HANDWRITTEN DIGIT RECOGNITION WITH LENET5 MODEL IN PYTORCH

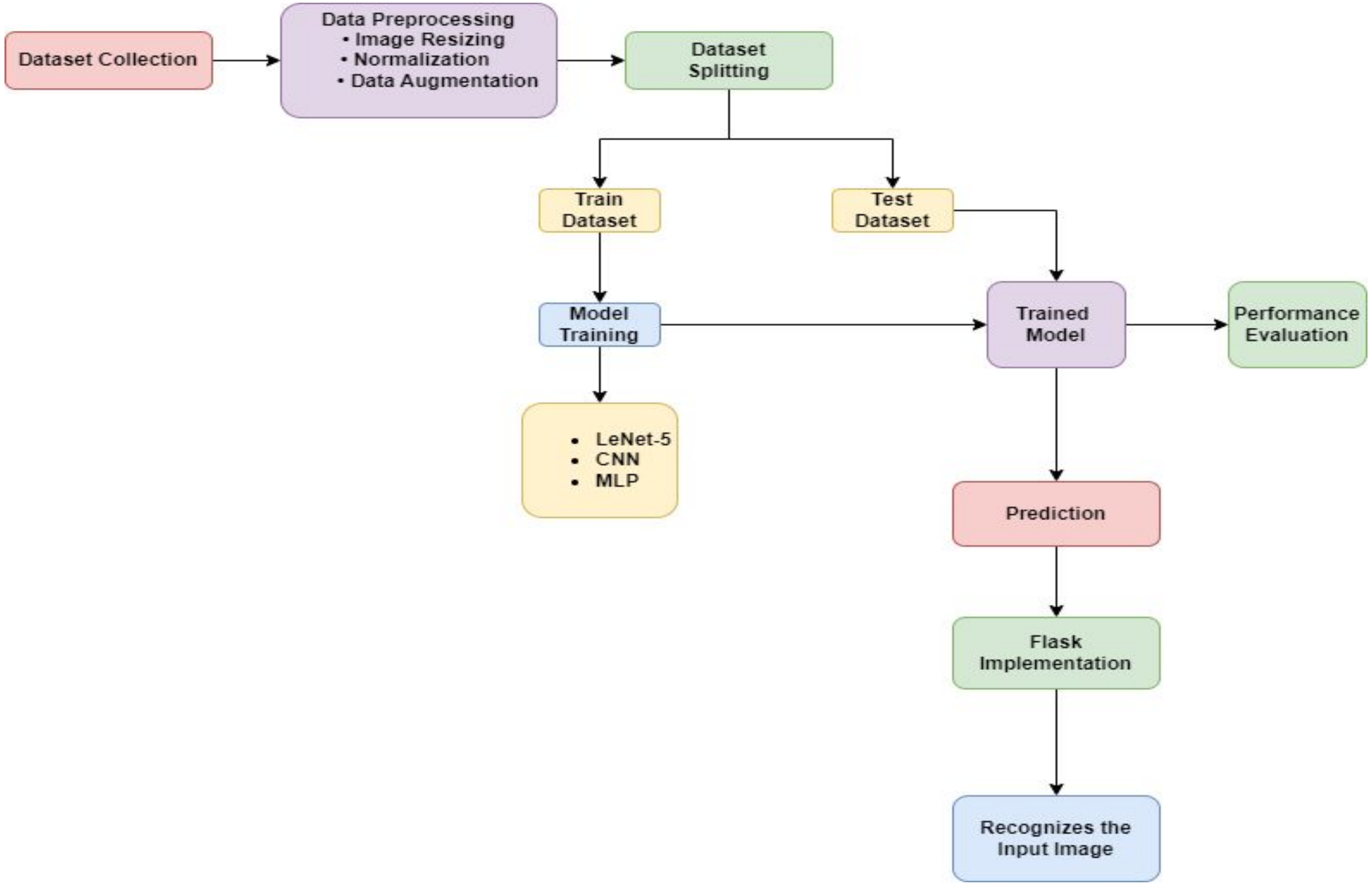
GROUP MEMBERS (GROUP 2)

★ Akshit Manke
★ Rohan Poddar
★ Avinash Naidu Damisetti
★ Ainy Khan
★ Sana Akhila

OBJECTIVE

The primary objective of this project is to evaluate and compare the effectiveness of LeNet-5, MLP, and CNN models in accurately classifying handwritten digits from the MNIST dataset. By implementing these models, the project aims to analyze their performance metrics such as accuracy, precision, recall, and F1-score. Additionally, the project aims to provide insights into the strengths and weaknesses of each model, identify the best-performing architecture for this specific task, and explore potential applications in digit recognition tasks.

PROPOSED APPROACH



Workflow of the Project

DATASET COLLECTION

MNIST Handwritten Digit Dataset:

- Contains 10,000 grayscale images of digits (0-9).
- Each image is 28x28 pixels.
- Divided into 7,000 training images and 3,000 testing images.

DATA PREPROCESSING

1) Image Resizing :

- Reshape data to fit input dimensions required by the LeNet-5 model (28*28 pixels)

2) Data Augmentation:

- Apply techniques to increase the diversity of the training data (e.g., rotations, shifts, flips).

3) Data Sampling:

- Sample the dataset to balance classes or reduce size for quicker iterations.

DATASET SPLITTING

- Dataset is divided into:

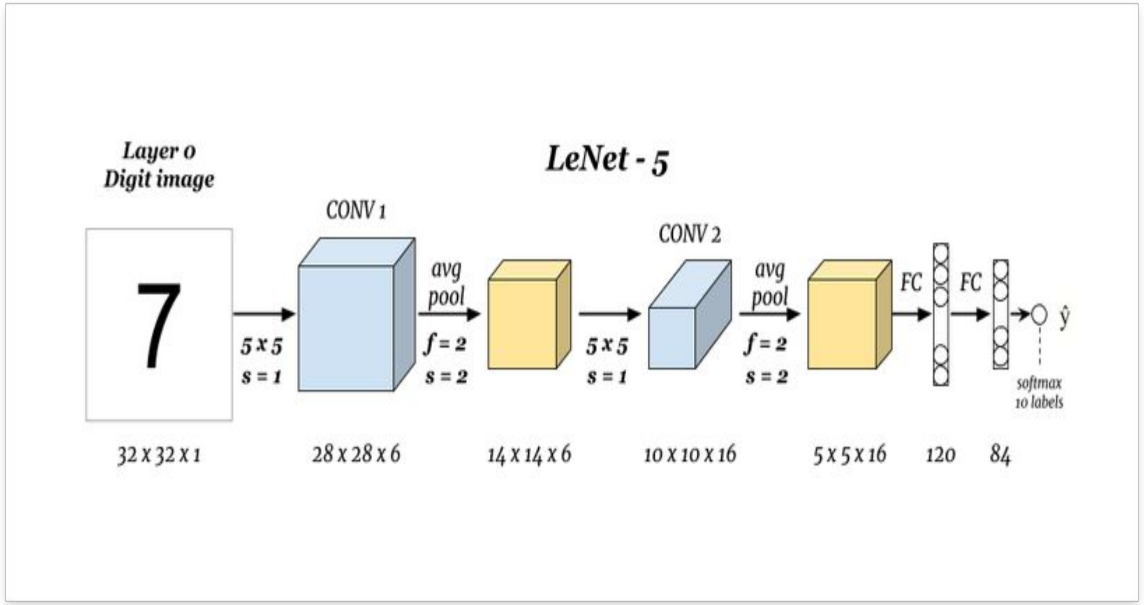
  Train Dataset: It is used to train the model to make accurate predictions

  Test Dataset: It is used to evaluate how accurately the trained model can predict

- Dataset is divided into a 7:3 ratio for training and testing purpose

LeNet-5 Architecture

- LeNet-5 is a the first convolutional neural network (CNN) architecture, developed by Yann LeCun .
- It has 7 layers:
  - ➢ Input layer (32x32 grayscale image)
  - ➢ Two convolutional layers
  - ➢ Two average pooling layers
  - ➢ Fully connected layers
  - ➢ Output layer



- It is used in handwritten digit recognition as it is simple and easy to understand.

MODEL TRAINING

- Model is trained using the LeNet-5
- Utilized 10 epochs with a batch size of 32 to progressively optimize the LeNet-5 model's parameters
- During training used Adam optimizer for adaptive learning rates to enhance the training efficiency
- To prevent overfitting, dropout regularization was implemented
- Batch Normalization was used to stabilize and accelerate training, ensuring consistent model performance across mini-batches

PERFORMANCE EVALUATION

- Model is evaluated using accuracy, precision and recall metrics
- Achieved an accuracy of 96%
- Leveraged CUDA operations for GPU acceleration, enhancing computational performance
- The model is capable to predict the digits from 0-9

Multi -Layer Perceptron (MLP)

The architecture of a Multilayer Perceptron (MLP) consists of an input layer, one or more hidden layers, and an output layer.

1. Input Layer - Receives the input data.

2. Hidden Layers - Transform input data into a format that the output layer can use.

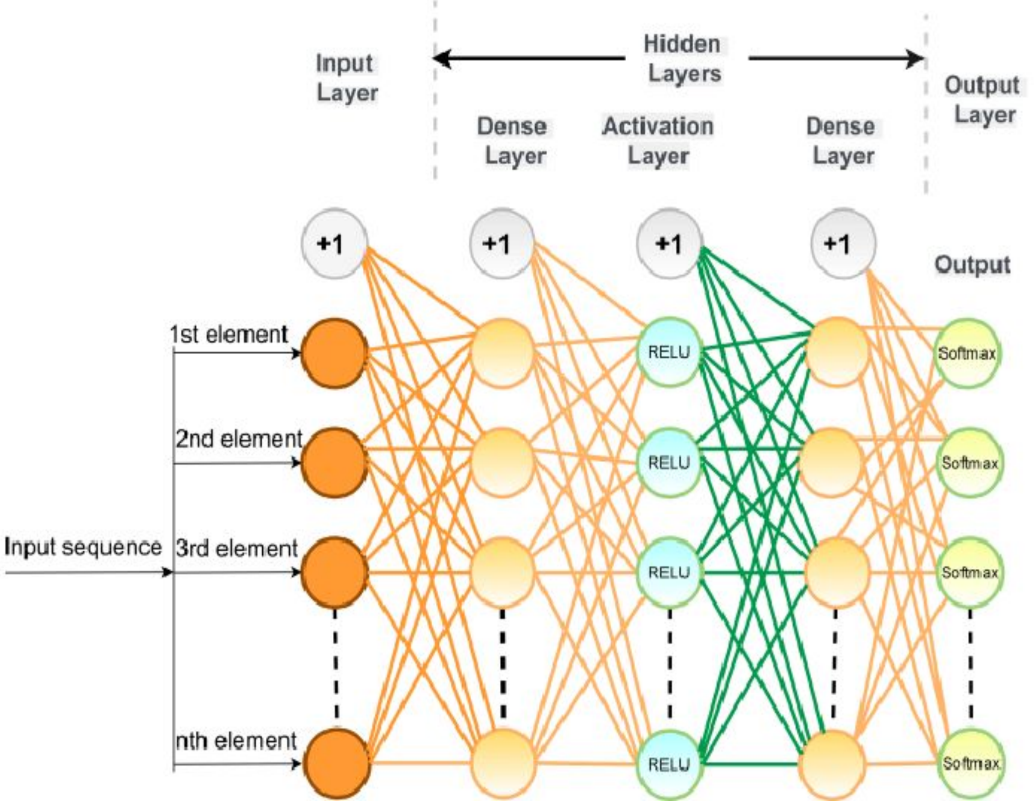
3. Output Layer - Produces the final output.

Activation Function: Use Relu for multi-class classification,

or sigmoid for binary classification.

4. Training Process

- **Forward Propagation:** Inputs are passed through the network layer by layer
- **Loss Function:** difference between the predicted output and the actual target.
- **Backpropagation:** Computes the gradient of the loss function with respect to each weight and bias using the chain rule.
- **Optimization Algorithm:** Updates the weights and biases to minimize the loss function.



MODEL TRAINING

- Model is trained using the MLP.
- Utilized epochs with a batch size of 32 to progressively optimize the MLP model's parameters
- During training used Adam optimizer for adaptive learning rates to enhance the training efficiency
- To prevent overfitting, dropout regularization was implemented
- Batch Normalization was used to stabilize and accelerate training, ensuring consistent model performance across mini-batches

PERFORMANCE EVALUATION

- Model is evaluated using accuracy, precision and recall metrics
- Achieved an accuracy of 97%
- Utilized CUDA operations to accelerate computations on the GPU, significantly improving performance.
- Model is capable to predict the digits from 0-9

Convolutional Neural Networks(CNN)

A Convolutional Neural Network (CNN) is a deep learning architecture commonly used for image and video recognition tasks.

1. Input Layer - Accepts the raw input image data.

2. Convolutional Layer - Applies convolutional filters to the input data to extract features such as edges, textures, and patterns.

3.Filters (Kernels): matrices that slide over the input data to compute dot products

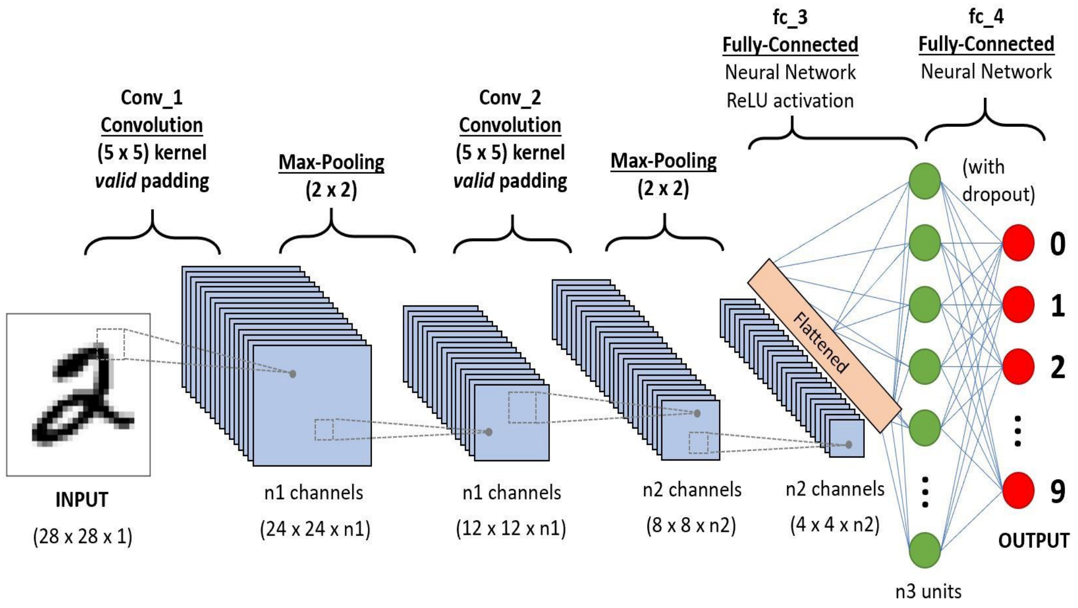
4. Stride: Determines the step size for the filter's movement across the image.

5. Padding: Adds extra pixels around the input image to control the spatial dimensions .

6. Activation Functions: ReLU.

7. Pooling Layer - Reduces the spatial dimensions (height and width) of the feature maps to reduce computation and control overfitting.

8. Fully Connected (Dense) Layer - Flattens the input and processes it through fully connected layers.



MODEL TRAINING

- The model was trained using a custom CNN.
- Training was done over multiple epochs with a batch size of 32 to gradually optimize the model's parameters.
- The Adam optimizer was used for adaptive learning rates to improve training efficiency.
- Dropout regularization was applied to prevent overfitting.
- Batch normalization was used to stabilize and speed up training, ensuring consistent performance across mini-batches.

PERFORMANCE EVALUATION

- The model's performance was evaluated using accuracy, precision, and recall metrics.
- It achieved an accuracy of 97%.
- CUDA operations were used to speed up computations on the GPU, greatly enhancing performance.
- The model can accurately predict digits from 0 to 9.

FLASK APPLICATION PROCEDURE

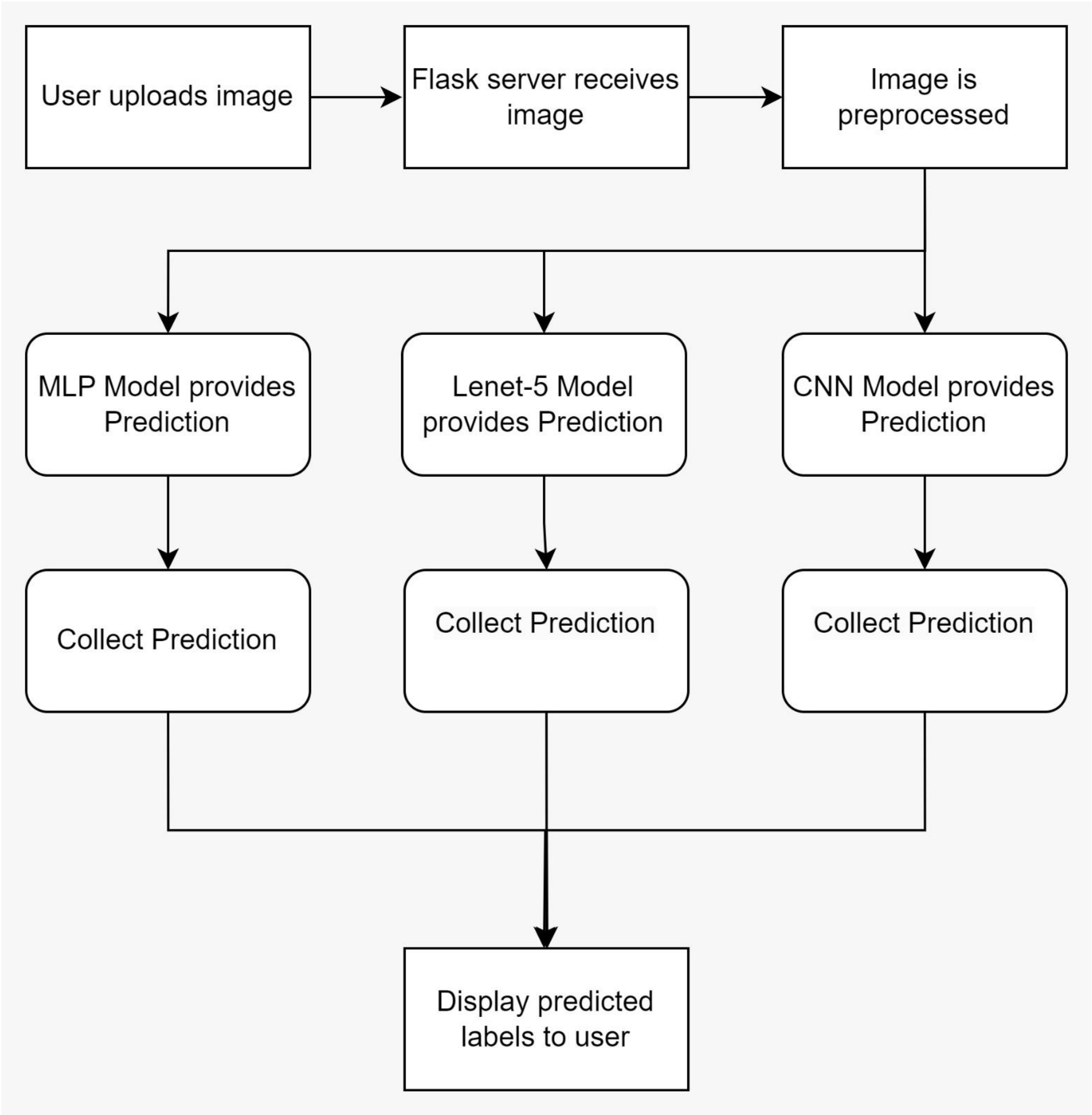
1. User Uploads Image: User uploads an image through the web interface.

2. Image Received by Flask Server: The Flask server receives the uploaded image.

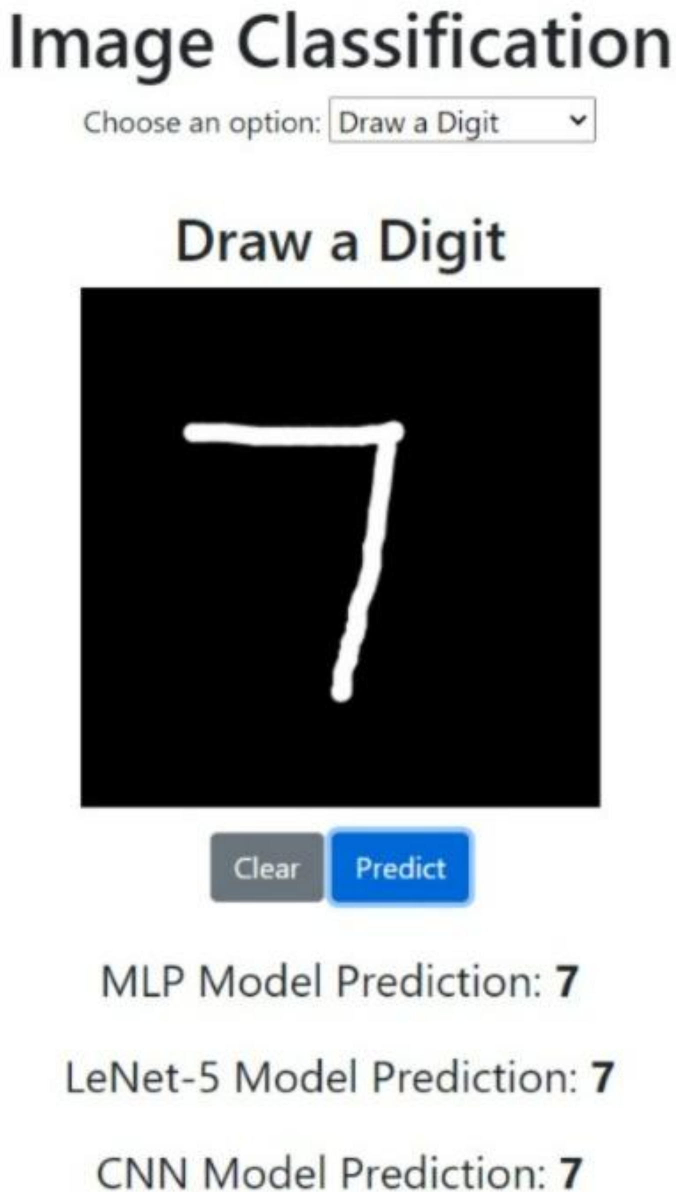
3. Image Preprocessing: The server preprocesses the image if necessary (e.g., resizing, normalization).

4. Model Predictions: The pre-processed image is passed to three different models for prediction and they provide individual predictions for the image.

5. Result Displayed: The predicted label from each model is displayed to the user.



FLASK APPLICATION INTERFACE



Deployment of Flask Project using Docker Hub

1. Create Dockerfile:

- Base Image: python:3.8-slim
- Copy application code
- Install dependencies: RUN pip install -r requirements.txt
- Set working directory: WORKDIR /app
- Expose port: EXPOSE 5000
- Run application: CMD ["python", "app.py"]

2. Build Docker Image:

- Command: docker build -t my-flask-app .

3. Test Locally:

- Command: docker run -p 5000:5000 my-flask-app
- Verify by visiting http://localhost:5000

4. Push to Docker Hub:

- Login: docker login
- Tag Image: docker tag my-flask-app username/my-flask-app:latest
- Push: docker push username/my-flask-app:latest

5. Deploy from Docker Hub:

- Pull Image: docker pull username/my-flask-app:latest
- Run: docker run -p 5000:5000 username/my-flask-app:latest

KEY TAKEAWAYS

- MLP provides baseline performance but lacks ability to capture spatial features.
- LeNet-5 and CNN effectively leverage convolutional layers for higher accuracy.
- CNN demonstrates superior performance in extracting and learning complex features.

FUTURISTIC PROJECTS BASED ON THESE MODELS

1. Character Recognition (OCR)

Extend the handwritten digit recognition to recognize handwritten letters and symbols.

Implement a complete Optical Character Recognition (OCR) system for scanned documents.

2. Image Classification with Advanced Architectures

Explore and implement more advanced neural network architectures such as VGG, ResNet, and Inception for image classification tasks.

3. Object Detection

Implement object detection models like YOLO (You Only Look Once) or SSD (Single Shot Multibox Detector) to detect and localize objects within images.

THANK YOU