**Project Summary: MNIST Digit Recognition using Deep Learning**

**1. Introduction**

This project focuses on building a deep learning model to recognize handwritten digits (0-9) from the MNIST dataset. The model is a Convolutional Neural Network (CNN) trained to classify digit images with high accuracy. It is deployed using Flask and hosted on Render to provide a web-based interface for users to upload an image and receive a predicted digit.

**2. Dataset Used**

* **MNIST Dataset:** A collection of 70,000 grayscale images of handwritten digits (28x28 pixels each).
* **Training Set:** 60,000 images
* **Testing Set:** 10,000 images
* Each image is labeled (0-9) and used for supervised learning.

**3. Model Architecture**

The model is implemented using TensorFlow and Keras and follows a CNN-based architecture:

model = models.Sequential([

layers.Conv2D(32, (3, 3), activation="relu", input\_shape=(28, 28, 1)),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation="relu"),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(128, (3, 3), activation="relu"),

layers.Flatten(),

layers.Dense(128, activation="relu"),

layers.Dropout(0.2),

layers.Dense(10, activation="softmax")

])

**Explanation of Layers:**

1. **Conv2D (32 filters, 3x3 kernel, ReLU activation):** Extracts low-level features like edges and textures.
2. **MaxPooling2D (2x2):** Reduces spatial dimensions, making the model efficient.
3. **Conv2D (64 filters, 3x3 kernel, ReLU activation):** Learns deeper features.
4. **MaxPooling2D (2x2):** Further down-sampling.
5. **Conv2D (128 filters, 3x3 kernel, ReLU activation):** Extracts high-level features.
6. **Flatten():** Converts feature maps into a single vector.
7. **Dense (128 neurons, ReLU activation):** Fully connected layer for feature learning.
8. **Dropout (0.2):** Prevents overfitting by randomly disabling neurons.
9. **Dense (10 neurons, Softmax activation):** Outputs probabilities for each digit (0-9).

**4. Model Training**

* **Optimizer:** Adam (Adaptive Moment Estimation)
* **Loss Function:** Categorical Crossentropy (since it's a multi-class classification problem)
* **Batch Size:** 32
* **Epochs:** 10-20 (tuned for optimal accuracy)
* **Evaluation Metric:** Accuracy
* The model achieves a high accuracy (~98%) on the test dataset.

**5. Web Deployment Using Flask**

* A Flask-based web app is created to allow users to upload a digit image.
* The image is preprocessed (grayscale conversion, resizing, normalization) before being fed into the model.
* The predicted digit is displayed on the web interface.
* Hosted on **Render** for easy access.

**6. Challenges and Solutions**

* **Deployment Issues:** Fixed missing template errors and incorrect start commands in Render.
* **Model Performance:** Applied dropout layers to reduce overfitting.
* **Input Image Handling:** Used PIL and NumPy for proper image preprocessing.

**7. Key Learnings**

✅ Implemented a deep learning model from scratch. ✅ Learned about CNN layers and their roles in feature extraction. ✅ Gained experience in deploying an ML model using Flask and Render. ✅ Debugged deployment errors and optimized the model for performance.

**8. Conclusion**

This project successfully demonstrates how deep learning can be applied to image classification tasks. The CNN model effectively classifies handwritten digits with high accuracy. The deployment allows users to interact with the model in real time, making it a practical machine learning application.

**9. Future Enhancements**

* Improve accuracy by fine-tuning hyperparameters.
* Deploy using **FastAPI** for better performance.
* Integrate with **React.js** for a more interactive UI.
* Train on more diverse handwritten datasets for generalization.