IT UNIVERSITY OF CPH

Clothing Classification: Exploring Different Methods to Classify Clothing Types

Sergio Bueso

serb@itu.dk

Martin Jørgensen
mjoer@itu.dk

Jan Vivo
jviv@itu.dk

January 6th, 2025

Abstract

This report explores the application of feedforward neural networks (FNN), decision trees, and other methods for classifying clothing images. The study includes data visualization, exploratory analysis, and classification results. Using a balanced dataset of grayscale clothing images, the goal is to compare the performance of these classifiers, emphasizing the trade-offs between complexity and interpretability. The analysis concludes with a discussion on the strengths and limitations of each method and suggests scenarios for their optimal application.

Contents

Abstract **ii**

1 Introduction **iv**

2 Exploratory Data Analysis **iv**
2.1 The Dataset iv

3 Dimensionality Reduction **iv**
3.1 Implementation iv
3.2 Visualization v

4 Classifiers **vi**
4.1 Decision Tree vi
4.1.1 Overview vi
4.1.2 Implementation Details vi
4.1.3 Results and Performance Evaluation vii
4.2 Feed-Forward Neural Network vii
4.2.1 From scratch implementation vii
4.2.2 Tensorflow implementation viii
4.2.3 Results and Evaluation viii
4.3 Convolutional Neural Network ix
4.3.1 Implementation ix
4.3.2 Results ix

5 Interpretation and Discussion **x**

1 Introduction

This study focuses on classifying clothing items using machine learning techniques, specifically feedforward neural networks, decision trees, and other methods. The goal is to develop efficient methods for recognizing types of clothing items from grayscale images by exploring data visualization, dimensionality reduction, and classification strategies.

The dataset, sourced from Zalando's Fashion-MNIST dataset [1], reflects practical challenges in image classification tasks. The dataset comprises 28×28 grayscale images of clothing items across five categories: t-shirts, trousers, pullovers, dresses, and shirts. With 15,000 samples in total, it provides a balanced distribution for training and testing.

The project's motivation lies in leveraging machine learning to tackle real-world datasets, providing insights into how these algorithms perform in practical applications. By investigating three contrasting approaches, we aim to highlight the strengths and weaknesses of both models, offering a comprehensive understanding of their suitability for various scenarios.

2 Exploratory Data Analysis

2.1 The Dataset

The dataset consists of 15,000 labeled grayscale images (28x28 pixels) divided into five clothing categories: T-shirt/top (0), trousers (1), pullover (4), dress (3), and shirt (2). The images are stored in NPY format. The dataset includes 10,000 training samples and 5,000 test samples. It is balanced, meaning that both the training data and the test data contain around the same amount of each respective label (the test data consists exactly of 1000 images of each label).

Figure 1: Sample grayscale images from the dataset, illustrating different clothing categories.

3 Dimensionality Reduction

In the high-dimensional feature space of the dataset, not all features contribute equally to the predictive power of the model. Some features may be redundant or noisy. To account for this large dimensionality, a Principal Component Analysis (PCA) was carried out. PCA identifies new axes, called principal components, which are linear combinations of the original features and capture maximum variance. This unsupervised method identifies and retains only the most significant components, thus reducing the influence of noise, improving the robustness and generalization of the model.

3.1 Implementation

We began by separating the feature set and labels from the dataset, organizing the features into a DataFrame for preprocessing. We then used the sklearn decomposition module after scaling the data for equal variance. The first principal component captures the direction of maximum

variance within the data, often representing the most dominant patterns or prominent features in an image. In image analysis, this component typically highlights global patterns or primary characteristics. The second principal component, being orthogonal to the first, accounts for the next largest source of variance and is uncorrelated with the first component.

3.2 Visualization

Below is a visualisation of these 2 first Principal Components plotted against each other, with hues representing each of the 5 different labels. Although the scaled 2-dimension reduction only represents 36.3% of the original data's variation, somewhat of a distinction can already be seen between the classes already.

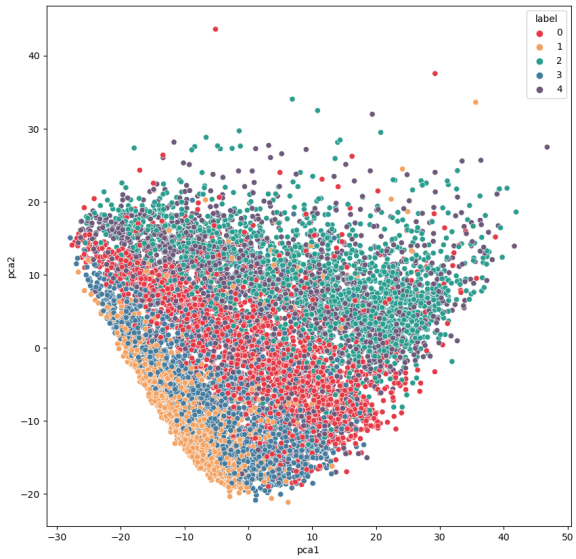
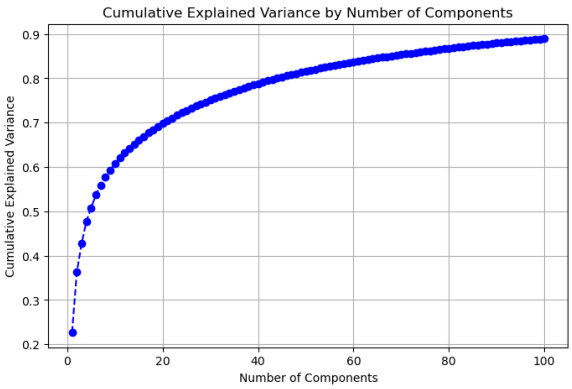


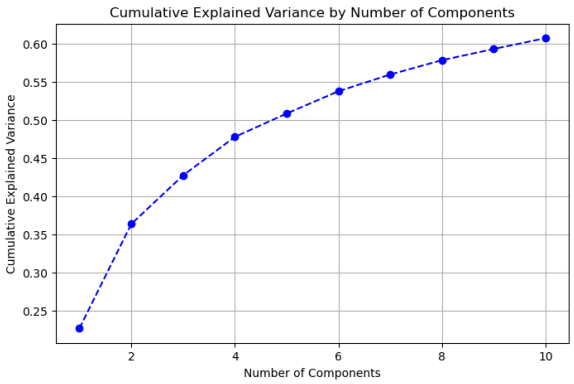
Figure 2: 2D Plot of the First Two Principal Components

After plotting the cumulative explained variance of the first 100 principal components (Figure 3(a)), we can see that around 90% of the information contained through the variance is explained. However, the slope of the curve decreases as more components are added, meaning each additional principal component contributes progressively less to explaining the variance. This is a key insight when deciding how many components to retain.

One can more or less identify the "elbow" or the point where the cumulative explained variance begins to level off. This point is often used to choose the number of components that balance variance explanation and dimensionality reduction. In this case, the 10th principal component is a sensible estimation, and data shows us that the latter explains 60.7% of the information contained through the variance (Figure 3(b)). As the reader will see later, this 10 feature dataset wields the best result on classifiers compared to other amounts of principal components



(a) First 100 Components



(b) First 10 Components

Figure 3: Cumulative Explained Variance of Principal Components

4 Classifiers

4.1 Decision Tree

4.1.1 Overview

Decision trees are supervised learning models that partition data based on feature values, forming a tree-like structure. Each internal node of the tree represents a decision based on a threshold value of a particular feature, while the branches represent the outcomes of these decisions. The algorithm selects splits at each node that minimize impurity, using measures such as the Gini index or entropy. This process is repeated recursively until a stopping criterion is met, such as a maximum tree depth or a minimum number of samples per leaf. The result is a decision tree that can be used for classification by following the paths from the root to the leaf nodes.

4.1.2 Implementation Details

From scratch implementation The custom decision tree classifier was implemented following a step-by-step approach. First, the dataset was divided into training and test sets to allow for model evaluation. The tree construction followed a recursive process: At each node, the algorithm calculated the Gini impurity for potential splits based on different features. The feature and threshold that minimized the weighted average impurity of the child nodes were selected. This splitting process continued recursively until a stopping condition, such as a maximum tree depth or a minimum number of samples per leaf, was reached. When making predictions, the model classified test samples by traversing the tree from the root to a leaf node. The class prediction was based on the majority class of the leaf node. The key hyperparameters used for the custom implementation included:

- Maximum tree depth: 9 (to avoid overfitting).
- Minimum samples per split: 2

To acquire the best accuracy while preserving underfitting, overfitting and run-time we chose the maximum tree depth to be 9 and and the minimum samples per split to be 2. The accuracy starts decreasing once the tree depth goes above 9 which indicates overfitting.

Reference implementation For comparison, the reference implementation used scikit-learn's DecisionTreeClassifier. This model was configured with the Gini impurity criterion and a max-

imum depth of 10. The reference implementation served as a benchmark to validate the correctness of the custom implementation.

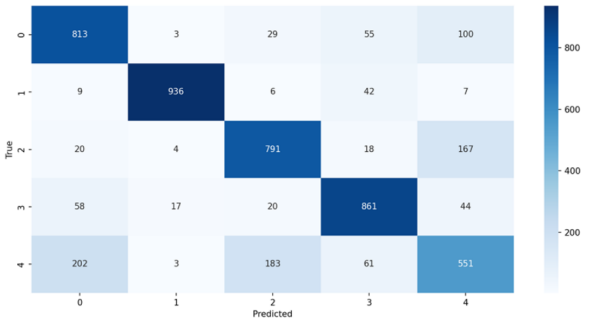
4.1.3 Results and Performance Evaluation

Training and Test Accuracy The table below summarizes the performance of both the custom (from scratch) and reference (scikit-learn) decision tree implementations:

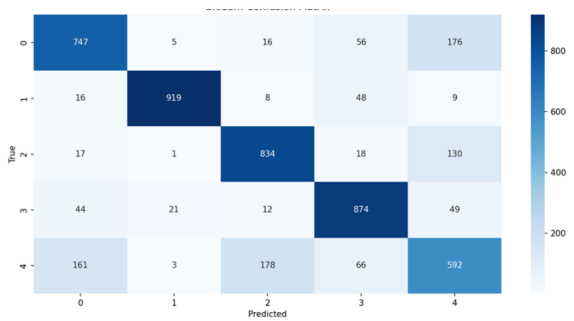
implementation	training accuracy	test accuracy
from scratch	0.9	0.79
Scikit-learn (Reference)	0.9	0.8

Both implementations achieved similar good results. This performance indicated that the model was effective, but also highlighted signs of slight overfitting, as the training accuracy was higher than the test accuracy. The from scratch implementation does however, have a longer runtime than the scikit-learn model.

Confusion matrix The confusion matrix was used to further evaluate the classifier's performance by identifying which classes were frequently misclassified. This analysis provided valuable insights into the model's weaknesses, showing specific areas where the model struggled to classify certain samples.



(a) Confusion matrix for the from scratch decision tree implementation



(b) Confusion matrix for the SKlearn decision tree implementation

Figure 4: Comparison of confusion matrices for decision tree implementations

4.2 Feed-Forward Neural Network

4.2.1 From scratch implementation

Weights and biases are initialized for the network. Random values are assigned to the weights using a technique to scale them based on the number of neurons in the previous layer. Biases are initialized to zeros. These parameters define the connections between neurons and are crucial for learning.

The forward pass propagates input data through the network. In the layers, the weighted sum of inputs is computed, and an activation function is applied. The ReLU function is used in the hidden layer to introduce non-linearity, while the softmax function is used in the output layer to convert raw scores into probabilities. After this, a loss function calculates the difference between the predicted outputs and the true labels. Categorical cross-entropy is used, which measures the divergence between the predicted probability distribution and the true distribution.

Backpropagation is implemented to compute gradients of the loss with respect to the weights and biases in the backpropagation() function. This involves propagating the error backward

through the network using the chain rule of calculus. The weights and biases are updated using gradient descent in the direction that minimizes the loss. The established learning rate controls the size of these adjustments. This update is performed in the update_parameters() function.

The network is then trained over multiple epochs. In each epoch, the forward pass, loss computation, backpropagation, and parameter updates are repeated. The loss is printed at intervals to monitor the training progress. Once training is complete, the model is evaluated on a separate test set. Predictions are made by passing test data through the network in a forward pass. The predicted class labels are of course compared with the true labels to calculate the accuracy.

4.2.2 Tensorflow implementation

This neural network, implemented using TensorFlow and Keras, is a simple feedforward architecture designed for multi-class classification. The network takes input data, which is scaled to normalize its range, and processes it through a single hidden layer with 126 neurons. The hidden layer applies a ReLU activation function to introduce non-linearity. All this just like the implementation from scratch. However, and as one of the main differences, the model is trained using the Adam optimizer, which adaptively adjusts learning rates for efficient and effective optimization.

4.2.3 Results and Evaluation

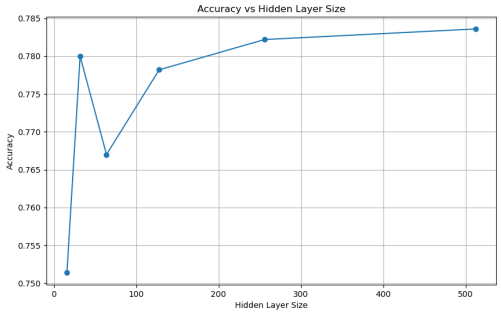
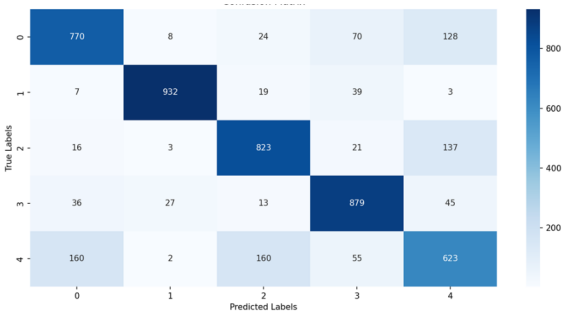


Figure 5: Plotting different hidden layer sizes

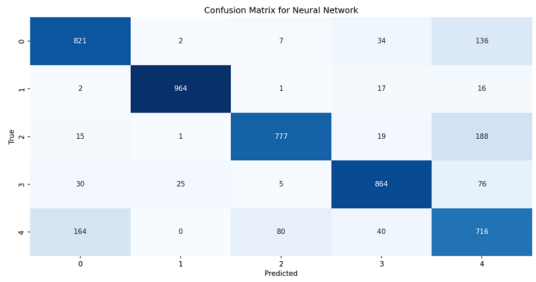
After subsequently testing with 6 different hidden layer sizes (16, 32, 64, 128, 256, 512), we found that except for size 16, the test accuracy fluctuated around a 1% difference for the rest. Therefore, to optimize computational complexity our final model contained a hidden layer of size 126. Below are the test accuracies obtained from both of the implementations, with a 5% accuracy difference between the models, as well as the confusion matrices for the test implementations.

Method	**Test Accuracy**
From Scratch	77.92%
Reference	83.14%

Table 1: Comparison of Test Accuracy Between Methods



(a) Confusion matrix for the from scratch FFNN



(b) Confusion matrix for TF implementation of FFNN

Figure 6: Comparison of confusion matrices for FFNN implementations

4.3 Convolutional Neural Network

For our last implementation, we choose to do a CNN (Convolutional Neural Network). CNNs try to find patterns in the input data. They work in a way similar to human brains. This CNN is implemented using TensorFlow and Keras. We thought a CNN would be a good fit because CNNs excel in tasks like clothes classification because they are tailored to analyze images by learning spatial hierarchies, being computationally efficient, and generalizing well across diverse and complex datasets.

4.3.1 Implementation

It consists of several layers designed to extract features from input images and perform multi-class classification. First, the images are run through what we call the convolution layers. These layers are the feature extraction ones. They will identify unique and distinctive features of the clothes, such as shapes or edges. The first layer, a convd2 layer with 32 filters and kernel size (3, 3) applies a ReLU activation function, enabling feature detection. The ReLu activation outputs the input directly if it's positive or otherwise puts a zero. This is very useful because it maintains the positive values unchanged, not letting the ones that not matter interfere.

We then have a pooling layer. Pooling layers make the image smaller. This helps with computational power and overfitting. In our case we used a Maxpooling layer. This splits the image in 2x2 blocks and extracts the single most valuable value from each. This ends up containing the same information regarding extracted features from the original image, but smaller.

The images are run by another cycle of convolution/ pooling before being sent in to the dense layers.

This dense layer, with 128 different neurons identify the patterns in the features and classifies each image with the corresponded label. It is just the classification part. We also applied a dropout layer. A Dropout layer with a 50% rate mitigates overfitting by randomly deactivating half of the neurons during training.

4.3.2 Results

The results show that the model achieved a strong performance on the validation and test datasets, with a test accuracy of 88.04%. The network converged well, showing improvements in both accuracy and loss across epochs.

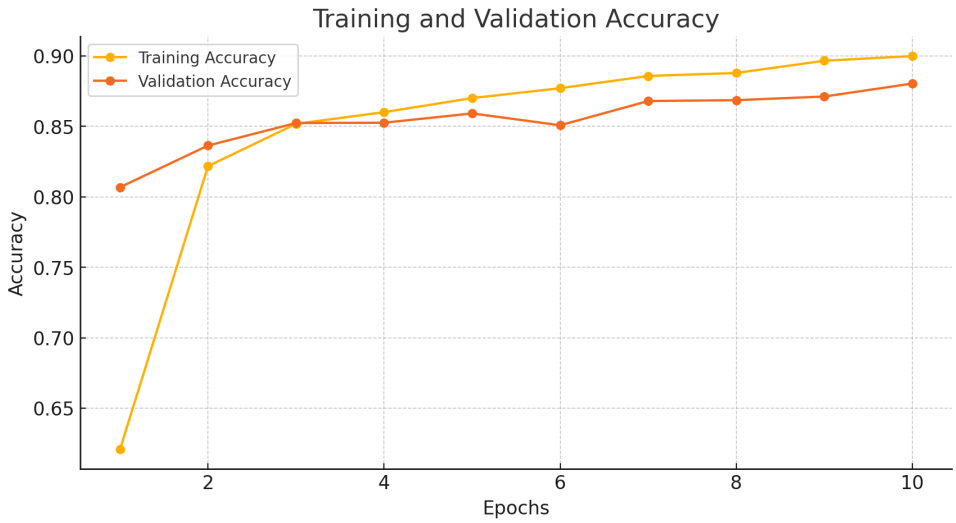


Figure 7: Plot of the results

5 Interpretation and Discussion

After thoroughly reviewing the three classification models, we compile all observations and results to evaluate and determine which model is best suited for the classification task of predicting clothing types using real data.

Model	**Original Dataset**	**PCA Data (10 P.C)**
Decision Tree	79%	76%
Decision Tree with Library	80%	76%
FFNN (Feedforward Neural Network)	77.92%	58.5%
FFNN with Library	83.14%	77.76%
CNN (Convolutional Neural Network)	88.04%	–

Table 2: Accuracy Comparison of Models on Original Dataset and PCA Data

Firstly, as seen in Table 2, none of the models benefitted from using a dataset of 10 principal components (or, as we also found, even a larger number of PCs). This suggests that the principal components, while capturing variance, may not effectively represent the specific patterns or relationships needed for accurate classification in this case. Decision trees rely on clear, interpretable features for splits, while FFNNs require rich and detailed inputs to learn complex relationships. PCA may have stripped away critical information, regardless of the number of components used, leading to reduced model performance.

The decision tree implementation is effective but has limitations. Overfitting is evident, with higher training accuracy than test accuracy, which could be mitigated with pruning or better hyperparameter tuning. The custom implementation is slower than scikit-learn, requiring optimization through vectorization or parallel processing. Overreliance on accuracy as a metric misses deeper insights—adding precision, recall, and F1-score would improve evaluation.

Lastly, the confusion matrices for both the custom (from scratch) and scikit-learn implementations showed weaknesses in correctly identifying Class 4 (Shirt). Both models confused Class 4 (Shirt) with Class 0 (T-shirt/top). This misclassification makes sense, as these classes share visual similarities, particularly in terms of shape and design, and it is very probable that the tree's leaves pools their features as if they were one

With regards to the Feed-Forward Neural Netowrk, The library implementation did not provide a very large improvement in performance, as there seems to be a limit on its ability to classify these images. This could be due to the limitations that a 1D neural network like this has on clothes detection. Since the input images are flattened into 1D arrays, the spatial relationships between pixels are destroyed. Nearby pixels that form edges, textures, or patterns in clothing are treated as independent, losing important contextual information contained in clothing, such as sleeves, collars or embroideries.

However, the CNN did seem to pick up more of these spatial relationships, as it is specifically designed to process 2D data like images. By utilizing convolutional layers, the CNN preserved the spatial structure of the input, enabling it to detect patterns such as edges, textures, and shapes that are critical for distinguishing different types of clothing. Consequently, the CNN outperformed the Feed-Forward Neural Network, demonstrating its suitability for the task of clothing type classification.

References

References

[1] Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms. Retrieved from `https://github.com/zalandoresearch/fashion-mnist`

[2] GeeksforGeeks. (n.d.). *Introduction to Convolutional Neural Networks*. Retrieved January 6, 2025, from `https://www.geeksforgeeks.org/introduction-convolution-neural-network/`

[3] JavaTpoint. (n.d.). *Machine Learning - Decision Tree Classification Algorithm*. Retrieved January 6, 2025, from `https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm`

[4] Harvard IACS. (2020). *CS109A Introduction to Data Science - Section 9*. Retrieved January 6, 2025, from `https://harvard-iacs.github.io/2020-CS109A/sections/sec_9/`