🧠 Convolutional Neural Network (CNN) Mini Project: MNIST Digit Classification

Project Overview

This project implements a Convolutional Neural Network (CNN) using TensorFlow/Keras to accurately classify handwritten digits (0-9) from the popular MNIST dataset. This serves as a foundational "Hello World" deep learning project, demonstrating key concepts of image recognition, data preprocessing, model architecture design, and performance evaluation.

Detail	Value
Problem Type	Multi-class Classification
Dataset	MNIST (Modified National Standards and Technology)
Model	Sequential CNN
Framework	TensorFlow 2.x / Keras
Typical Accuracy	$98.5\% - 99.4\%$

🚀 How to Run the Project (Google Colab)

This project is designed to be executed step-by-step in a **Google Colaboratory** environment, which provides all necessary dependencies (TensorFlow, NumPy, Matplotlib) and free GPU acceleration.

1. **Create a New Colab Notebook:** Go to Google Colab and start a new notebook.
2. **Paste Code:** Copy the content of the primary notebook file (`MNIST_CNN_Project.md`) and paste it into the first code cell.
3. **Set Runtime:** (Optional but recommended) Go to **Runtime** > **Change runtime type** and select **GPU** as the hardware accelerator.
4. **Execute Cells:** Run each cell sequentially (using `Shift + Enter` or the Run button) from Section 1 to Section 7.

⚙️ Key Steps and Components

1. Data Loading and Preprocessing

The MNIST dataset is loaded directly from Keras. Critical preprocessing steps include:

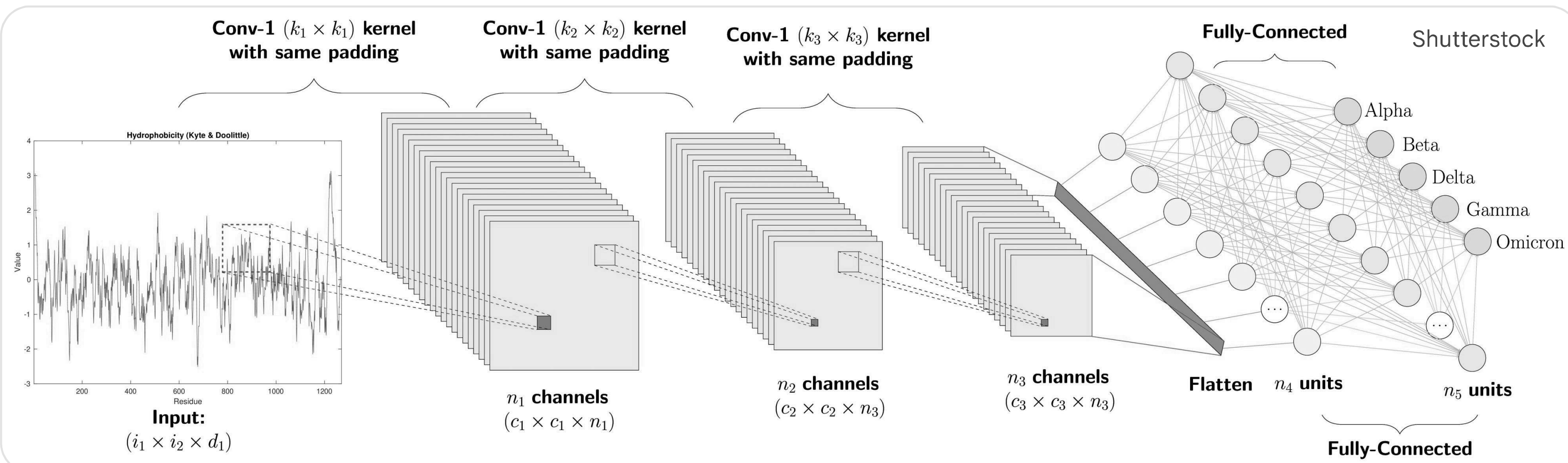
- **Normalization:** Scaling pixel values from the initial range of 0 to 255 down to 0.0 to 1.0. This helps the neural network converge faster.
- **Reshaping:** Adding a color channel dimension to transform the images from $(28, 28)$ to $(28, 28, 1)$, which is the expected input shape for a `Conv2D` layer.
- **One-Hot Encoding:** Converting integer labels (e.g., `5`) into categorical vectors (e.g., $[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]$) for use with the `categorical_crossentropy` loss function.

2. CNN Model Architecture

The model uses a simple Sequential API structure:

1. `Conv2D(32, (3, 3), activation='relu')`: Learns 32 distinct features from the images.

2. `MaxPooling2D((2, 2))`: Halves the spatial dimensions, reducing computational load and increasing robustness to slight shifts.
3. `Conv2D(64, (3, 3), activation='relu')`: Further feature extraction.
4. `MaxPooling2D((2, 2))`: Second spatial reduction.
5. `Flatten()`: Converts the 2D feature maps into a 1D vector.
6. `Dense(128, activation='relu')`: A hidden layer for high-level classification logic.
7. `Dense(10, activation='softmax')`: The output layer, yielding a probability distribution over the 10 classes (digits 0-9).



3. Training and Evaluation

The model is trained for 8 epochs using the **Adam optimizer** and **Categorical Crossentropy loss**. After training, the performance is evaluated on the dedicated test set.

📊 Results and Visualization

The project generates visualizations to assess the model's learning process:

- **Final Test Accuracy:** Reports the final classification accuracy on unseen data.
- **Training History Plots:** Two plots are generated using Matplotlib:
  - **Accuracy vs. Epoch:** Shows how training and validation accuracy changes over each training cycle.
  - **Loss vs. Epoch:** Shows how training and validation loss decreases over each training cycle.
- **Sample Predictions:** Displays a few test images along with the model's prediction, highlighting correct (green) and incorrect (red) classifications.

Expected Performance (Example)

Metric	Value
Test Loss	$\approx 0.03 - 0.06$
Test Accuracy	$\approx 99.00\%$