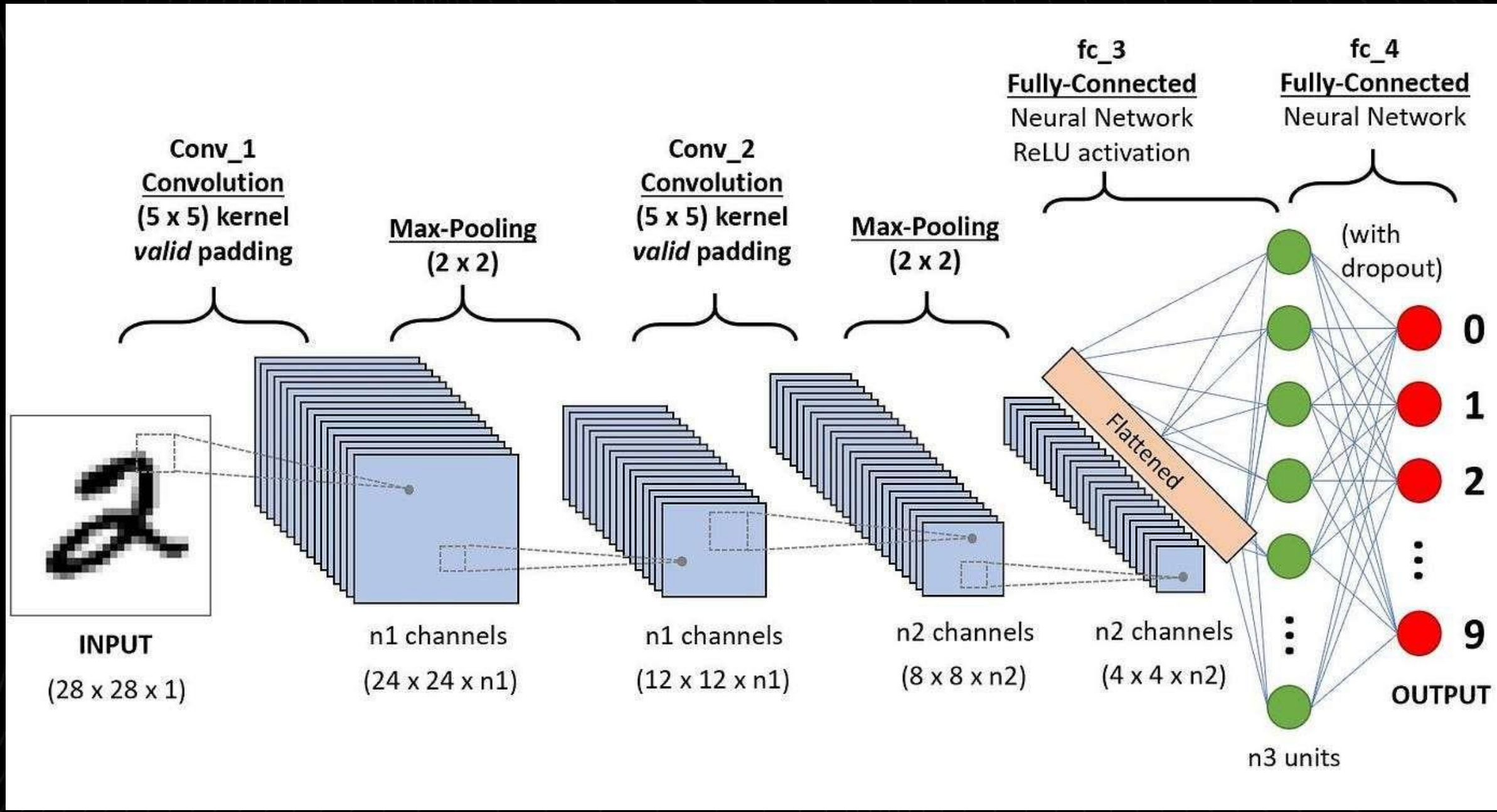
  Fully connected (dense) layers

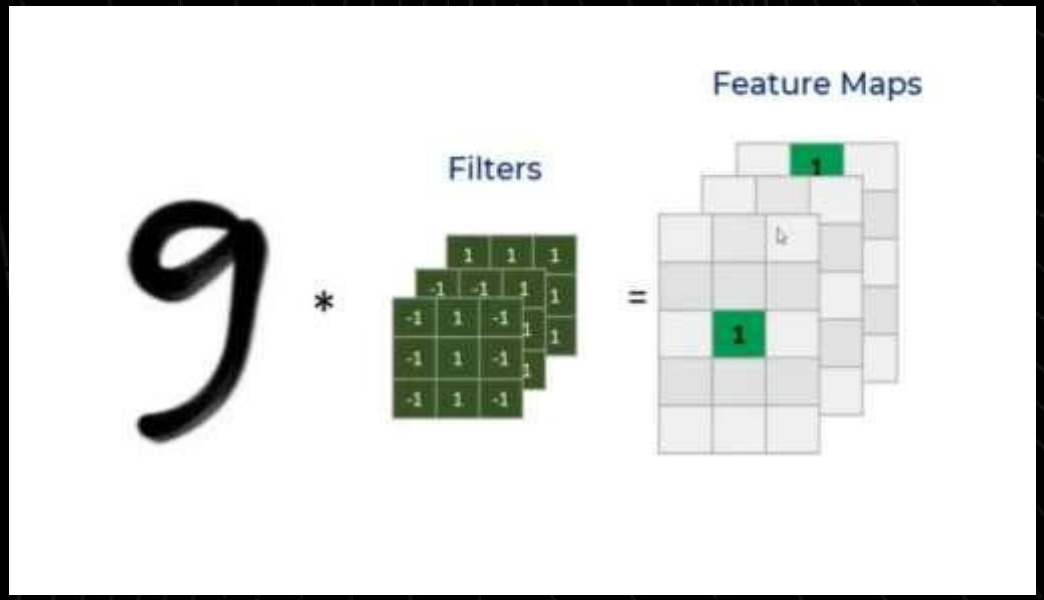
  Dropout layers (to prevent overfitting).

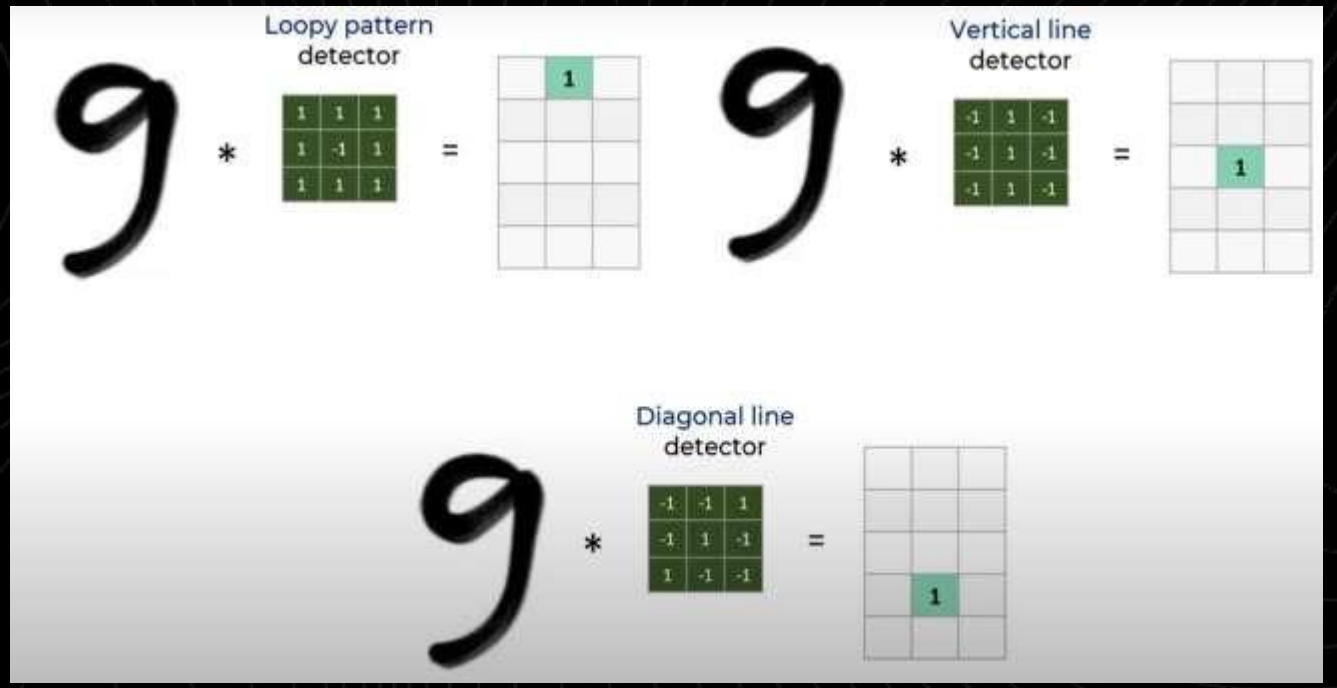
- Hyperparameters :

Choose hyperparameters such as learning rate, batch size, and optimizer (e.g., Adam).

Experiment with different architectures (e.g., LeNet , VGG, ResNet) to find the best fit.


Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with dropout)
Flattened
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT


Feature Maps
Filters
1
1
1
1
1
1
-1
-1
1
1
-1
1
-1
1
-1
1
-1
1
-1
1
-1
1
1


Loopy pattern detector
1
1
1
1
-1
1
1
1
1
1
Vertical line detector
-1
1
-1
-1
1
-1
-1
1
-1
1
Diagonal line detector
-1
-1
1
-1
1
-1
1
-1
-1
1

3. TRAINING:

COMPILE THE MODEL:

- Define the loss function (e.g., categorical cross-entropy)
- Compile the model with the chosen optimizer.

TRAIN THE MODEL:

- Feed training data into the model.
- Monitor training loss and accuracy
- Use early stopping to prevent overfitting.
- Save the best model checkpoint.

4. EVALUATION:

VALIDATION SET:

- Evaluate the model on the validation set. Calculate metrics (e.g., accuracy, precision, recall, F1-score).

TEST SET:

- Assess the model's performance on unseen data (test set).
- Generate a confusion matrix to analyze class-wise performance.

5.Fine-Tuning and Optimization:

Hyperparameter Tuning: Fine-tune hyperparameters (e.g., learning rate, dropout rate) using techniques like grid search or random search.

Regularization Techniques: Apply L2 regularization to prevent overfitting. Experiment with different dropout rates.

Learning Rate Scheduling: Implement learning rate schedules (e.g., reduce-on-plateau) to adaptively adjust learning rates during training.

CONTRIBUTIONS

P.V.NAGESWARA RAO: Contributed by extensively **working on** data acquisition, preprocessing, and model development

T.BHUVANESHU: : Implemented the code, contributed to testing, and code optimization.

S.M.SATYA SANDEEP: handled cross-validation and tutorials

SHAIK RAFI: Contributed to documentation

We shared our experience and knowledge gained from respective contributions among ourselves throughout the project.

Together, these contributions formed a cohesive and multidimensional approach to the Implementation of **HANDWRITTEN DIGIT RECOGNITION**

THANK YOU