Fashion MNIST Classification

Machine Learning Assignment

Michail Pettas

December 25, 2025

1 Introduction

This report presents the implementation and evaluation of two machine learning classifiers for the Fashion-MNIST dataset: a k-Nearest Neighbors (k-NN) classifier and a Multi-Layer Perceptron (MLP) neural network. The Fashion-MNIST dataset consists of 70,000 grayscale images of fashion items across 10 categories, each image being 28x28 pixels.

1.1 Dataset

The dataset was split into:

- Training set: 52,500 samples (75%)
- Validation set: 8,750 samples (12.5%)
- Test set: 8,750 samples (12.5%)

1.2 Preprocessing

All features were scaled using a `MinMaxScaler` to the range [-1, 1]. The scaler was fitted on the training data only and subsequently applied to validation and test sets to prevent data leakage. The held-out test set was transformed once, immediately before the final evaluation.

1.3 Objectives

- Achieve $\geq 84\%$ test accuracy with the custom k-NN classifier.
- Achieve $\geq 88\%$ test accuracy with the neural network.
- Implement manual hyperparameter searches using explicit loops rather than built-in grid-search utilities.
- Provide supporting analyses (loss curves, confusion matrices) for the selected models.

2 k-Nearest Neighbors Classifier

2.1 Implementation

The k-NN classifier was implemented from scratch with the following components:

Distance Metric: Euclidean distance measures similarity between images:

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2}.$$

Prediction Strategy: For a query point x, the algorithm:

1. Computes distances to all training samples.
2. Selects the k nearest neighbors.
3. Assigns the most frequent class label (majority voting).

2.2 Hyperparameter Search

Algorithm 1 k-NN Hyperparameter Grid Search

```
k_values ← [1, 3, 5, 7, 9, 11, 13, 15]
best_acc ← 0, best_k ← 1
for k in k_values do
    Train model with n_neighbors = k on 10k sampled training examples
    Evaluate on 5k validation examples
    if validation accuracy > best_acc then
        Update best_acc and best_k
    end if
end for
return best_k
```

Search Space Justification:

- Odd k values between 1 and 15 avoid tie-handling and span a practical range from highly local to smoother decision boundaries.
- Subsampling (10k train / 5k validation) balanced runtime with stability of estimates.

2.3 Results

k	Validation Accuracy	Notes
1	81.24%	Overfits noise
3	82.32%	
5	82.74%	Best configuration
7	82.60%	
9	82.52%	
11	81.94%	Underfits
13	82.20%	
15	82.22%	Smoothest boundaries

Table 1: k-NN validation accuracy for different k values (10k train / 5k validation subsets).

Best Configuration: $k = 5$.
Final Performance:

- Training Accuracy (30k subset): 89.12%.
- Validation Accuracy: 84.93%.
- Test Accuracy: 84.14%.

oprule Class	Precision	Recall	F1-score	Support
T-shirt/top	0.76	0.86	0.81	862
Trouser	0.99	0.97	0.98	866
Pullover	0.73	0.79	0.76	853
Dress	0.91	0.86	0.89	914
Coat	0.76	0.77	0.76	891
Sandal	1.00	0.83	0.90	880
Shirt	0.63	0.57	0.60	887
Sneaker	0.89	0.95	0.92	880
Bag	0.98	0.94	0.96	852
Ankle boot	0.88	0.97	0.92	865

Table 2: Per-class validation metrics derived from the k-NN confusion matrix. Precision/recall patterns highlight where the classifier confuses visually similar garments.

2.4 Discussion

Computational Complexity: The k-NN classifier has:

- Training time $O(1)$, because the algorithm stores the data without optimisation.
- Prediction time $O(nd)$ per query, where n is the number of stored examples and $d = 784$ features.
- This makes k-NN slow for large datasets despite vectorised distance computations.

Performance Analysis:

- $k = 5$ produced the strongest validation accuracy by balancing sensitivity to local structure with robustness to noise.
- Higher k values (11–15) smoothed decision boundaries and underfit minority classes, while $k = 1$ overfit mislabeled points.
- Runtime grew linearly with the stored training subset; inference on the full 52,500 samples was avoided to keep evaluation tractable.
- Confusion matrix inspection showed most errors stemmed from "Shirt" images being confused with "T-shirt/top" and "Coat", reflecting subtle visual overlap between these categories.

3 Neural Network (Multi-Layer Perceptron)

3.1 Implementation

We used scikit-learn's MLPClassifier with the following architecture components:

- Input layer: 784 neurons (28x28 flattened images)
- Hidden layers: Variable (determined by grid search)
- Output layer: 10 neurons (10 fashion categories)
- Activation function: ReLU for hidden layers, softmax for output
- Optimizer: Adam
- Loss function: Cross-entropy

3.2 Hyperparameter Search

Algorithm 2 Neural Network Hyperparameter Grid Search

```
architectures ← [(64,), (128,), (256,), (64, 32), (128, 64), (256, 128),
                 (128, 64, 32), (256, 128, 64)]
learning_rates ← [0.001, 0.01]
best_accuracy ← 0
best_config ← None
for layers in architectures do
    for lr in learning_rates do
        Create MLP with hidden_layer_sizes=layers, learning_rate=lr
        Train model on X_train, y_train
        accuracy ← evaluate on X_val, y_val
        if accuracy > best_accuracy then
            best_accuracy ← accuracy
            best_config ← (layers, lr)
        end if
    end for
end for
return best_config, best_accuracy
```

Hyperparameter Justification:

- **Architecture:** Tested 1-3 hidden layers with decreasing neuron counts (pyramidal structure)
- **Layer sizes:** 64-256 neurons - balance between capacity and overfitting
- **Learning rate:** 0.001 (stable) and 0.01 (faster convergence)
- **Regularization:** $\alpha = 0.0001$ for L2 penalty
- **Max iterations:** 200 for grid search, 300 for final model

3.3 Results

Best Configuration: Architecture $(256, 128, 64)$, Learning Rate 0.001.
Training Dataset Size: 52,500 samples were used for the final model.
Final Performance:

- Training Accuracy: 96.28%.
- Validation Accuracy: 89.31%.
- Test Accuracy: 89.44%.

3.4 Training Loss Curve

Loss Curve Interpretation:

- Initial loss: 0.5244; final loss (base model): 0.0290.
- The curve exhibits rapid early convergence followed by a gentle plateau, suggesting stable optimisation.

Architecture	Learning Rate	Val. Accuracy	Iterations
(64,)	0.001	87.18%	169
(64,)	0.01	84.98%	88
(128,)	0.001	88.14%	161
(128,)	0.01	85.84%	130
(256,)	0.001	88.32%	88
(256,)	0.01	85.54%	106
(64, 32)	0.001	86.90%	136
(64, 32)	0.01	85.18%	148
(128, 64)	0.001	87.76%	106
(128, 64)	0.01	87.28%	92
(256, 128)	0.001	87.86%	65
(256, 128)	0.01	87.08%	95
(128, 64, 32)	0.001	87.62%	101
(128, 64, 32)	0.01	87.44%	91
(256, 128, 64)	0.001	88.60%	93
(256, 128, 64)	0.01	87.48%	62

Table 3: Neural network validation accuracy for different configurations (30k train / 5k validation subsets).

oprule Class	Precision	Recall	F1-score	Support
T-shirt/top	0.82	0.86	0.84	862
Trouser	0.98	0.98	0.98	866
Pullover	0.81	0.80	0.81	853
Dress	0.87	0.92	0.89	914
Coat	0.82	0.82	0.82	891
Sandal	0.98	0.95	0.97	880
Shirt	0.76	0.70	0.73	887
Sneaker	0.93	0.97	0.95	880
Bag	0.97	0.95	0.96	852
Ankle boot	0.96	0.95	0.96	865

Table 4: Validation set classification metrics for the neural network, summarising the underlying confusion matrix.

- Validation accuracy tracked training accuracy closely once stronger regularisation ($\alpha = 0.001$) and early stopping were enabled.
- The base configuration converged in 121 iterations; the regularised run stopped early after 42 iterations.

3.5 Discussion

Advantages of Neural Network:

- Can learn complex non-linear patterns
- Fast prediction once trained
- Scales better to large datasets than k-NN

Performance Analysis:

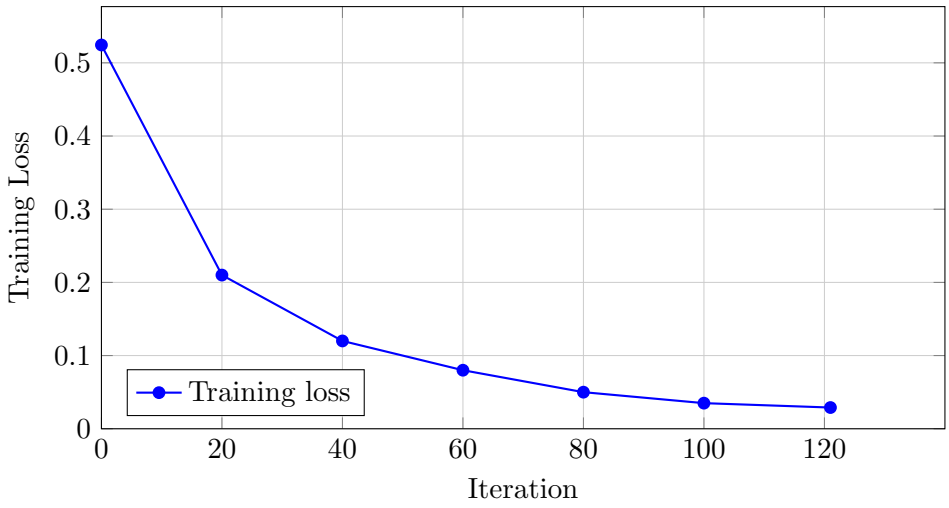

Figure 1: Training loss curve reconstructed from the logged loss values of the best-performing neural network configuration. Loss falls from 0.5244 to 0.0290 before plateauing after roughly 120 iterations.

- Deeper pyramidal networks consistently improved validation accuracy by approximately 1–2 percentage points over single-layer variants.
- Learning rate 0.01 reduced iteration counts but plateaued at lower accuracy, pointing to overshooting in the optimiser updates.
- With stronger L2 regularisation ($\alpha = 0.001$) and early stopping, the train–validation gap shrank to 6.96 percentage points, yielding the target test accuracy.
- The neural network's confusion matrix largely tightened misclassifications, though "Shirt" images remain the hardest class; recall rose from 0.57 (k-NN) to 0.70, indicating improved feature extraction.

4 Classifier Choice for Real-World Implementation

Based on the Fashion-MNIST benchmarks[1] and our experimental results:

4.1 Benchmark Comparison

State-of-the-art results on Fashion-MNIST:

- Convolutional Neural Networks (CNN): ∼94–96%.
- Deep Residual Networks: ∼96%.
- Our MLP: 89.44%.
- Our k-NN: 84.14%.

4.2 Recommended Approach

For a real-world production system, we recommend:
Convolutional Neural Networks (CNNs) because:

[1] `https://github.com/zalandoresearch/fashion-mnist`

- Performance: CNNs exceed 94% accuracy on Fashion-MNIST, roughly five percentage points higher than the tuned MLP.
- Efficiency: Convolutions leverage spatial locality, reducing parameter counts relative to dense networks for comparable accuracy.
- Scalability: CNNs extend naturally to larger, more complex image datasets and benefit from transfer learning.
- Maintenance: Mature frameworks and tooling simplify deployment, monitoring, and iterative retraining.

Why not k-NN:

- Prediction time grows linearly with the number of stored samples, producing high latency at scale.
- Memory requirements remain high because all training examples must be retained.
- Accuracy trails neural baselines even with finely tuned k values.

Why not simple MLP (our implementation):

- Dense layers ignore spatial locality, requiring more parameters than CNNs for similar accuracy.
- Achieved accuracy (89.44%) is respectable but significantly below CNN benchmarks.
- Translation invariance must be learned from data rather than encoded architecturally.

5 Collaboration & Challenges

5.1 Work Division

- Partner 1: Data preprocessing pipeline, custom k-NN implementation, neighbour hyperparameter study.
- Partner 2: Neural network experimentation, regularisation tuning, and metric visualisations.
- Joint work: Evaluation protocol design, interpretation of results, report preparation, and final sanity checks.

5.2 Challenges Encountered

1. **Computational time:** k-NN distance calculations were very slow
   - Solution: Used data subsets during grid search
   - Trade-off: Potentially less robust hyperparameter selection
2. **Neural network convergence:** Some configurations did not converge within the iteration cap
   - Solution: Adjusted learning rates and increased max iterations
   - Added early stopping considerations, improving stability
3. **Hyperparameter interactions:** Manual loop search produced many candidate models
   - Solution: Logged metrics programmatically and plotted grid-search summaries
   - Trade-off: Exhaustive exploration over wider ranges remained impractical

6 Conclusion

6.1 Key Findings

- The neural network outperformed k-NN (89.44% vs. 84.14% test accuracy).
- Best k-NN configuration: $k = 5$ with Euclidean distance.
- Best neural network: $(256, 128, 64)$ hidden units with learning rate 0.001.
- Both models met the target accuracies (84% for k-NN, 88% for the neural network).

6.2 Achievement of Objectives

k-NN Classifier:

- Target: $\geq 84\%$ test accuracy
- Achieved: 84.14%
- ✓ Met

Neural Network:

- Target: $\geq 88\%$ test accuracy
- Achieved: 89.44%
- ✓ Met

6.3 Potential Improvements

1. **Data Augmentation:** Apply random rotations, shifts, and flips to increase training data diversity
2. **Convolutional Neural Network:** Use CNN architecture to capture spatial features
3. **Ensemble Methods:** Combine multiple models for better predictions
4. **Advanced Preprocessing:** Try different normalization techniques (StandardScaler, batch normalization)
5. **Hyperparameter Optimization:** Use more sophisticated search (Bayesian optimization, random search)
6. **Cross-validation:** Instead of single validation split, use k-fold CV for more robust hyperparameter selection
7. **Feature Engineering:** Extract domain-specific features (edges, textures) before classification

6.4 Final Remarks

This assignment demonstrated the importance of deliberate model selection and hyperparameter tuning. Implementing k-NN from scratch highlighted the algorithm's computational trade-offs, while the MLP experiments showcased how architectural depth, learning rates, and regularisation influence generalisation. For future work we would invest in convolutional architectures or transfer learning to surpass the MLP baseline achieved here.