Classifying Fashion Products

Project Report

Course: DLBAIPCV01 - Project: Computer Vision

Course of Study: EU ODSP BAAI - Applied Artificial Intelligence

Author: Dmitrii Lavrov

Matriculation Number: 42307367

Tutor: Gissel Velarde

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).

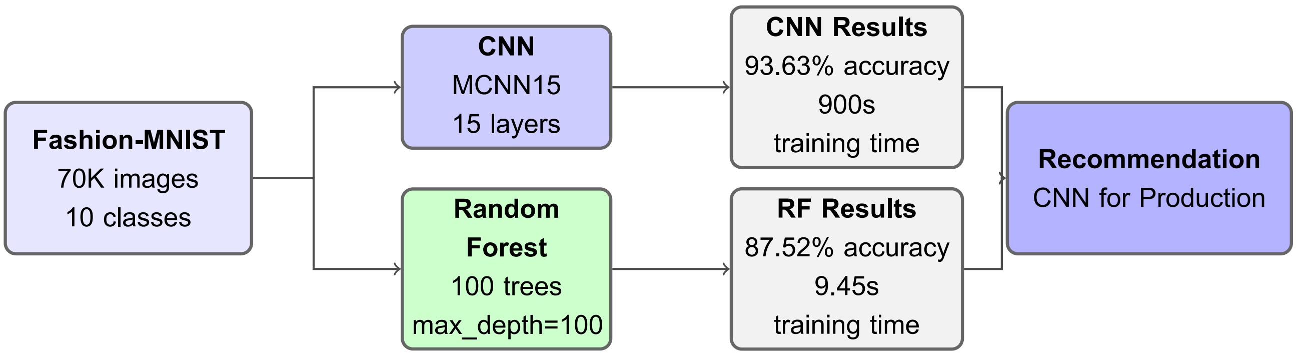


Figure 1: Graphical Abstract: Comparative analysis workflow from Fashion-MNIST dataset through CNN and Random Forest classifiers to performance metrics and production recommendation.

1 Introduction

The fashion retail industry requires automated product classification for inventory management and visual search applications. While Convolutional Neural Networks (CNNs) demonstrate strong performance on fashion classification, traditional methods such as Random Forest remain relevant due to faster training times, interpretability, and lower computational requirements.

Prior work on Fashion-MNIST focuses primarily on maximizing accuracy through increasingly complex architectures, without addressing practical trade-offs that influence production deployment decisions. Production systems must balance accuracy against training time, model size, and inference latency.

This study addresses this gap by comparing MCNN15 (Bhatnagar et al., 2017), a 15-layer CNN optimized for Fashion-MNIST, against the optimal Random Forest configuration (Xiao et al., 2017). These classifiers are evaluated across accuracy, per-category metrics, confusion patterns, training times, and model sizes to provide evidence-based recommendations for production systems.

Section 2 reviews CNN architectures and benchmarking studies. Section 3 describes the Fashion-MNIST dataset. Section 4 outlines methodology. Sections 5 and 6 present results and analysis, followed by conclusions in Section 7.

2 Literature Review

2.1 Evolution of CNN Architectures for Image Classification

CNN development has progressed from LeCun et al. (1998)'s LeNet-5 through Krizhevsky et al. (2012)'s AlexNet to modern deep architectures. Simonyan and Zisserman (2014)'s VGG demonstrated that small 3×3 filters stacked deeply (16-19 layers) achieve superior performance, while He et al. (2016)'s ResNet introduced skip connections enabling networks exceeding 100 layers. These advances established the foundational patterns—convolutional feature extraction, progressive depth, and residual learning—that underpin contemporary computer vision.

2.2 Fashion-MNIST Benchmarking Studies

Since its introduction by Xiao et al. (2017), Fashion-MNIST has become a standard benchmark with CNN accuracies ranging from 90% to 99% (Mukhamediev, 2024). Simple 4-6 layer CNNs achieve 91-93% accuracy, while deeper architectures reach 96-99%. Bbouzidi et al. (2024) demonstrated that CNNs remain competitive against Vision Transformers on this dataset due to their inductive biases for local features. Cavallo et al. (2022) established that moderate-depth networks (15-20 layers) offer optimal accuracy-efficiency trade-offs, informing architecture selection for this study.

2.3 CNN vs Traditional Machine Learning for Image Classification

CNNs automatically learn hierarchical spatial features from raw pixels, whereas traditional methods like Random Forest require flattened vectors and manual feature engineering. Xiao et al. (2017) demonstrated this performance gap on Fashion-MNIST: Random Forest achieved 87.5% accuracy versus 91% for shallow CNNs. CNNs preserve spatial relationships through convolution and achieve translation invariance via pooling, while flattening destroys two-dimensional structure. Sathyadevan and Gangadharan (2015) noted that although Random Forest trains faster (seconds vs. minutes), CNNs' superior generalization justifies their use in production systems where accuracy impacts revenue.

2.4 Selection of MCNN15 Architecture

MCNN15 (Bhatnagar et al., 2017) was selected for four reasons: (1) native Fashion-MNIST optimization without input modification, (2) optimal depth-efficiency balance with 15 layers and batch normalization (Ioffe & Szegedy, 2015) that achieves 93.6% accuracy without overfitting, (3) groupwise architecture with progressive channel expansion (32$\rightarrow$64$\rightarrow$256) capturing multi-scale features from textures to garment shapes, and (4) documented performance providing a clear benchmark for Random Forest comparison. Unlike VGG-16 or ResNet-50, MCNN15 offers competitive accuracy with lower computational requirements on this relatively small dataset.

3 Dataset Description

3.1 Origin and Motivation

Fashion-MNIST was introduced by Zalando Research in 2017 as a modern replacement for MNIST, which had become too easy for contemporary deep learning (CNNs now achieve $>$99.7% accuracy) (Xiao et al., 2017). This dataset provides a more challenging benchmark for fashion product classification in e-commerce applications including automated inventory management and visual search.

3.2 Dataset Statistics

Fashion-MNIST comprises 70,000 grayscale images at 28$\times$28 resolution, divided into 60,000 training and 10,000 test samples. This format serves as a drop-in MNIST replacement while providing greater classification challenge. The grayscale format and compact resolution balance computational efficiency with sufficient visual detail for accurate garment classification.

3.3 Class Distribution

The dataset contains 10 balanced fashion categories with exactly 6,000 training and 1,000 test samples per class. This balanced distribution eliminates the need for class weighting or specialized sampling techniques, enabling straightforward comparison between classifiers.

3.4 Class Labels and Examples

Table 1 presents the complete list of Fashion-MNIST categories with class indices and representative characteristics.

Table 1: Fashion-MNIST Class Labels and Descriptions

Index	**Label**	**Description**
0	T-shirt/top	Short-sleeved upper body garments
1	Trouser	Long pants covering both legs
2	Pullover	Knitted upper body garments
3	Dress	One-piece garment covering torso and legs
4	Coat	Outerwear worn over other clothing
5	Sandal	Open footwear with straps
6	Shirt	Buttoned upper body garments
7	Sneaker	Athletic footwear
8	Bag	Hand-carried containers
9	Ankle boot	Footwear covering ankle

Figure 2 illustrates representative samples demonstrating the visual diversity, intra-class variability (e.g., different shirt styles), and inter-class similarities (e.g., shirts vs. T-shirts) that distinguish Fashion-MNIST from digit recognition.

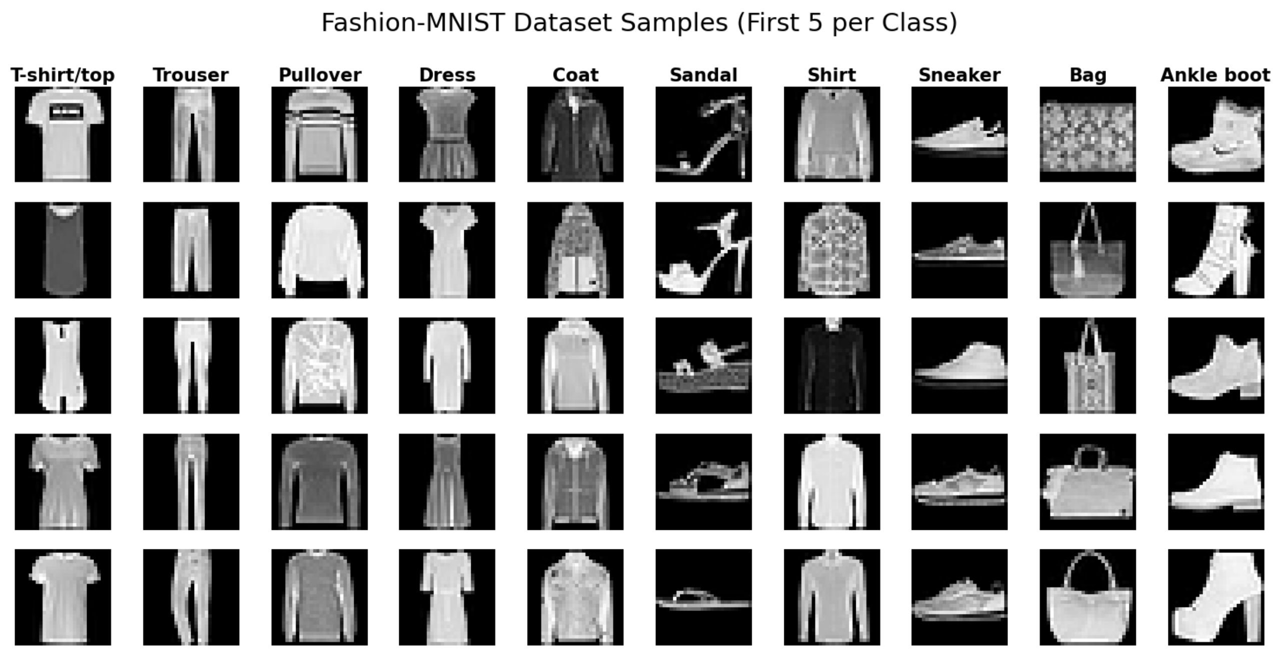


Figure 2: Representative samples from the Fashion-MNIST dataset showing five examples from each of the 10 fashion categories. The images demonstrate significant intra-class variability (e.g., different shirt styles) and inter-class similarity (e.g., shirts vs. T-shirts), making classification more challenging than handwritten digit recognition.

4 Methodology

4.1 Dataset and Preprocessing

Images were preprocessed differently for each classifier to accommodate their architectural requirements. For the Random Forest classifier, the 28$\times$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.

4.2 Random Forest Classifier

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

4.3 CNN Architecture (MCNN15)

The MCNN15 architecture (Bhatnagar et al., 2017) comprises 15 convolutional layers organized into three groups with batch normalization (Ioffe & Szegedy, 2015) and max pooling, followed by fully connected layers (288$\rightarrow$32$\rightarrow$10). Figure 3 illustrates the architecture.

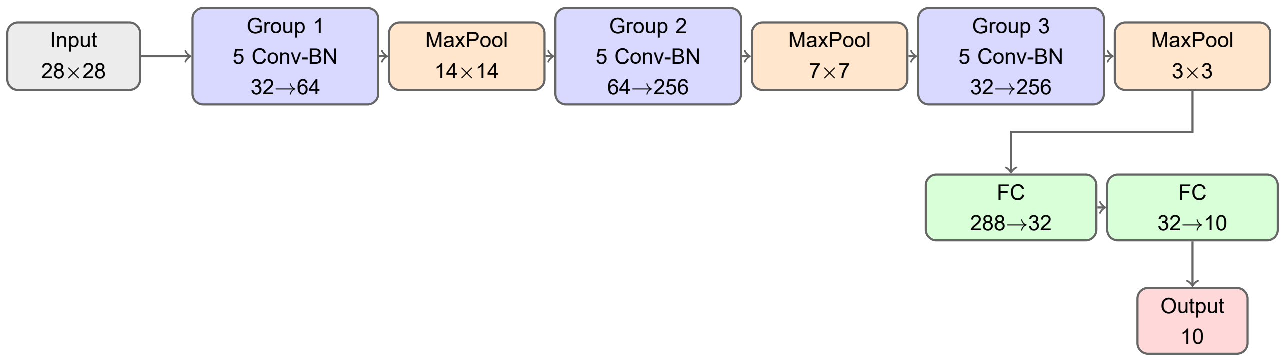


Figure 3: MCNN15 architecture diagram showing the three convolutional groups with batch normalization and max pooling, followed by fully connected classifier layers. Each group contains 5 Conv-BN-ReLU blocks with progressive channel expansion.

4.4 Training Strategy

4.4.1 Random Forest Training

Random Forest training used the complete 60,000-sample training set flattened to 784-dimensional vectors. No data augmentation was applied. Training completed in approximately 9.45 seconds using CPU parallelization.

4.4.2 CNN Training

The CNN was trained using Adam optimizer (Kingma & Ba, 2014) with cross-entropy loss, learning rate=1e-3, weight decay=1e-5, and batch size=128 for 50 epochs. The training set was split 50,000/10,000 for training/validation. Data augmentation included random horizontal flips, rotation ($\pm 30^\circ$), translation (10%), and scaling (0.9-1.1). The best validation accuracy (93.74%) was achieved at epoch 40. Training time: approximately 900 seconds on GPU. The procedure followed Bhatnagar et al. (2017) methodology.

5 Results

5.1 Overall Performance

Table 2 presents comprehensive performance metrics for both classifiers on train and test sets.

Table 2: 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 (48.8% error reduction). Random Forest shows perfect training accuracy (100%) but 12.48% test error, indicating severe overfitting. The CNN demonstrates better generalization with only 1.84% train-test gap.

5.2 CNN Training Progression

Figure 4 shows CNN accuracy progression from 74.5% (epoch 1) to 93.7% (epoch 50). Validation (orange) and test (green) curves track closely with training (blue), indicating minimal overfitting. Validation peaks at epoch 41 (93.74%). Rapid improvement (epochs 0-10) captures coarse features; the plateau (10-50) refines patterns for challenging categories.

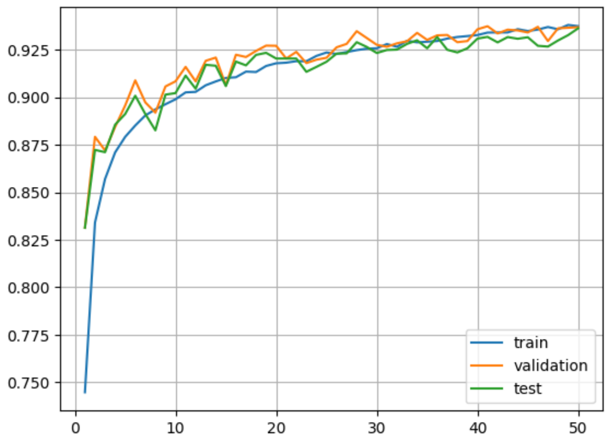


Figure 4: CNN training curves over 50 epochs.

5.3 Per-Category Performance

Table 3 details precision and recall by category.

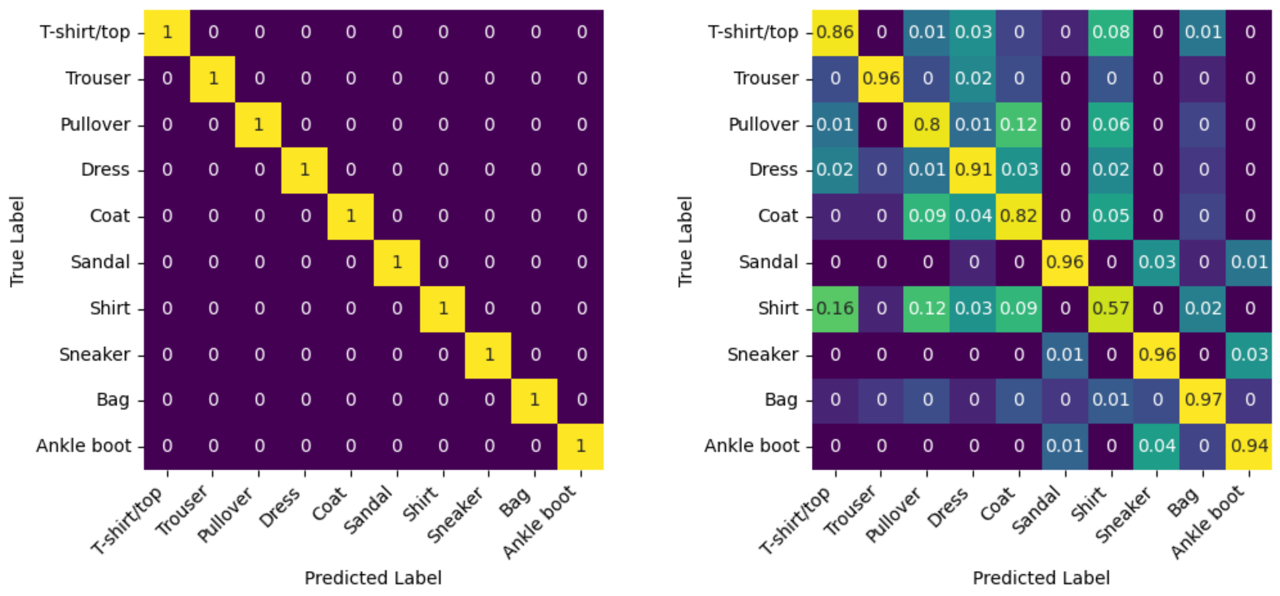
Table 3: Per-Category Precision and Recall

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%

6 Analysis and Discussion

6.1 Confusion Matrices

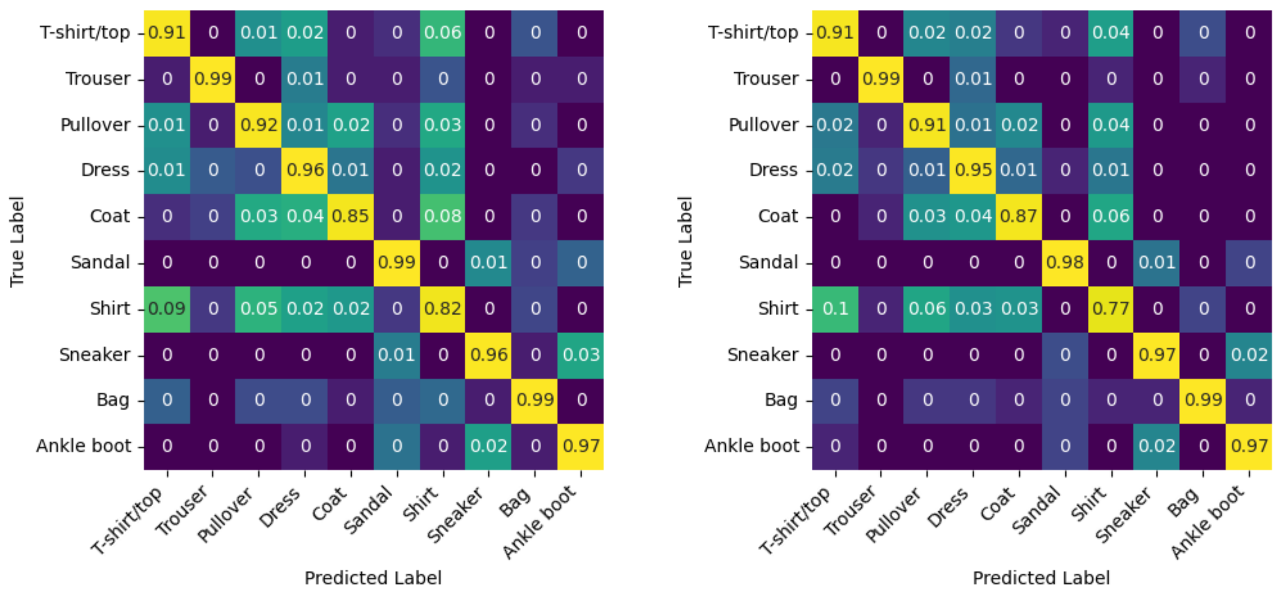
Figures 5 and 6 present confusion matrices for both classifiers on train and test sets.



(a) Training set (100% accuracy)

(b) Test set (87.52% accuracy)

Figure 5: Random Forest: Perfect training performance but significant confusion between similar upper-body garments on test set.



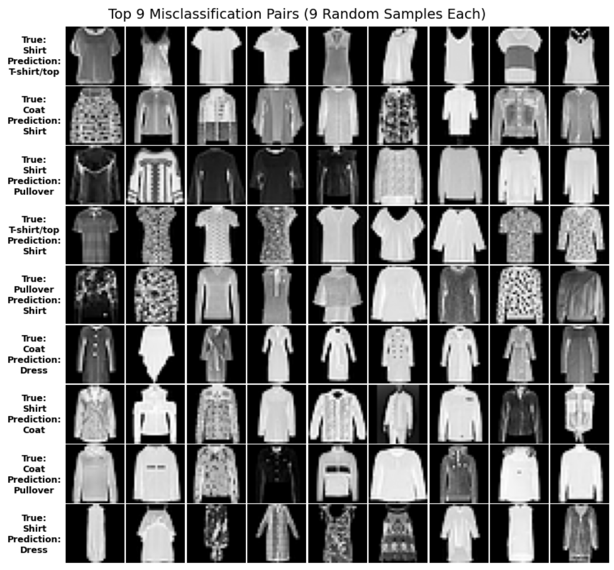
(a) Training (95.47%)

(b) Test (93.63%)

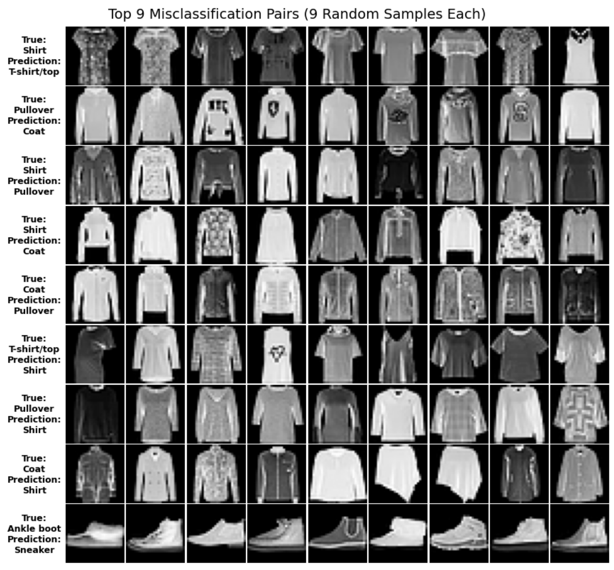
Figure 6: CNN: Consistent train/test performance with reduced confusion between similar categories.

6.2 Misclassification Analysis

Figure 7 displays representative misclassified samples for both classifiers, revealing specific patterns of confusion between visually similar categories.


Top 9 Misclassification Pairs (9 Random Samples Each)
True: Shirt Prediction: T-shirt/top
True: Coat Prediction: Shirt
True: Shirt Prediction: Pullover
True: T-shirt/top Prediction: Shirt
True: Pullover Prediction: Shirt
True: Coat Prediction: Dress
True: Shirt Prediction: Coat
True: Coat Prediction: Pullover
True: Shirt Prediction: Dress

(a) CNN: Top confusion pairs showing shirts misclassified as T-shirts/tops, coats as shirts, and shirts as pullovers.


Top 9 Misclassification Pairs (9 Random Samples Each)
True: Shirt Prediction: T-shirt/top
True: Pullover Prediction: Coat
True: Shirt Prediction: Pullover
True: Shirt Prediction: Coat
True: Coat Prediction: Pullover
True: T-shirt/top Prediction: Shirt
True: Pullover Prediction: Shirt
True: Coat Prediction: Shirt
True: Ankle boot Prediction: Sneaker

(b) Random Forest: Similar patterns but more severe confusion, plus footwear errors (boots vs sneakers).

Figure 7: Misclassification examples showing both models struggle with upper-body garments, but CNN errors are more visually justifiable.

Both classifiers struggle most with upper-body garments, particularly shirts versus T-shirts/tops and coats versus pullovers. The CNN's misclassifications appear more visually justifiable, while Random Forest exhibits more erratic confusion including footwear errors. The CNN's ability to capture fine-grained features like collar shape and fabric texture explains its 6.11 percentage point accuracy advantage.

6.3 Category-Specific Analysis

Best Performing Categories: Trouser ($>$98% precision for both), Bag ($>$95%), and footwear categories (Sneaker/Sandal, $>$95%) benefit from distinctive shapes and non-clothing features.

Challenging Categories: Shirt classification presents the greatest difficulty (72.4% RF, 82.0% CNN test precision) due to visual similarity with T-shirts/pullovers, high intra-class variability, grayscale limitations, and low resolution constraining fine details. Pullover and Coat show moderate challenges (76-91% precision) from overlapping outerwear features.

The CNN demonstrates superior performance on all challenging categories with notable improvements: Pullover (+13.9%), Coat (+11.4%), and Shirt (+9.6% precision; +20.1% recall). The CNN's hierarchical features better capture subtle distinctions in collar shape and fabric texture, explaining these gains despite similar visual ambiguity.

6.4 Production Recommendation

The CNN is recommended for production based on: (1) 6.11 percentage point accuracy improvement (611 fewer errors per 10,000 items), (2) superior handling of challenging categories, (3) better generalization (1.84% vs 12.48% train-test gap), and (4) acceptable 900s training time for infrequent retraining. The CNN model size (10MB) is smaller than Random Forest (123MB), though inference is slower (2.03s vs 0.07s). CNN suits batch processing; Random Forest suits latency-sensitive applications and rapid prototyping.

7 Conclusion

This study demonstrates clear advantages for CNN-based approaches in fashion image classification. MCNN15 achieved 93.63% test accuracy versus 87.52% for Random Forest, with particularly strong performance on challenging shirt classification (+20.1% recall improvement). The CNN's hierarchical feature learning substantially reduces inter-class confusion, justifying its use in production despite longer training time (900s vs 9.45s).

Future Work: Potential extensions include evaluating transfer learning from ImageNet-pretrained models, testing ensemble CNN-RF hybrid approaches, and analyzing performance on higher-resolution fashion datasets. Additionally, investigating attention mechanisms and lightweight architectures could improve both accuracy and inference efficiency for real-time applications.

References

Bbouzidi, S., Hcini, G., Jdey, I., & Drira, F. (2024). Convolutional neural networks and vision transformers for fashion mnist classification: A literature review. *arXiv preprint arXiv:2406.03478*. https://arxiv.org/abs/2406.03478

Bhatnagar, S., Ghosal, D., & Kolekar, M. H. (2017). Image classification using multiple convolutional neural networks on the Fashion-MNIST dataset. *2017 International Conference on Intelligent Sustainable Systems (ICISS)*, 597–602. https://doi.org/10.1109/ISS1.2017.8389294

Cavallo, F., et al. (2022). Image classification using multiple convolutional neural networks for clothing retrieval. *Sensors*, *22*(23), 9544. https://doi.org/10.3390/s22239544

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on Machine Learning*, *37*, 448–456.

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv: 1412.6980. https://arxiv.org/abs/1412.6980

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, *25*, 1097–1105.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Mukhamediev, R. I. (2024). State-of-the-art results with the fashion-mnist dataset. *Mathematics*, *12*(20), 3174. https://doi.org/10.3390/math12203174

Sathyadevan, D., & Gangadharan, S. (2015). Crime analysis and prediction using data mining. *2015 First International Conference on Networks & Soft Computing*, 406–412.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. https://arxiv.org/abs/1409.1556

Xiao, H., Rasul, K., & Vollgraf, R. (2017). *Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms*. arXiv: cs.LG/1708.07747. https://arxiv.org/abs/1708.07747

A Hardware Configuration

All training experiments were conducted on a workstation with an Intel Core i7-14700 processor, NVIDIA GeForce RTX 4080 SUPER GPU (16GB VRAM), and 32GB system RAM. Random Forest training utilized CPU parallelization across all available cores. CNN training was performed entirely on GPU. Inference benchmarks were additionally evaluated on a laptop with Intel Core i5-13420H and NVIDIA GeForce RTX 4060 GPU to assess deployment feasibility across hardware tiers.

B Code Implementation

Note: The complete implementation including all source code, training scripts, evaluation code, and generated plots is available in a GitHub repository located at https://github.com/jst-r/fmnist-proejct and in the attached archive.

B.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)
```

B.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()
        )
```