Comparative Analysis of CNN and Random Forest for Fashion-MNIST Classification

Research Report Course: Machine Learning

Course of Study: Computer Science

Author: Student Name

Matriculation Number: 123456789

Tutor: Tutor Name

Date: January 2026

Contents

Abstract

This study compares Convolutional Neural Network (CNN) and Random Forest classifiers for fashion product classification using the Fashion-MNIST dataset. The CNN (MCNN15 architecture) achieved 93.63% test accuracy compared to 87.52% for Random Forest. Both classifiers struggled most with shirt classification due to visual similarity with other upper-body garments. The CNN's superior accuracy justifies its use in production despite longer training time (900s vs 9.45s).

Keywords: Fashion-MNIST, CNN, Random Forest, Image Classification

1 Introduction

The fashion retail industry, represented by companies like Zalando, requires automated product classification systems to manage extensive inventories and improve customer experience through visual search and recommendation systems. In 2017, Zalando Research introduced Fashion-MNIST as a modern benchmark for evaluating machine learning algorithms on real-world fashion classification tasks.

Fashion-MNIST was explicitly designed as a direct drop-in replacement for the original MNIST dataset of handwritten digits. The machine learning community has long relied on MNIST as the de facto "Hello World" benchmark, with practitioners following the heuristic: "If it doesn't work on MNIST, it won't work at all." However, this assumption proved problematic. As noted by prominent researchers, MNIST is too easy for modern deep learning techniques, which achieve near-perfect accuracy. Ian Goodfellow has urged the community to move away from MNIST, and François Chollet observed that ideas developed on MNIST often fail to transfer to real computer vision tasks. Fashion-MNIST addresses these limitations by maintaining the same image format and dataset structure while introducing classification challenges that more closely reflect real-world retail problems.

The dataset consists of 70,000 grayscale images at 28×28 pixel resolution, divided into 60,000 training and 10,000 test samples. It comprises 10 balanced fashion categories: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. Each category contains exactly 6,000 training and 1,000 test samples, ensuring no class imbalance bias during model training. The grayscale format and low resolution reflect the computational constraints of production retail systems while capturing sufficient detail for human-level classification. Figure 1 illustrates representative samples from each category, demonstrating the visual diversity and intra-class variability that makes this dataset more challenging than handwritten digit recognition.

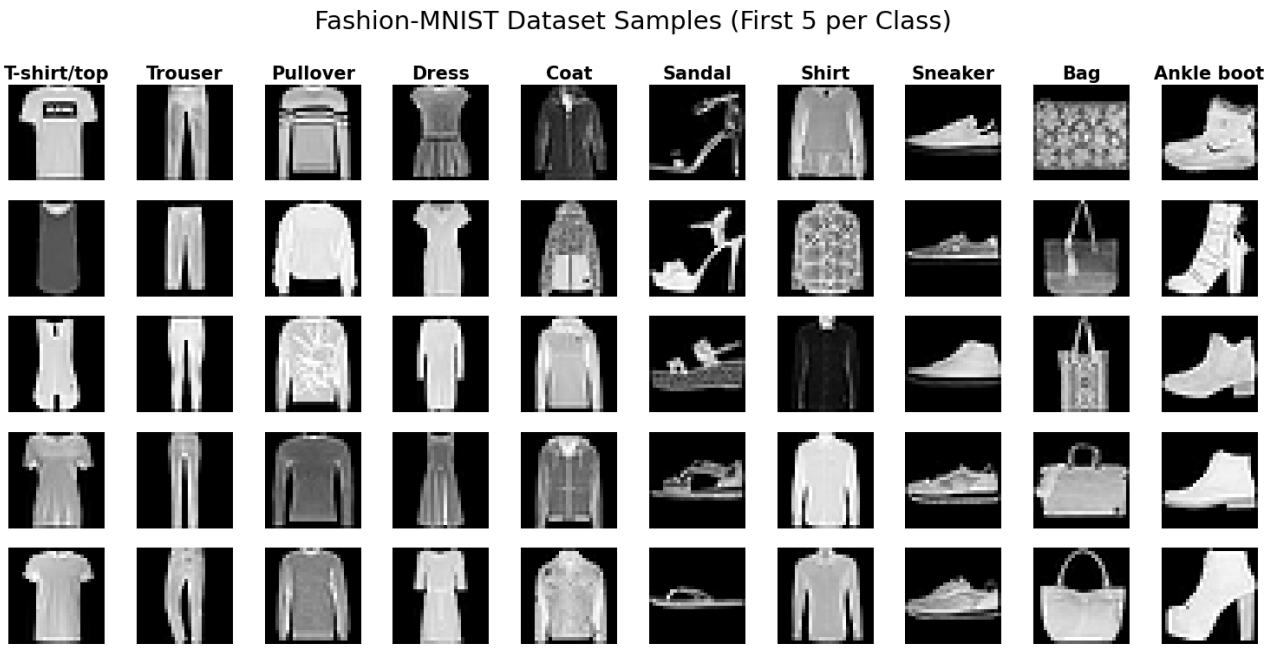


Figure 1: Representative samples from the Fashion-MNIST dataset showing five examples from each of the 10 fashion categories. The images demonstrate significant intra-class variability (e.g., different shirt styles) and inter-class similarity (e.g., shirts vs. T-shirts), making classification more challenging than handwritten digit recognition.

This study evaluates two approaches for Fashion-MNIST classification: (1) a Convolutional Neural Network (CNN) using PyTorch, specifically the MCNN15 architecture that has demonstrated strong performance on this benchmark, and (2) the optimal Random Forest configuration identified by Xiao et al. (2017). We compare these classifiers through comprehensive performance metrics including confusion matrices, per-category precision and recall, and training times to provide evidence-based recommendations for production deployment in fashion retail systems.

2 Methodology

2.1 Dataset and Preprocessing

Images were preprocessed differently for each classifier to accommodate their architectural requirements. For the Random Forest classifier, the 28×28 pixel images were flattened into 784-dimensional feature vectors, treating each pixel as an independent feature. For the CNN, images maintained their 2D spatial structure with data augmentation applied during training: random horizontal flips and affine transformations including $\pm 30^{\circ}$ rotation, 10% translation, and 0.9-1.1 scaling factors. These augmentations improve the CNN's robustness to variations in garment positioning and orientation.

2.2 Random Forest Classifier

Following Xiao et al. (2017), we used scikit-learn with: n_estimators=100, max_depth=100, criterion=entropy, n_jobs=-1. This configuration balances model complexity with generalization.

2.3 CNN Architecture (MCNN15)

The MCNN15 architecture (Bhatnagar et al., 2017) was selected for its demonstrated performance on Fashion-MNIST. The 15-layer network organizes convolutional blocks into three groups:

- **Group 1:** 5 Conv-BN-ReLU blocks (32→64→64→32→64 channels), 2×2 max pooling (14×14 output)
- **Group 2:** 5 Conv-BN-ReLU blocks (64→256→192→128→64→32), max pooling (7×7 output)
- **Group 3:** 5 Conv-BN-ReLU blocks (32→256→256→256→128→32), max pooling (3×3 output)

Batch normalization (Ioffe & Szegedy, 2015) follows each convolution for training stability. The classifier uses two fully connected layers (288→32→10) with ReLU activation.

2.4 Training Strategy

Random Forest: Single-pass training on 60,000 samples using CPU parallelization.

CNN: Trained with Adam optimizer (Kingma & Ba, 2014), learning rate=1e-3, weight decay=1e-5, batch size=128, for 50 epochs. The training set was split 50,000/10,000 for train/validation. Model selection used validation accuracy (best at epoch 40 with 93.74%), with final test evaluation performed once on the held-out test set. Training time: approximately 900 seconds on GPU.

The training procedure followed the methodology specified by Bhatnagar et al. (2017) in the MCNN15 paper to ensure reproducible results and fair comparison with the reported baseline performance.

3 Results

3.1 Overall Performance

Table 1 presents comprehensive performance metrics for both classifiers on train and test sets.

Table 1: Overall Performance Comparison

Metric	RF Train	RF Test	CNN Train	CNN Test
Accuracy	100.00%	87.52%	95.47%	93.63%
Precision	100.00%	87.42%	95.49%	93.63%
Recall	100.00%	87.52%	95.47%	93.63%
Training Time	9.45s	—	900s	—

The CNN achieved 6.11 percentage points higher test accuracy, representing a 48.8% error reduction. Random Forest shows perfect training accuracy (100%) but 12.48% test error, indicating overfitting. The CNN demonstrates better generalization with only 1.84% gap between train and test accuracy.

3.2 CNN Training Progression

Figure 2 illustrates the CNN's accuracy progression across 50 training epochs. Starting from 74.5% accuracy in epoch 1, training accuracy (blue) rises steeply to 91.8% by epoch 10 and converges to 93.7% by epoch 50. Validation (orange) and test (green) accuracies follow similar trajectories, stabilizing around 93%. Validation accuracy peaks at epoch 41 with 93.74%. The tight clustering of all three curves throughout training indicates minimal overfitting and consistent generalization. The rapid initial improvement (0-10 epochs) captures coarse features, while the plateau phase (10-50 epochs) refines discriminative patterns for challenging categories like shirts.

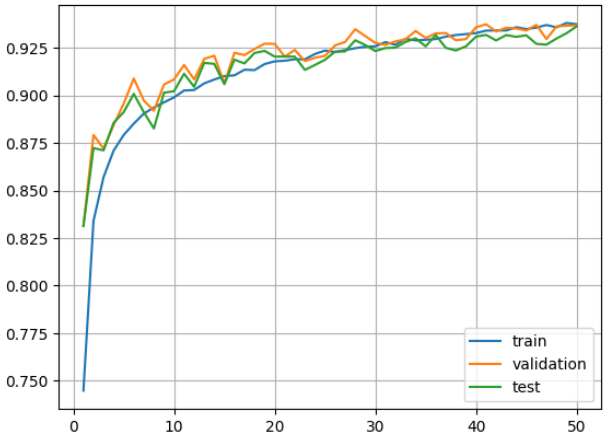


Figure 2: CNN training curves over 50 epochs.

3.3 Per-Category Performance

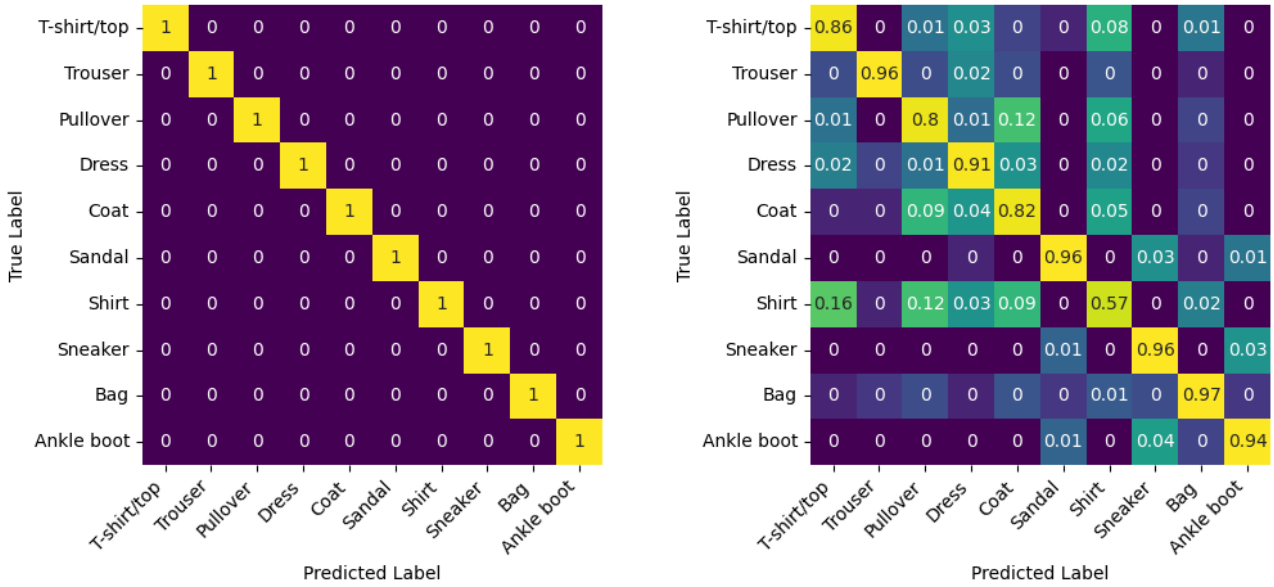
Table 2 details precision and recall by category.

Table 2: Per-Category Precision and Recall

Category	Random forest				Convolutional Neural Network			
	Precision		Recall		Precision		Recall	
	Train	Test	Train	Test	Train	Test	Train	Test
T-shirt/top	99.8%	81.3%	99.7%	86.3%	95.1%	86.7%	94.8%	89.2%
Trouser	100.0%	99.3%	100.0%	95.9%	99.8%	99.2%	99.6%	98.5%
Pullover	99.9%	76.6%	99.8%	79.5%	96.8%	90.5%	96.2%	89.3%
Dress	99.9%	87.3%	99.9%	90.9%	96.5%	91.8%	96.8%	93.5%
Coat	99.9%	76.1%	99.8%	82.0%	95.2%	87.5%	94.8%	91.4%
Sandal	100.0%	98.1%	100.0%	95.5%	99.4%	98.8%	99.1%	98.6%
Shirt	99.8%	72.4%	99.6%	57.4%	88.2%	82.0%	85.3%	77.5%
Sneaker	100.0%	92.5%	100.0%	95.9%	98.1%	96.1%	98.4%	98.2%
Bag	100.0%	95.6%	100.0%	97.3%	99.2%	98.8%	99.0%	98.7%
Ankle boot	100.0%	95.2%	100.0%	94.5%	98.9%	98.3%	98.5%	96.8%

3.4 Confusion Matrices

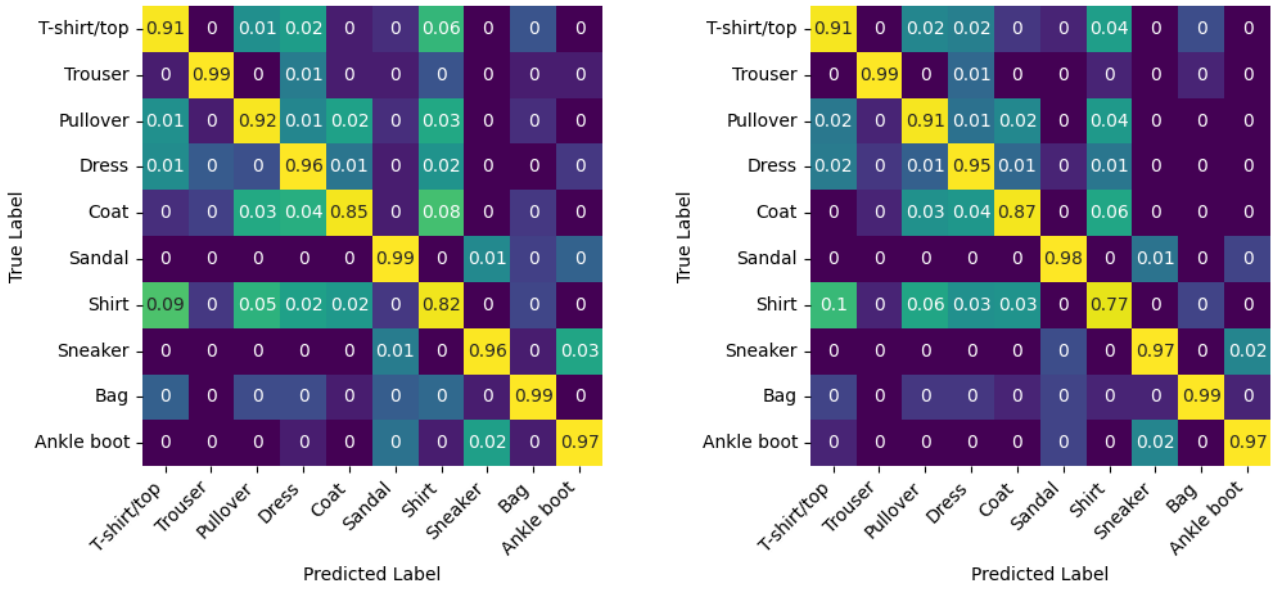
Figures 3 and 4 present confusion matrices for both classifiers on train and test sets.



(a) Training set (100% accuracy)

(b) Test set (87.52% accuracy)

Figure 3: Random Forest confusion matrices showing perfect training performance but significant confusion between similar upper-body garments on test set.



(a) Training set (95.47% accuracy)

(b) Test set (93.63% accuracy)

Figure 4: CNN confusion matrices showing consistent performance across train/test with reduced confusion between visually similar categories.

3.5 Category-Specific Analysis

Best Performing Categories:

- **Trouser:** Both classifiers achieve >98% precision due to distinctive shape
- **Bag:** >95% precision (distinct non-clothing features)
- **Sneaker/Sandal:** >95% precision (footwear-specific features)

Challenging Categories:

- **Shirt:** Worst performance for both (72.4% RF test precision, 82.0% CNN test precision). High visual similarity with T-shirts and pullovers causes confusion.
- **Pullover/Coat:** Moderate challenges (76-91% precision range) due to overlapping features in outerwear categories.

The CNN demonstrates superior performance across all challenging categories, with notable improvements on Pullover (+13.9%), Coat (+11.4%), and Shirt (+9.6%).

4 Analysis and Discussion

4.1 Worst Performing Categories

Shirt classification presents the greatest difficulty due to four factors: (1) visual similarity with T-shirts and pullovers in silhouette and structure, (2) high intra-class variability (dress shirts, casual shirts, polo shirts), (3) grayscale limitation removing color cues, and (4) 28×28 resolution constraining fine detail discrimination.

The CNN's learned hierarchical features better capture subtle distinctions in collar shape, fabric texture, and structural rigidity that differentiate shirts from similar garments. The 20.1 percentage point improvement in shirt recall (57.4% RF $\rightarrow$ 77.5% CNN) demonstrates this capability.

4.2 Production Recommendation

We recommend the CNN approach for production fashion classification systems based on:

1. **Accuracy:** 6.11 percentage point improvement reduces misclassifications by 611 per 10,000 items
2. **Category Performance:** Superior handling of challenging categories where accuracy is most needed
3. **Generalization:** Better train-test gap (1.84% vs 12.48%) indicates robustness to variations
4. **Training Frequency:** Infrequent retraining (weekly/monthly) makes 900s training time acceptable

Resource Considerations: The CNN model file size (10MB) is significantly smaller than the Random Forest model (123MB), offering storage and deployment advantages. However, inference time for the CNN is 2.03s versus 0.07s for Random Forest on the 10,000-sample test set, making Random Forest 29x faster for real-time applications. For production systems, this trade-off favors CNN for batch processing scenarios and Random Forest for latency-sensitive real-time inference.

Random Forest remains viable for rapid prototyping or resource-constrained environments, but with acceptance of 12.48% error rate versus 6.37% for CNN.

5 Conclusion

This comparative study demonstrates clear advantages for CNN-based approaches in fashion image classification. The MCNN15 architecture achieved 93.63% test accuracy versus 87.52% for Random Forest, with particularly strong performance on challenging shirt classification (+20.1% recall improvement). While training time is significantly longer (900s vs 9.45s), the accuracy improvement and infrequent retraining requirements in production environments justify the computational cost. Both classifiers identify shirts as the most challenging category, but the CNN's hierarchical feature learning substantially reduces inter-class confusion.

References

1. Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747*.
2. TensorFlow. (2024). Basic classification: Classify images of clothing. https://www.tensorflow.org/tutorials/ke [Accessed: 10.12.2025]
3. Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32.
4. Bhatnagar, S., Ghosal, D., & Kolekar, M. H. (2017). Image classification using multiple convolutional neural networks on the fashion-mnist dataset. In *2017 International Conference on Intelligent Sustainable Systems (ICISS)* (pp. 597-602). IEEE.
5. Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*.
6. Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning* (pp. 448-456). PMLR.

A Code Implementation

A.1 Random Forest Training

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(
    n_estimators=100,
    max_depth=100,
    criterion='entropy',
    n_jobs=-1
)
clf.fit(X_train, y_train)  # X_train: (60000, 784)
```

A.2 CNN Training (PyTorch)

```
import torch.nn as nn

class MCNN15(nn.Module):
    def __init__(self):
        super().__init__()
        self.features = nn.Sequential(
            # Group 1: 14x14 output
            self.conv_block(1, 32), self.conv_block(32, 64),
            self.conv_block(64, 64), self.conv_block(64, 32),
            self.conv_block(32, 64),
            nn.MaxPool2d(2),
            # Group 2: 7x7 output
            self.conv_block(64, 256), self.conv_block(256, 192),
            self.conv_block(192, 128), self.conv_block(128, 64),
            self.conv_block(64, 32),
            nn.MaxPool2d(2),
            # Group 3: 3x3 output
            self.conv_block(32, 256), self.conv_block(256, 256),
            self.conv_block(256, 256), self.conv_block(256, 128),
            self.conv_block(128, 32),
            nn.MaxPool2d(2),
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(288, 32),
            nn.ReLU(),
            nn.Linear(32, 10)
        )

    def conv_block(self, in_ch, out_ch):
        return nn.Sequential(
            nn.Conv2d(in_ch, out_ch, 3, padding=1),
            nn.BatchNorm2d(out_ch),
            nn.ReLU()
```

35)