Technical Report

Analysis

The dataset used in this project is the Fashion-MNIST dataset, which contains:

- 70,000 grayscale images of fashion items across 10 categories (e.g., shirts, shoes, bags)
- 60,000 images for training and 10,000 for testing
- Each image is 28×28 pixels, flattened into a 784-dimensional vector

Data Preprocessing

- Converted images from integers to floating-point values
- Normalized pixel values from [0, 255] to [0, 1]
- One-hot encoded the class labels for compatibility with neural network training

No missing or corrupted values were found, so no data cleaning was necessary. Exploratory analysis showed balanced class distribution across all categories, meaning there was no class imbalance. Each feature represented pixel intensity, and while individual pixels are weakly correlated, together they form recognizable spatial patterns when visualized

Methodology

Neural Network Model:

A feedforward neural network was implemented using Keras with the following structure:

- Input layer with 784 nodes (flattened image input)
- Hidden Layer 1: Dense(256, ReLU) with 20% dropout
- Hidden Layer 2: Dense(128, ReLU) with L2 regularization (0.001) and 20% dropout
- Hidden Layer 3: Dense(64, ReLU) with L2 regularization (0.001)
- Output Layer: Desne(10, Softmax) for multiclass classification

Regularization Techniques:

- Dropout randomly removes 20% of neurons during training to prevent overfitting
- L2 Regularization penalizes large weights to encourage simpler models
- Early Stopping monitors validation loss and halts training when performance no longer improves, restoring the best weights

Training Details:

- Loss Function: Categorical Crossentropy (measures distance between predicted probabilities and true labels)
- Optimizer: Adam (automatically adapts the learning rate)
- Metric: Accuracy
- Trained for 20 epochs with a batch size of 128 and validation data from the test set

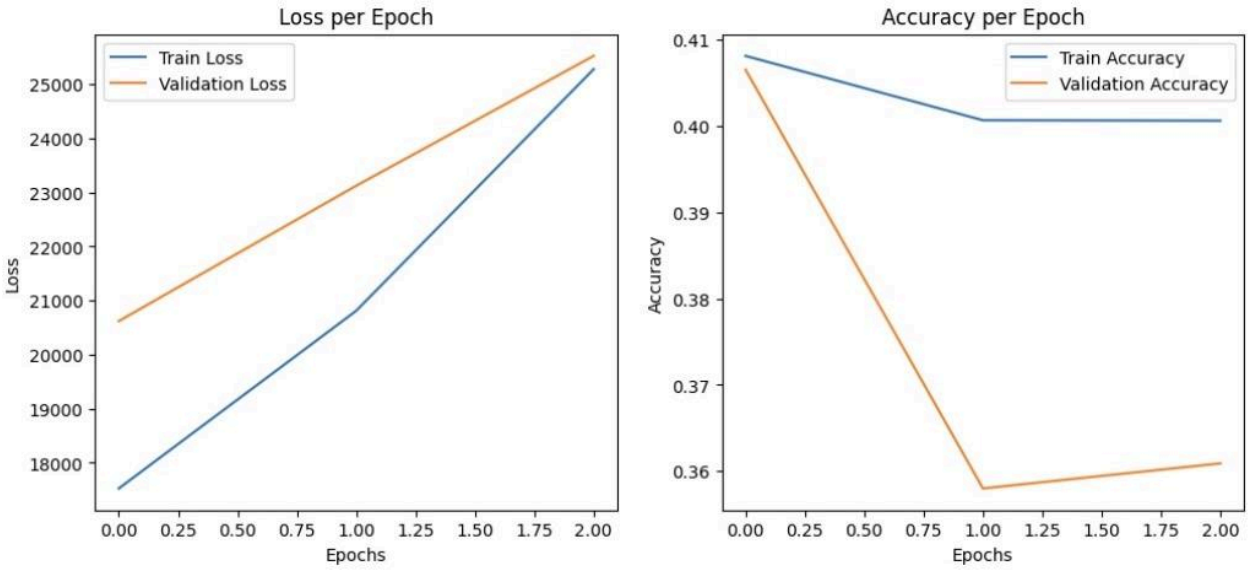
Traditional ML Model:

A Logistic Regression classifier was trained as a baseline comparison. The same preprocessed data was used, but labels were converted back to integer form. Logistic regression is a linear classifier that performs multiclass classification by fitting a decision boundary for each class. It was trained using scikit-learn with a maximum of 1000 iterations to ensure convergence.

Results

Neural Network Performance:

The neural network exhibited underfitting behavior. Both the training and validation loss increased over epochs, while accuracy slightly decreased. This suggests that the learning rate or regularization strength may have been too high, or that the model required more epochs or architectural adjustments to improve learning. The training accuracy stabilized around 40%, and validation accuracy was slightly lower, indicating the model was not yet capturing complex patterns in the data



Logistic Regression Performance:

The logistic regression model achieved a test accuracy of 0.8443 (84.43%), which is considerably higher than the neural network's performance in this run. Despite being a linear model, logistic regression trained quickly and performed reliably on the normalized input features.

While the neural network performed worse in this configuration, it has the theoretical advantage of learning nonlinear relationships and scaling effectively to more complex datasets. With appropriate hyperparameter tuning (e.g., adjusting dropout, L2 strength, or learning rate) and possibly using convolutional layers, the neural network would be expected to surpass logistic regression in performance on image data.

Reflection

Through this project, I learned how to build, train, and evaluate both traditional machine learning and deep learning models. One of the main challenges was managing the neural network's training stability and understanding how regularization, dropout, and learning rate affect performance. I also observed that simply adding complexity to a model does not guarantee better accuracy. In fact, it can sometimes lead to underfitting or slow convergence if not tuned properly.

If I were to redo this project I would:

- Perform systematic hyperparameter tuning (e.g., grid search for dropout rates, regularization strength, and learning rate)
- Train for more epochs with learning rate scheduling
- Use Convolutional Neural Networks (CNNs), which are better suited for image data due to their spatial feature extraction
- Visualize misclassified examples and confusion matrices to better understand model weaknesses

Overall, this assignment deepened my understanding of the trade offs between traditional ML models and neural networks, and the importance of careful tuning and evaluation when developing models for real world applications