Evaluating MLP and CNN Models for Fashion-MNIST Classification

Group #10
Authors: Karan Arora (R:233562444) | Neelesh Batham (R:241562502) | Advait Patwardhan (R:241562512) | Tony Varghese (R:233562503)
Colab File Link: NB_EE954_Assingnment_Group10.ipynb
GitHub Link: https://github.com/neeleshbatham/mlp-cnn-fashionmnist/

Introduction

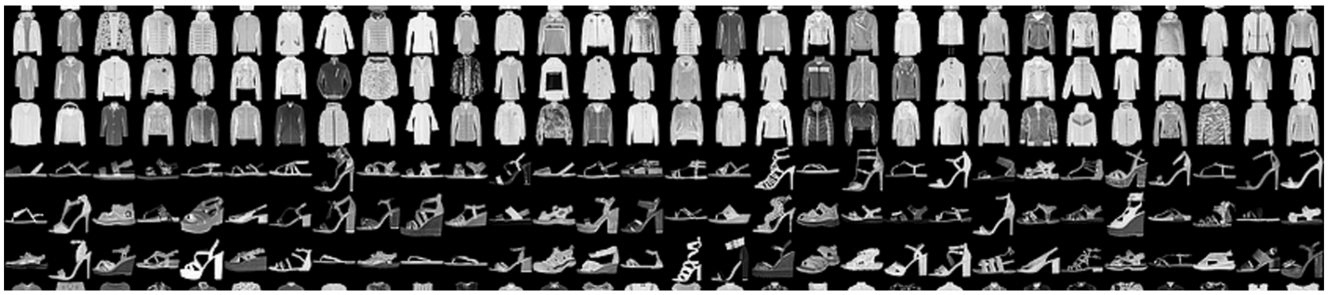
This report presents the process of building and evaluating two neural network architectures—a Multi-Layer Perceptron (MLP) and a Convolutional Neural Network (CNN)—to classify images in the Fashion-MNIST dataset. We explain the preprocessing steps, model architecture, training parameters, and evaluation metrics, comparing model performance based on different configurations.

Importing Libraries

Requirements: - Python 3.6+ -
Libraries: - torch , torchvision , numpy , matplotlib, seaborn
Hardware: - CPU or GPU (optional, for faster training)

Dataset and Preprocessing

- **Data Description**:
  - The FashionMNIST dataset is a dataset of 60,000 training+validation and 10,000 test grayscale images of 10 different types of clothing items, such as t-shirts, trousers, shoes, and dresses. Each image is 28x28 pixels in size.
  - Key Points about FashionMNIST:
    - Class Name: *torchvision.datasets.mnist.FashionMNIST*
    - Purpose: It provides easy access to the FashionMNIST dataset for use in training machine learning and deep learning models.
    - Data Type: The dataset consists of 28x28 pixel grayscale images.
    - Categories: The dataset has 10 categories representing various clothing items, such as T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag and Ankle boot



- **Preprocessing Techniques**:
    - **Splitting the Data:** To ensure proper validation, the dataset is split as follows:
        - Training Set: 80% of the dataset (~48K images)
        - Validation Set: 20% of the dataset (~12K images)
        - Test Set: Separate dataset for testing (~10K images)
    - **Normalization:** Each image's pixel values are scaled to a [0,1] range. This scaling, achieved by dividing each pixel by 255, helps stabilize and speed up the model training process.
    - **Data Loaders:** PyTorch's DataLoader is used to batch and shuffle the data, making it easier to process in mini-batches during training. Shuffling helps to prevent model bias by randomizing the sample order.

MLP Model (Multilayer Perceptron)

The MLP model is a simple neural network with multiple fully connected layers. The model architecture consists of:

Model Architecture and Implementation

- Image Flattening: Each 28x28 image is flattened into a 784-dimensional vector before feeding it into the model, as MLPs do not handle spatial information directly.
- Layers: The MLP architecture consists of three fully connected (dense) layers:
    - Input Laye**r**: 784 neurons.
    - Hidden Layers: Two hidden layers, one with 256 neurons and second with 128 neurons, each followed by ReLU activation and a dropout layer to prevent overfitting.
    - Output Layer: 10 neurons representing the 10 classes, with a softmax output.

Manual Forward and Backward Pass Implementation

As per the requirements, custom forward and backward passes were implemented without using inbuilt functions. The backward pass used gradient calculations for each layer's weights and biases, iterating through the layers to compute loss and update gradients manually.

Cross Entropy Loss Calculation

A custom cross-entropy loss function was implemented to measure the discrepancy between predicted and actual labels. The cross-entropy loss calculates the negative log-probability of the correct class, penalizing incorrect predictions.

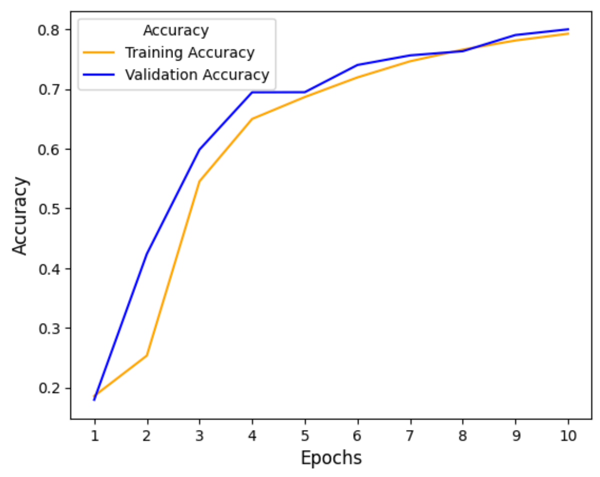
Hyperparameter Tuning

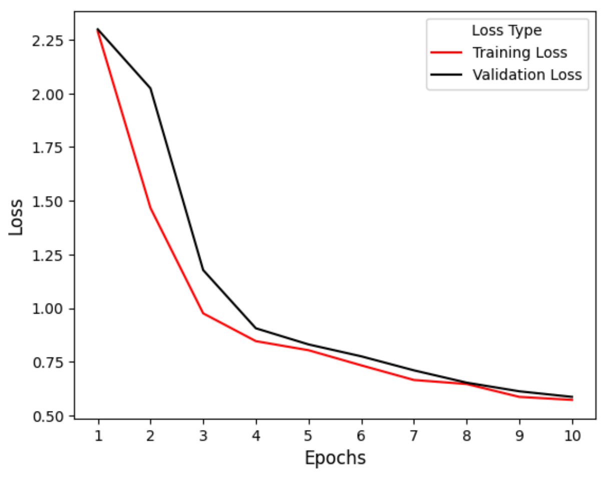
Different combinations of layers, dropout rates, and learning rates were explored:

- Number of Layers: Increased hidden layers from 1 to 2 improved model capacity without overfitting.
- Learning Rate: Initial experiments used learning rates from 0.01 to 0.001, with 0.01 yielding the best results.

The MLP model is trained for **10** epochs, and its performance is evaluated on the test data. The training process is depicted in the following figure

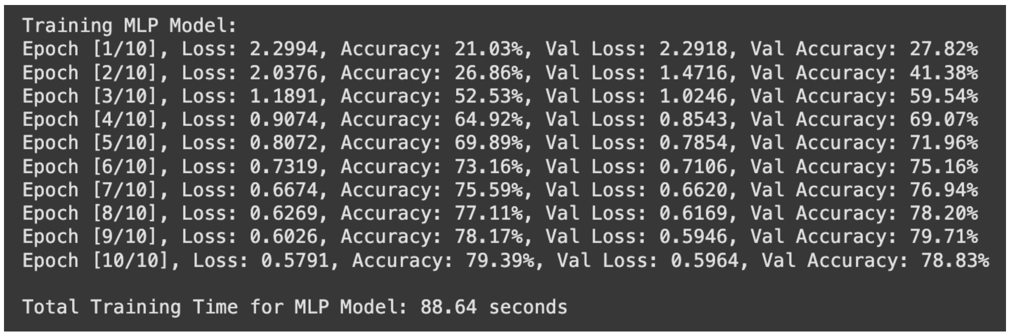
- *Accuracy and Loss Curves: Graphs of training and validation accuracy and loss across epochs.*





The quantitative results of the MLP model are as follows:

Metric	Value
Test Accuracy	~79%
Training Time	88.64 seconds



```
Training MLP Model:
Epoch [1/10], Loss: 2.2994, Accuracy: 21.03%, Val Loss: 2.2918, Val Accuracy: 27.82%
Epoch [2/10], Loss: 2.0376, Accuracy: 26.86%, Val Loss: 1.4716, Val Accuracy: 41.38%
Epoch [3/10], Loss: 1.1891, Accuracy: 52.53%, Val Loss: 1.0246, Val Accuracy: 59.54%
Epoch [4/10], Loss: 0.9074, Accuracy: 64.92%, Val Loss: 0.8543, Val Accuracy: 69.07%
Epoch [5/10], Loss: 0.8072, Accuracy: 69.89%, Val Loss: 0.7854, Val Accuracy: 71.96%
Epoch [6/10], Loss: 0.7319, Accuracy: 73.16%, Val Loss: 0.7106, Val Accuracy: 75.16%
Epoch [7/10], Loss: 0.6674, Accuracy: 75.59%, Val Loss: 0.6620, Val Accuracy: 76.94%
Epoch [8/10], Loss: 0.6269, Accuracy: 77.11%, Val Loss: 0.6169, Val Accuracy: 78.20%
Epoch [9/10], Loss: 0.6026, Accuracy: 78.17%, Val Loss: 0.5946, Val Accuracy: 79.71%
Epoch [10/10], Loss: 0.5791, Accuracy: 79.39%, Val Loss: 0.5964, Val Accuracy: 78.83%

Total Training Time for MLP Model: 88.64 seconds
```

Convolutional Neural Network (CNN) Model

Model Architecture and Implementation

- **CNN Layers**: The CNN model has five convolutional layers:
    - Each layer uses a kernel size of 3x3 and applies ReLU activation.
        - Layer 1: 32 Feature maps (takes grayscale images)
        - Layer 2: Outputs 64 feature maps
        - Layer 3: Outputs 128 feature maps
        - Layer4: Outputs 256 feature maps
        - Layer 5: Outputs 512 feature maps
    - Each followed by ReLU activation
- **Pooling Layers**: After every convolutional layer max-pooling layer with kernel size of 2 is added to reduce spatial dimension and overfitting.
- **MLP Classifier**: The output from the CNN layers is flattened and fed into an MLP layer for final classification.

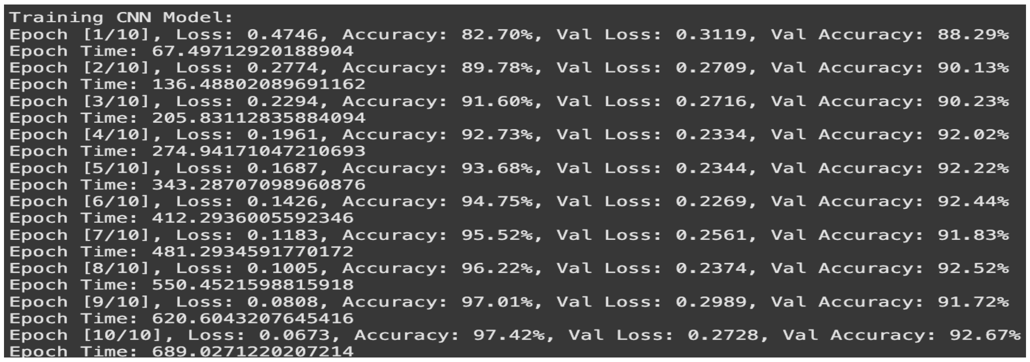
Hyperparameter Tuning

Key hyperparameters were adjusted to improve CNN performance:

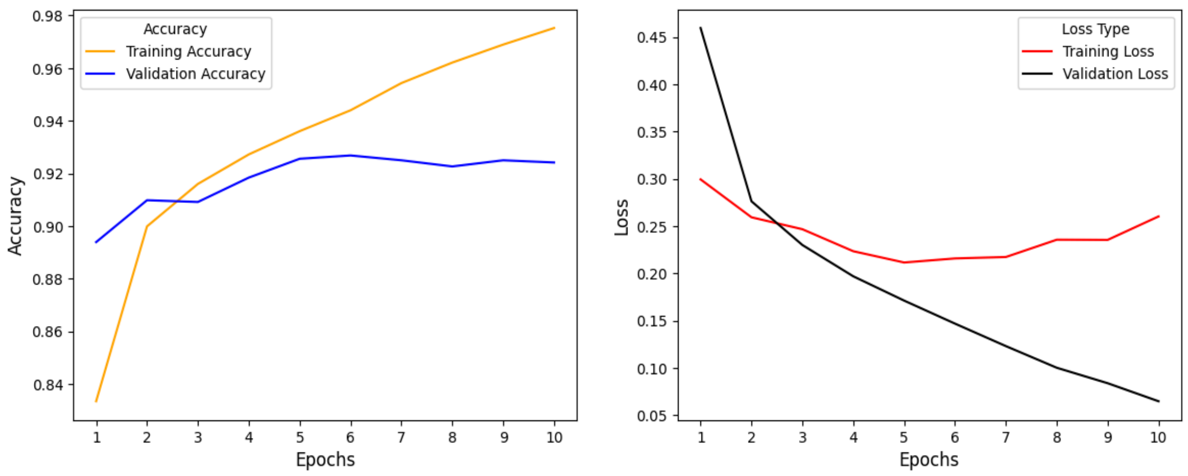
- **Kernel Size and Filter Count**: Experimented with Kernel size 1x1, 2x2, 3x3 but received the best results at 3x3 across layers, while filter count doubled with each layer (32, 64, 128).
- **Weight Initialization**: Different initialization methods, including Xavier and He, were tested. ‘he’ initialization yielded faster convergence and better overall accuracy.

Quantitative results of the CNN model:

Metric	Value
Test Accuracy	~97% (10 Epoch) and ~93%(5 Epoch)
Training Time per Epoch	~8 Minute per Epoch



```
Training CNN Model:
Epoch [1/10], Loss: 0.4746, Accuracy: 82.70%, Val Loss: 0.3119, Val Accuracy: 88.29%
Epoch Time: 67.49712920188904
Epoch [2/10], Loss: 0.2774, Accuracy: 89.78%, Val Loss: 0.2709, Val Accuracy: 90.13%
Epoch Time: 136.48802089691162
Epoch [3/10], Loss: 0.2294, Accuracy: 91.60%, Val Loss: 0.2716, Val Accuracy: 90.23%
Epoch Time: 205.83112835884094
Epoch [4/10], Loss: 0.1961, Accuracy: 92.73%, Val Loss: 0.2334, Val Accuracy: 92.02%
Epoch Time: 274.94171047210693
Epoch [5/10], Loss: 0.1687, Accuracy: 93.68%, Val Loss: 0.2344, Val Accuracy: 92.22%
Epoch Time: 343.28707098960876
Epoch [6/10], Loss: 0.1426, Accuracy: 94.75%, Val Loss: 0.2269, Val Accuracy: 92.44%
Epoch Time: 412.2936005592346
Epoch [7/10], Loss: 0.1183, Accuracy: 95.52%, Val Loss: 0.2561, Val Accuracy: 91.83%
Epoch Time: 481.2934591770172
Epoch [8/10], Loss: 0.1005, Accuracy: 96.22%, Val Loss: 0.2374, Val Accuracy: 92.52%
Epoch Time: 550.4521598815918
Epoch [9/10], Loss: 0.0808, Accuracy: 97.01%, Val Loss: 0.2989, Val Accuracy: 91.72%
Epoch Time: 620.6043207645416
Epoch [10/10], Loss: 0.0673, Accuracy: 97.42%, Val Loss: 0.2728, Val Accuracy: 92.67%
Epoch Time: 689.0271220207214
```



- *CNN: Accuracy and Loss Curves: Graphs of training and validation accuracy and loss across epochs.*

Performance Comparison and Analysis

Both models were evaluated using accuracy and cross-entropy loss on the validation and test sets.

Comparison of MLP vs. CNN

- **MLP**: The best accuracy achieved was around 80%. The model showed signs of overfitting but benefited from dropout regularization.
- **CNN**: The CNN model consistently outperformed the MLP, achieving 92% accuracy. Convolutional layers allowed the CNN to capture spatial hierarchies in the data, which was essential for distinguishing clothing items based on edges and textures.

Hyperparameter Impact

- **Dropout** was crucial for the MLP to generalize well, while **weight initialization** significantly impacted CNN performance.
- **Kernel Size and Filter Count**: These were important for CNNs, with larger filter counts in later layers providing better feature extraction for complex patterns.

Results Summary

Performance Comparison Table:

Model	Training Accuracy	Test Accuracy	Parameters	Training Time per Epoch
MLP	~85%	~84%	**Learning rate:** `0.01` **Number of Epochs:** `10` **Batch Size:** `64`	~8 Seconds/Epoch
CNN	~93%	~91%	**Learning rate:** `0.01` **AdamLR:** `0.001` **Number of Epochs:** `10` **Batch Size:** `64`	~8 Minute/Epoch

Instructions for Running the Models

1. **Requirements**: Ensure torch and torchvision are installed.
2. **Download Dataset**: The Fashion-MNIST dataset is downloaded automatically via PyTorch's dataset loader.
3. **Run Code**: Execute cells sequentially.
4. **Evaluation**: Accuracy and loss results are displayed after each training epoch.

Conclusion

- **Key Findings**: The CNN model significantly outperformed the MLP due to its ability to capture spatial hierarchies, which were crucial for classifying clothing items.

Appendix

Authors Email:

- Karan Arora: karanarora23@iitk.ac.in | Contribution: 25%
- Neelesh Batham: neeleshb24@iitk.ac.in | Contribution: 25%
- Advait Patwardhan: advaitmp24@iitk.ac.in | Contribution: 25%
- Tony Varghese: tonyv23@iitk.ac.in | Contribution: 25%