Machine Learning Project

Course code: BSMALEA1KU

Giulia Xixi Curtolo, Pavel Dohnal, Patricia Nita
gicu@itu.dk pavd@itu.dk patn@itu.dk

1. Overview

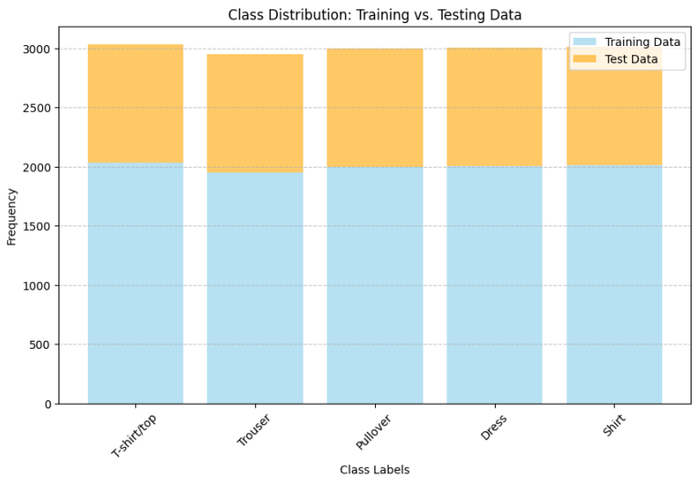
This project aims to explore classification methods on clothing image data. This includes both original and built-in library implementations. The data used is labelled and split into 10000 training and 5000 test samples stored as 784 varying pixel values per entry (ranging from 0-255 in value).

To better understand the dataset and prepare for classification, we begin with an exploratory analysis of the clothing images.

2. Exploratory Data Analysis

The samples are approximately evenly distributed among 5 classes, averaging 2000 and 1000 for training and test sets

Figure 1: Sample Distribution across classes



Each image below represents the average of the 5 different image classes as follows: 0 – t-shirt / top; 1 – pants; 2 – pullover; 3 – dress; 4 – shirt. There are noticeable visual similarities between some of the labels, i.e. shirts and T-shirts look very similar, which could make the classification for these specific classes more challenging; in contrast, classes 1 and 3 (pants and dresses) have more distinct shapes, making them easier to identify. This observation suggests the model may have more difficulty with certain classes than others.

Figure 2: Means of Classes

Building on the initial analysis, we apply PCA to further explore the data and reduce dimensionality, allowing us to capture key characteristics of the classes with a lower feature count.

2.1 Dimensionality reduction

It is evident that at least given the white background in all images, the dimensionality of the sample can be reduced significantly without dramatic losses to the accuracy of classification.

Principal Component Analysis (PCA) is a method that can be applied to the dataset to lower the number of features that represent the samples. PCA allows to select axis that are aligned with the direction that is obtained by maximizing the variance of the data.

PCA is a dimensionality reduction method to represent maximum information of a dataset using fewer features. For instance, many dimensions of features describing the white image background can be generalized without losses in accuracy of describing the overall image.

2.2 Scale of the features

It is generally advised to standardize the scale of the features when working with PCA as this method is sensitive to the scale of the features.

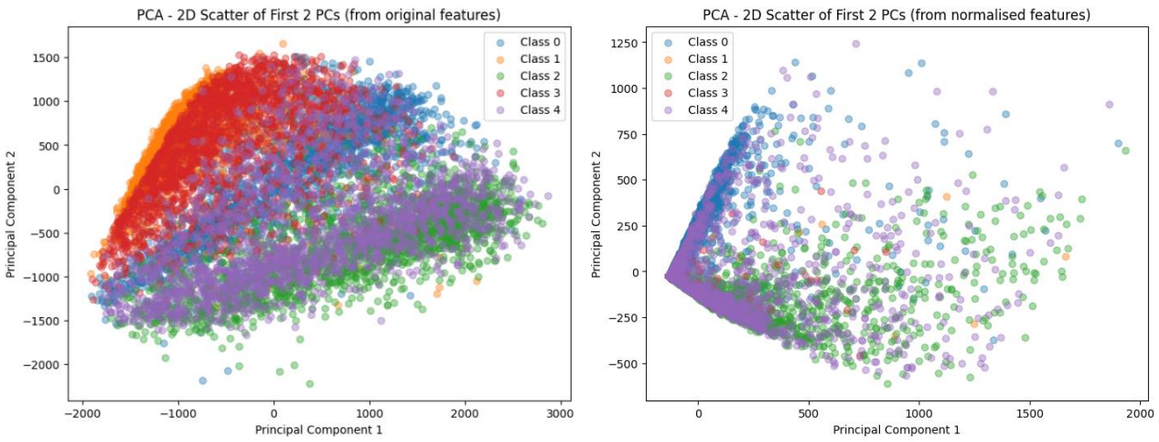
We have considered PCA with both standardized and unstandardized scales of the features. Simple comparison of the cumulative explained variance with the same number of principal components shows that indeed, with this specific dataset it is beneficial to standardize the scales of the input variables.

Figure 3: Variance Explained, unscaled vs scaled data

```
Explained Variance Ratio (PC1, PC2) of original data: [0.32286567 0.16258462]
Explained Variance Ratio (PC1, PC2) of scaled data: [0.40317348 0.17384003]
```

However, for the purposes of visualization, the first 2 principal components of the original features provide a more intuitive interpretation of the division between classes of the grouped samples (as a result of the data scaling)

Graph 1: Scatterplots of 2 first principal components: unscaled data (left) and scaled data (right)

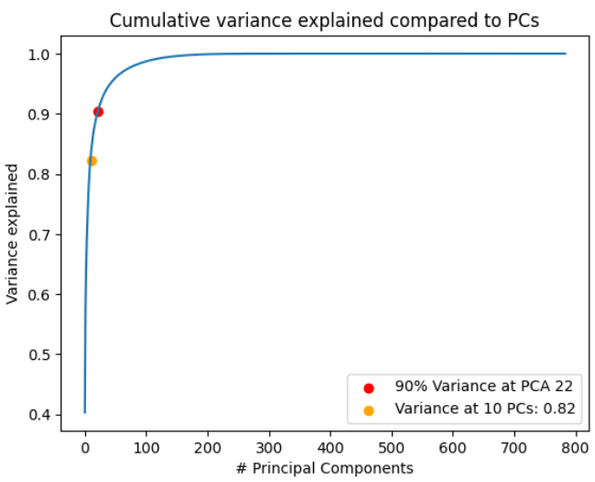


2.3 Amount of Principal Components

The amount of Principal Components is chosen by “eyeballing the scree graph” (James et al., 2023). What one should be looking for is the point at which the contribution of a new principal component, or “the elbow” in the scree plot (see **Eroare! Fără sursă de referință.**).

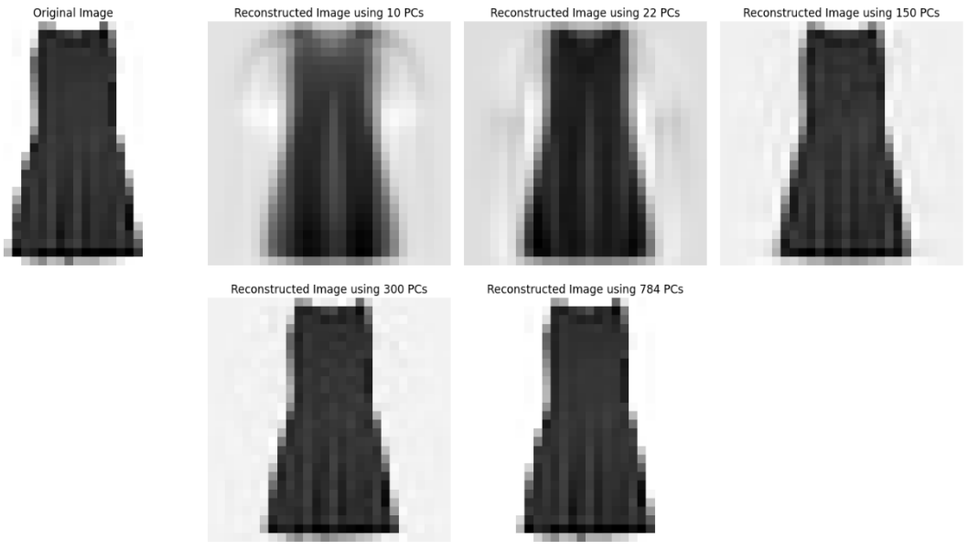
A threshold of 0.9 was chosen as an acceptable value of Cumulative Variance Explained. For all purposes in this work, 22, the minimum amount of principal components that reach this value, will be used.

Graph 2: Cumulative variance explained by principal components (min-max scaled)



Alternatively, one may define an absolute threshold for the contribution of a singular principal component or a percentual decrease in contribution to explained variance of a new principal component in relation to the previous one.

Figure 4: Reconstruction of a sample image using various counts of principal components



Now that we've reduced the data dimensions using PCA, we are ready to apply some machine learning methods for classification and explain how each model works with the dataset.

3. Decision Trees (M1)

The first method we'll be exploring are decision trees, a popular supervised learning algorithm composed of nodes in a tree-like structure that make separations to data based on decision boundaries formed on a feature.

Decision trees are based on a relatively simple, but scalable principle. For each value of each feature in the training dataset, an impurity of a split based on such a value is considered. When all possible splits have been considered, the split with the lowest impurity is chosen (Scikit-learn, 2024).

Our decision tree implementation uses a node-based recursive algorithm for training. The tree is initialised as an object with a node subclass:

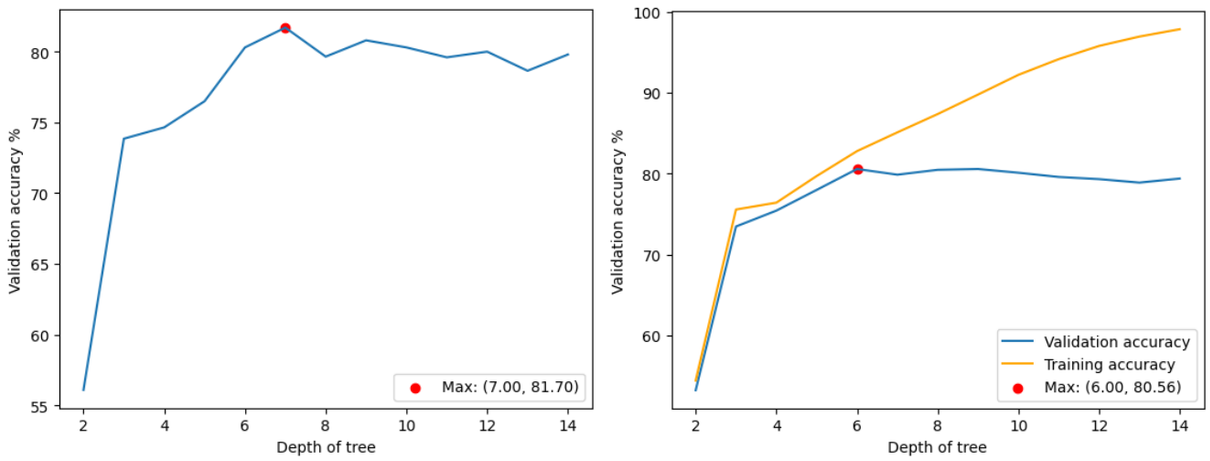
```
class tree:

    class node:
        def __init__(self,  left = None, right = None, feature = None,
threshold = None, prediction = None):
            self.left = left
            self.right = right
            self.feature = feature
            self.threshold = threshold
            self.prediction = prediction


    def __init__(self, root= None):
        self.root = root
```

3.1 Bias-Variance Trade-off (and Cross-validation)

In training decision trees, boundaries are created in relation to the trends observed in the specific training set. As the depth of the tree increased / training samples are split into smaller groups, these boundaries reflect fewer abstract patterns in the dataset and instead are based on smaller inconsistent nuances prone to small changes in training data, exhibiting high variance.

To define the optimal tree depth, we then have to avoid overfitting on training data when obtaining performance comparison metrics, resolved by adapting k-fold cross validation. Since we don't have access to multiple training sets, we aggregate accuracy scores from randomly sampled groups of the training dataset, which repeatedly verify the strength of a tree at that depth. We choose a 9:1 split of training data to evaluate as closely to the final tree classifier as possible.

Figure 5: Cross-Validation Visualized



3.2 Impurity Measures: Gini vs Cross Entropy

$$\text{Gini} = 1 - \sum_{i=1}^{n} (p_i)^2 \;\; vs \; Binary \; Entropy = -\sum_{x \epsilon X} p \log p$$

The implementation of the decision tree-building algorithm allows the user to choose a specific impurity measure, based on which the defining feature and its value for the split are chosen.

```
@staticmethod
    def gini_impurity(label_counts, total_count):
        if total_count == 0:
            return 0
        label_counts = np.array(label_counts)
        proportions = label_counts / total_count
        return 1 - np.sum(proportions ** 2)

    @staticmethod
    def binary_entropy(label_counts, total_count):
        if total_count == 0:
            return 0
        label_counts = np.array(label_counts)
        proportions = label_counts / total_count
        # Avoid log(0) by masking zero proportions
        valid_proportions = proportions[proportions > 0]
        return - np.sum(valid_proportions * np.log(valid_proportions))
```

3.3 Termination Conditions

Aside from the predetermined tree depth, we considered other conditions that would be used to build a leaf node. One of the ways we used to determine that a newly built node would be assigned a class prediction value instead of a split, is checking, whether the samples of the training dataset that would be yielded by the split of its parent node pertain to a single class. This is demonstrated in the snippet below.

```
            if X_left.size > 0 and y_left.size > 0:
                node.left = self.fit(X_left, y_left, max_depth, depth=depth+1)
            else:
                p_class = np.bincount(y_right.astype(int)).argmax()
                return self.node(prediction=p_class)
            if X_right.size > 0 and y_right.size > 0:
                node.right = self.fit(X_right, y_right, max_depth,
depth=depth+1)
            else:
                p_class = np.bincount(y_left.astype(int)).argmax()
                return self.node(prediction=p_class)
```

3.4 Implementation Validity Assessment

The decision tree implementation was tested against the scikitlearn implementation. The resulting splits and classification outcomes matched those of the scikit-learn implementation using identical input parameters and data. Based on this comparison, we conclude that we were able to successfully implement a decision tree algorithm.

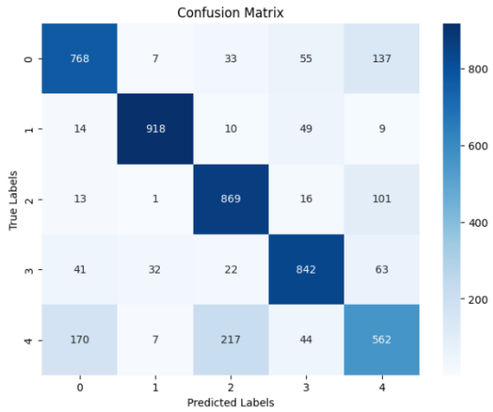


Figure 6: Confusion Matrix, Decision Tree Classifications

As for the performance of the fitting function, as a result of experimentation, it is clear that our algorithm is slower than scikitlearn's decision tree fitting. We attribute this performance difference to scikitlearn's use of optimized Cython code, whereas our implementation relies solely on Python and NumPy for acceleration. Further optimization goes beyond the scope of this work.

4. Feed-Forward Neural Network (M2)

4.1 Activation Functions

We chose to use ReLU activation function for the hidden layers which is simpler to compute for forward/backward passes than sigmoid as it works only with a comparison statement.

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha \cdot x & \text{if } x \leq 0 \end{cases} \qquad \frac{d}{dx}\text{ReLU}(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{if } x \leq 0 \end{cases}$$

Additionally, due to the short range of input which provides gradient outputs far from 0, backward passes with sigmoid activation will have almost no update on weights in training, which with multiplication through multiple hidden layers emphasizes the vanishing gradient problem. (Thakur, 2024) By setting negative values close to 0, ReLU is also good at making the model more efficient by increasing network sparsity.

$$\text{Softmax}(x_i) = \frac{e^{x_i - \max(\mathbf{X})}}{\sum_{j=1}^{n} e^{x_j - \max(\mathbf{X})}}$$

SoftMax was used for the multi-class classification on the output layer with one-hot encoding representation of our labels where the loss minimized would be through summed class probabilities subtracted from binary class labels.

4.2 Loss function

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij} + \epsilon)$$

We applied cross-entropy loss which strongly penalizes misclassification of probabilities close to 0 for true class labels (1). It is also less likely to reach extreme values and encourage confident (high probability) accurate predictions. (Sorokin, 2024)

4.3 Forward pass and backpropagation (and batch normalisation)

To get predictions and calculate new gradients for weights and biases, we set up matrices defined by the number of training samples and the number of neurons for each layer. The initial weights are randomly sampled from standard normal distributions and then summed over the input-weights products for each layer.

Before passing weighted inputs into the activation function, the model first applies batch normalisation to stabilise learning and avoid passing extreme values into backpropagation step which would in turn lead to more dramatic derivatives and an exploding gradient. This ensures each layer's activations remain in a consistent range of values centred around 0 with unit variance, making it easier for each layer to obtain information compared to potentially skewed previous distributions of activations. (Dutta, 2024)

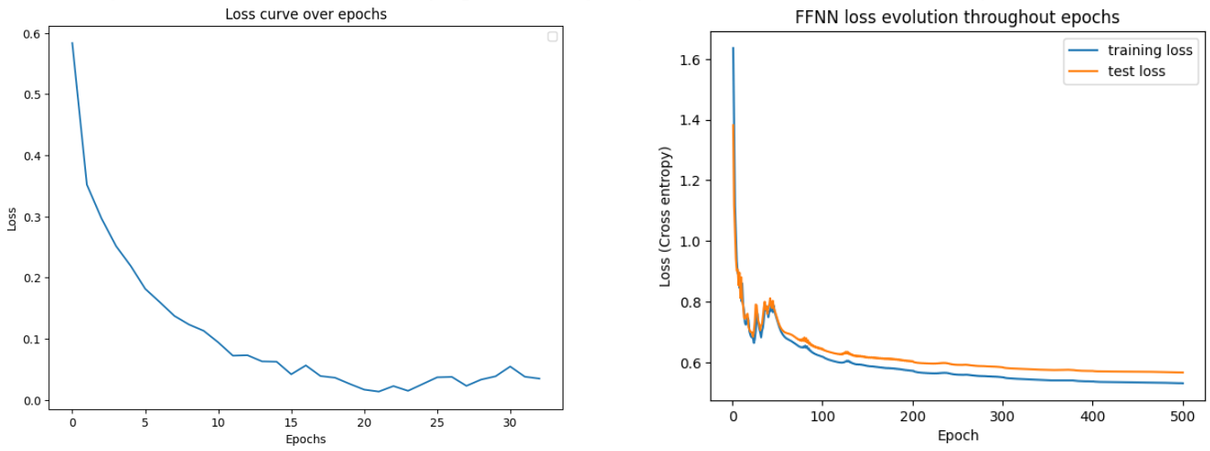
4.4 Number of layers

Our model was implemented with 3 hidden layers to balance between capturing the complexity of a 784-pixel image which might be lost in shallower networks, and not overfitting / exceeding time capacity in training given the size of our dataset.

The layers consisting of 64, 48, and 32 neurons slowly generalise low-level features to high-level features with enough dimensionality reduction to train more epochs, without missing important information in the first layers.

4.5 Number of epochs and training rate

Graph 3: Loss Curves, FFNN (Right: Library implementation, Left: Own implementation



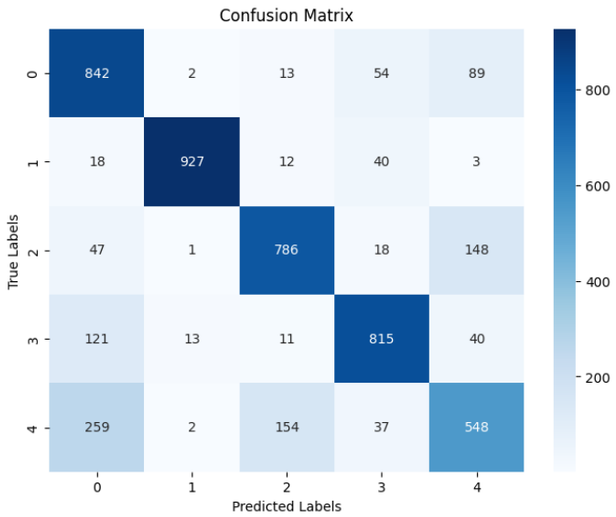
When first training our model, we attempted a higher number of neurons per layer, with approximately 256, 128, and 64 in order. While this allowed us to gain an accuracy rate of 84% close to the library implemented model's accuracy of 85%, the training loss history showed strange fluctuations which indicate that the model's predictions might not be very stable and thus not generalize well (make riskier predictions) to an unseen dataset outside of out training and test sets.

We attempted to mitigate this by reducing our learning rate, originally 0.0001, however the model performed poorly, and the unstable loss history preserved. Instead, by significantly reducing the number of neurons per layer, this resulted in a much smoother loss history curve that showed convergence at 700 epochs. To verify that this amount of iterations will not make the model parameters overfit to the training set, we can compare it to the test loss, which at 700 epochs has a negligible difference to the training loss but diverges with more iterations. On the other hand, with a higher

learning rate, the loss evolution is more unstable, so we decided to remain with 0.0001=a, gradually converging to the lowest loss value.

In the library implementation, the optimizer ‘Adam’ was used for more efficient weight updates by smoothing the loss environment resulting in quicker convergence, which is why the model didn’t train for as many epochs. (geeksforgeeks, 2024). The performance of our model varied from run to run due to the randomness of the weight initialization from the standard normal distribution.

Figure 7: Confusion matrix, results of the FFNN classification



4.6 Termination condition

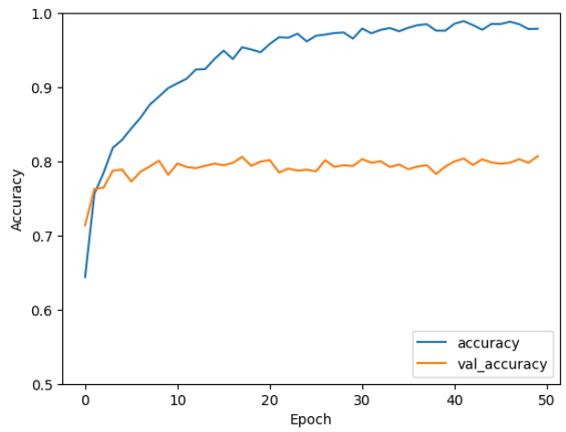
In general, the conditions to stop training should be for when the model parameters have been updated to predict well on both the training and validation set, and the value of the loss gradient is close to minimized. We have implemented early stopping while the performance of both sets is still high.

Additionally, we also introduced L2 regularization to reduce overfitting by reducing weights (Payam, 2024). This ridge regression adds the squared magnitude of the coefficients as a penalty to the loss function to prevent drastic update in model parameters.

5. Convolutional Neural Network (M3)

Convolutional Neural Networks (CNNs) are highly effective for image-based tasks due to several key characteristics. First, they reduce the dimensionality of input data through pooling layers, which preserve essential features while minimizing the size of the image representation. Second, CNNs are robust to spatial variations within images, such as slight shifts or rotations of objects. Lastly, convolutional layers extract meaningful patterns, such as shapes and textures, and encode them into feature maps for further processing.

Graph 4: Training and Validation Accuracy of the



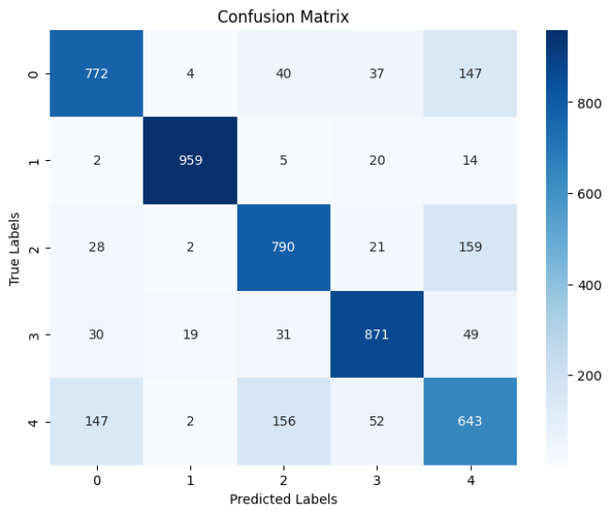
For this project, we developed a CNN using TensorFlow to classify images into five categories. The architecture consisted of three convolutional layers with ReLU activation functions, interspersed with max-pooling layers to reduce the resolution of the data while retaining critical information. Following

feature extraction, the data was flattened and passed through two fully connected layers for classification. The model was trained using the Adam optimizer, sparse categorical cross-entropy as the loss function, and accuracy as the primary evaluation metric.

The training dataset comprised 10,000 images, while the test dataset contained 5,000 images. Training was conducted over 50 epochs, and both training and validation accuracy were monitored throughout. The final test accuracy of 80.70% demonstrates the effectiveness of the model in distinguishing different clothing types, even with its relatively simple architecture.

Additionally, we plotted accuracy curves for both training and validation to assess the model's

Figure 8: Confusion Matrix, CNN Classification



performance over time. While the validation loss exhibited some fluctuations, the validation accuracy remained consistent, indicating the model's robustness in handling unseen data.

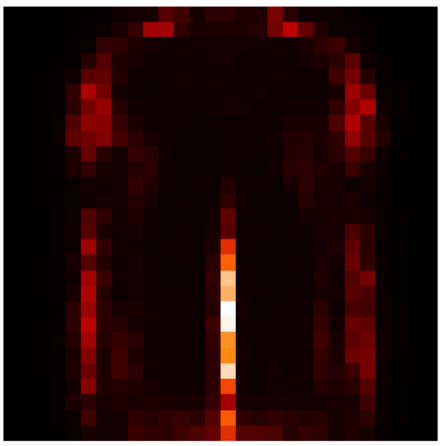
6. Interpretation and discussion of the results

Methods used	Accuracy	Precision	Recall
Decision Tree	79.11%	79.02%	79.18%
FFNN	78.45%	78.96%	78.36%
CNN	80.70%	80.79%	80.70%

Since we are dealing with balanced classes, we chose accuracy as our main evaluation metric because it is easy to interpret and useful to contrast the overall performance of the models as we don’t emphasise the performance on any specific class (Evidently AI). As there isn’t a high cost of false positives, precision is less emphasised, however recall can also be useful to return all relevant entries when these models are used to identify a specific class from large datasets, though similar to accuracy.

Comparing the implementation of each model, decisions trees though prone to overfitting still performed impressively to the unseen test data and generate clear decision boundaries. On the other hand, feed-forward neural networks can be tuned flexibly in training and handle more complex patterns. While the performance is not as impressive, we chose to go with model parameters which still provided a stable improvement in accuracy over the training period compared to a more impressive final model that had a more unstable training accuracy growth. Finally convolutional neural networks are more suitable to image classification since they leverage spatial patters from the data, although they are computationally intensive and thus slower to train.

Figure 9: Feature Importances generated with Random Forest classifier (from SKlearn library)



As seen from figure 4, the lower central pixels of the image contribute highly to the accuracy of the model (geeksforgeeks, 2024). This is important to consider to focus training on most influential features extracted in convolution layers. Compared with FFNNs where using all features /pixels in training is sometimes too computationally expensive and spatial relations between pixels is loss from flattening the input to 1D, CNNs are useful to identify unique shapes and edges to distinguish classes.

7. Future work

Although we originally applied dimensionality reduction to our dataset to make processing and training of large amounts of samples with high features amounts more computationally feasible, it was manageable to apply all our model implementations to the original dataset. While beneficial for preserving maximal information, PCA-processed data samples can still be practical for more complex classifications, such as if this dataset was composed of images of higher resolution and if the sample size was significantly higher.

Furthermore, with more time and ~inspiration~, we could explore more hyperparameter configurations that maximise accuracy, especially with feed-forward neural networks that are variable to an array of activation functions, and structure variables.

8. Conclusion

In this project, we tried out three different machine learning models (Decision Trees, FFNN, and CNN) to classify images of clothing. Each model had its pros and cons. Decision trees were easy to understand but were limited in accuracy. The FFNNs captured more complex patterns by leveraging multiple layers and achieved better performance. The CNN (which is specifically made for image data) worked the best by picking up important details like edges and textures, getting a test accuracy of 80.26%.

We also used PCA to reduce the number of features in the dataset. This helped us make the data easier to work with while keeping most of the important information. After comparing the models, we found that CNNs were the most effective for this task, but there’s still room to improve, especially when it comes to similar-looking classes like shirts and T-shirts.

Overall, this project showed how machine learning can be used for image classification and gave us a good starting point for more experiments in the future.

References:

Bhalla, Deepanshu. “Understanding Bias-Variance Tradeoff.” *ListenData*, 2017, www.listendata.com/2017/02/bias-variance-tradeoff.html.

Dutta, S. (2024, June 7). *Understanding Vanishing and Exploding Gradients - Sanjay Dutta - Medium*. Medium. https://medium.com/@sanjay_dutta/understanding-vanishing-and-exploding-gradients-a8a3c815ffbc

Geeksforgeeks. (2020, October 22). *What is Adam Optimizer?* GeeksforGeeks. https://www.geeksforgeeks.org/adam-optimizer/

Geeksforgeeks. (2024, April 5). *Feature Importances with Random Forests*. GeeksforGeeks. https://www.geeksforgeeks.org/feature-importance-with-random-forests/

Evidently AI. “Accuracy vs. Precision vs. Recall in Machine Learning: What’s the Difference?” *Www.evidentlyai.com*, 1 Oct. 2024, www.evidentlyai.com/classification-metrics/accuracy-precision-recall.

James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). Unsupervised Learning. In G. James, D. Witten, T. Hastie, R. Tibshirani, & J. Taylor (Eds.), *An Introduction to Statistical Learning: With Applications in Python* (pp. 503–556). Springer International Publishing.

Scikit-learn. (2024). 1.10. Decision Trees. Scikit-Learn. https://scikit-learn/stable/modules/tree.html

Sorokin, M. (2024, January 17). *Categorical Cross-Entropy: Unraveling its Potentials in Multi-Class Classification*. Medium; Medium. https://medium.com/@vergotten/categorical-cross-entropy-unraveling-its-potentials-in-multi-class-classification-705129594a01

Thakur, Ayush. “ReLU vs. Sigmoid Function in Deep Neural Networks.” *W&B*, 19 Aug. 2020, wandb.ai/ayush-thakur/dl-question-bank/reports/ReLU-vs-Sigmoid-Function-in-Deep-Neural-Networks--VmlldzoyMDk0MzI.

Weights & Biases. (n.d.). W&B. Retrieved 26 November 2024, from httpss://wandb.ai/ayush-thakur/dl-question-bank/reports/ReLU-vs-Sigmoid-Function-in-Deep-Neural-Networks--VmlldzoyMDk0MzI