Machine Learning Mini-Project Report: Fashion-MNIST Classification

1. Problem Analysis & Objective

The objective of this project is to develop and compare machine learning models for the **Fashion-MNIST image classification task**. This is a **multi-class classification problem** with ten distinct categories of clothing items.

The goal is to create a robust model capable of accurately classifying **28×28 pixel grayscale images** and to demonstrate the model's performance through comparative analysis and a practical GUI deployment.

Feature	Details
Dataset	Fashion-MNIST (60,000 train + 10,000 test images)
Task	10-class image classification (T-shirt/top, Trouser, Pullover, etc.)
Success Metric	Test Accuracy ($\%$)
Innovation Goal	Deploying a robust model capable of handling non-standard custom images (via Data Augmentation).

---

2. Data Preprocessing (EDA)

Data Loading and Preparation

- The raw data was loaded from **CSV files** (fashion-mnist_train.csv and fashion-mnist_test.csv) containing 785 columns (label + 784 pixel values).
- The full training set (60,000 images) was split into a **Training set (90%)** and a **Validation set (10%)** for effective model monitoring.

Critical Preprocessing Steps

1. **Normalization:** All pixel feature values (originally 0-255) were scaled to the range **0.0 to 1.0** by dividing by 255.0. This is crucial for stabilizing gradient descent in neural networks.
2. **Reshaping:** Data intended for the CNN was reshaped from a flat vector (784) to a 28 * 28 *1tensor (Height* Width *Channels) as required by the Conv2D layers.
3. **One-Hot Encoding:** CNN labels (y) were converted to a **one-hot categorical format** (e.g., class 2 becomes [0, 0, 1, 0, ..]) to work with the softmax output and categorical_crossentropy loss function.

Data Augmentation (Innovation)

An ImageDataGenerator was configured for the CNN training to introduce small, random variations to the training data. This is an innovation designed to improve the model's **generalization** and make it **robust to slight shifts, rotations, and zooms** often found in real-world images.

Augmentation Parameter	Value
Rotation Range	8 degrees
Zoom Range	0.08(8%)
Shift Range	0.08 (8% for width and height)

---

3. Model Development & Hyperparameter Tuning

The project compared a classic machine learning model (Logistic Regression) with an advanced Deep Learning model (Deep CNN).

3.1. Algorithm Comparison and Performance

Model	Architecture	Hyperparameters	Test Accuracy
Logistic Regression	Linear Model (Baseline)	Solver='saga', max_iter=200	Typically= 85
Deep CNN	3 x Conv2D Blocks, 1 x Dense Layer	Dropout=0.5, epochs=20, batch_size=128	**Expected =92%+**

3.2. Deep CNN Architecture (Innovation)

The final model is a **Deep Convolutional Neural Network** built for high accuracy and robustness.

Layer	Type & Filters	Output Shape	Rationale
1	Conv2D(32) + ReLU	28 *28	Base feature extraction
2	MaxPooling2D	14 *14	Reduces dimensionality
3	Conv2D(64) + ReLU	14 *14	Deeper feature extraction
4	MaxPooling2D	7 * 7	Further reduction
5	Conv2D(128) + ReLU	7 * 7	Deepest feature map generation
6	Flatten	6,272	Prepares data for Fully Connected layers
7	Dense(128) + ReLU	128	High-level feature learning
8	Dropout(0.5)	128	**Regularization** to prevent overfitting
9	Dense(10) + Softmax	10	Final classification output

3.3. Hyperparameter Tuning & Optimization

- **Optimizer: Adam** was selected for its adaptive learning rate properties.

- **Regularization:** A **Dropout** rate of 0.5 was applied to the dense layer to significantly reduce overfitting, which is critical in deep networks.
- **Runtime Optimization: Early Stopping** with a patience of 5 was implemented, monitoring val_loss. This ensures that training halts once performance plateaus, guaranteeing that the **best-performing weights** (lowest validation loss) are used and saving computation time.

---

4. Final Deployment and GUI (Innovation)

The best-performing model, the **Deep CNN**, was saved as best_deep_fashion_classifier_final.h5 and deployed using a custom **In-Notebook GUI**.

Deployment Architecture

The GUI uses the following components:

- **ipywidgets**: Creates the interactive user interface (Upload button, Classify button, Output area).
- **PIL (Pillow)**: Used to open and preprocess the uploaded image.
- **Prediction Pipeline**:
    1. User uploads image.
    2. Image is converted to **grayscale** and resized to **28 *28 pixels**.
    3. Pixel values are **normalized** (0.0-1.0) and reshaped to the **1 * 28 * 28 * 1** tensor format.
    4. The best_model (CNN) predicts the class and confidence.
- **Output Enhancement**: The result displays the original **Uploaded Image**, the **Predicted Class**, and the prediction **Confidence (%)**, providing immediate, clear feedback to the user.