IT UNIVERSITY OF CPH

Clothing Classification

Investigating methods for determining types of clothing from images

---

January 2nd, 2023

Ida Ugilt Wennergaard
idwe@itu.dk

Julie Langeland Hagen
jhag@itu.dk

This project report is presented for the course:
BSMALEA1KU Machine Learning,
3rd semester of BSc, Data Science
IT University of Copenhagen
Denmark

Github Repository:
https://github.com/julielhagen/Clothing_classification-

Contents

Introduction

Image classification is a task that humans perform seemingly effortless. But it has taken many years to develop methods that enable computers to recognise and categorize input images. It was not until fairly recently that computers were able to perform trivial tasks such as detecting a puppy in a picture. But during the last couple of years, the study of convolutional neural networks (CNNs) have managed to achieve super human performance on some very complex visual tasks. The scope of the possibilities with these techniques are enormous, and they can be utilized in various applications such as facial recognition, medical image analysis, self-driving cars and image search services. The main factors enabling this development is the increase in computational power and the amount of available training data.[2, p. 445] Deep learning enables excellent performance in tasks such as image classification, but often you can also get quite far by using simpler methods for classification. This report will explore different methods available for determining the type of clothing from an image of the item. The aim is to develop a classifier that can correctly classify as many images as possible using three different methods: a Naive Bayes classifier, a K Nearest Neighbor classifier and a CNN.

Exploratory data analysis and visualisation

In this report, we are working with a data set consisting of 15,000 labelled images derived from the Zalando website(Xiao et al., 2017). Each image is a gray scale image of 28x28 pixels picturing a clothing item, which belong to one of five categories. The categories and an example image from each category is shown in Figure 2.1.

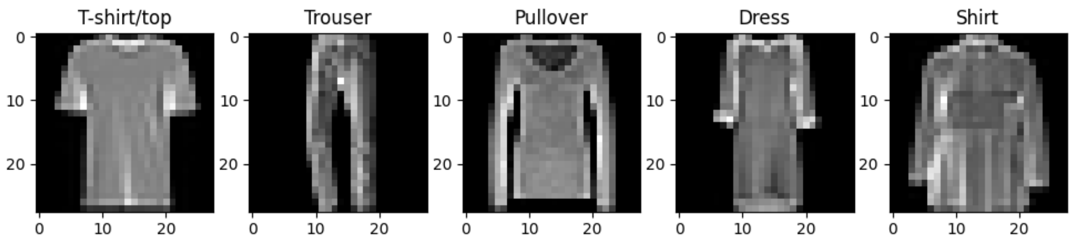


Figure 2.1: Example image from each class.

Table 2.1 shows the distribution of images among the different categories. It shows that there is an equal distribution of images among the different categories.

Type of Clothing	T-shirt/Top	Trouser	Pullover	Dress	Shirt
Label	0	1	2	3	4
Quantity	2033	1947	2001	2005	2014

Table 2.1: Categories of clothes

In the following two sections the data will be investigated further utilizing two methods of dimensionality reduction. Dimensionality reduction is used to reduce the dimension of a p-dimensional data set by projecting it onto a lower dimensional subspace. Among other things this enables us to visualize the data in a meaningfull way. The methods used are Linear Discriminant Analysis (LDA) and Principal

Component Analysis (PCA). Both LDA and PCA are linear transformation methods, but while PCA is unsupervised, LDA is supervised, meaning that it takes the labels of the data into account. PCA gives the directions that maximize the variance of the data, whereas LDA aims to find the directions that maximize the separation between different classes. [6]

2.1 Principal component analysis

2.1.1 Theory

Principal component analysis is a dimensionality-reduction method that aims to lower dimensions while keeping as much variation as possible in the data. [5, p. 498-499] PCA finds a linear combination of features that maximizes the total variance in the data set, and thereby also minimises the mean square error, called principal components(MSE). These components define a new lower-dimensional subspace onto which the original data is projected. The transformed data can be used for visualization or as input for classifiers[2, p. 498-507]. Mathematically, PCA is solved by determining eigenvalues and eigenvectors from the data's covariance matrix, where the eigenvectors corresponding to the largest eigenvalues explain the most variation in the data.

2.1.2 Implementation

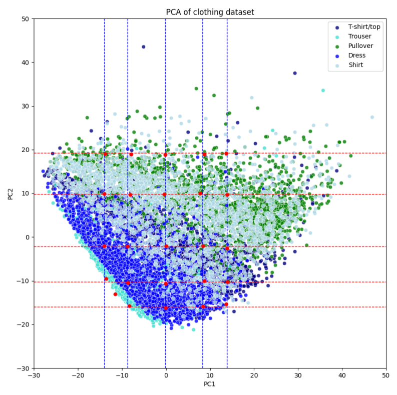
We first standardized our data using the class **sklearn.preprocessing.StandardScaler**. We then used the class **sklearn.decomposition.PCA** to perform the PCA. The first principal component captures the maximum variation in the data, while the second principal component accounts for the second-highest variation.

2.1.3 Results and visualization

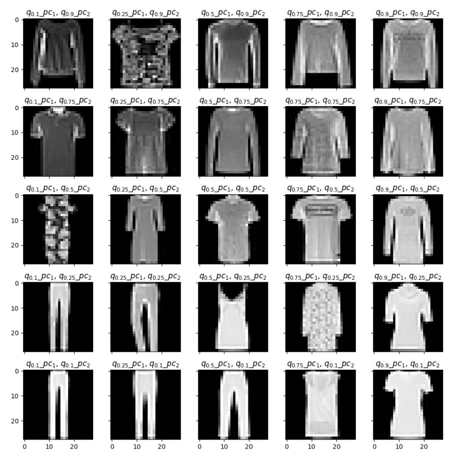
Figure 2.2a shows a scatterplot of the first two principal components. We have computed the 0.10,0.25,0.5,0.75 and 0.90 quantile for both components visualized as the dotted lines in the grid. The red dot represent the images closest to each of the points, where the line representing the quantiles intersect. The method is inspired by the method used in 'The Elements of Statistical Learning' [4, page 536-538] Figure 2.2b shows the images corresponding to the red dots. Such that the image in the top left corner i figure 2.2b corresponds to the red dot in the top left corner in figure 2.2a.

Figure 2.3 show the first two eigenvectors plotted as a heatmap, where negative values, represented by darker colours, symbolises subtraction, and positive values, represented by lighter colours, symbolises adding to the image. If the original image have positive pixel values in the bright areas this contributes to a positive value along the dimension of the component, and correspondingly if the original image have positive values in the dark regions this contributes negatively.

From figure 2.3a you can see that the first eigenvector captures if the clothing item has shoulders and whether it is pants or not. This corresponds to what we see i figure 2.2b where images of pants have a negative value along the first axis, images of t-shirts and pullovers have a positive value, and dresses is somewhere in between as they do not have shoulders but are still filled out in the middle. From 2.3b we can see that images of pants will be assigned a negative value along the axis of the second principal components, and items with long sleeves will be assigned a positive value. Again it corresponds with with we see in figure 2.2b, where it in general is true that the clothing items get longer sleeves as we move up along the second axis.

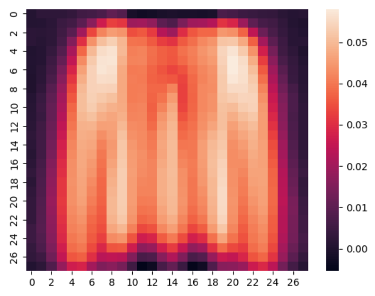


(a) Scatterplot of the first two principal components

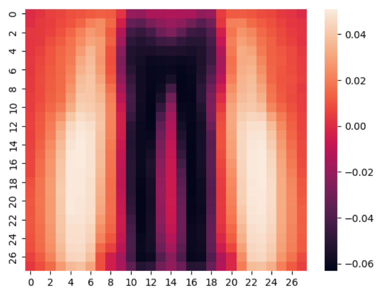


(b) Visualisation of the first two principal components

Figure 2.2



(a) First principal component



(b) Second principal component

Figure 2.3

2.2 Linear Discriminant Analysis

2.2.1 Theory

Linear Discriminant Analysis is a dimensionality reduction technique used to find a linear combination of features that maximises separability between the classes in a data set. In our data set we have five different classes that we want to find linear discriminant variables that best separates the classes. In order to do so we utilized the methods shown in ”Introduction to machine learning” by Ethem Alpaydin.

We want to find a matrix **W** such that when the data are projected onto **W**, the samples from the different classes are as well separated as possible. After projection, for the two classes to be well separated, we would like the means to be as far apart as possible, and the examples of each class to be scattered in as small a region as possible. To achieve this we need to find the between-class scatter matrix $\mathbf{S}_B$ and the within-class scatter matrix $\mathbf{S}_W$. $\mathbf{S}_B$ explains how well separated the classes are, and it is calculated as:

$$\mathbf{S}_B = \sum_{i=1}^{K} N_i(\mathbf{m}_i - \mathbf{m})(\mathbf{m}_i - \mathbf{m})^T \tag{2.1}$$

where m_i is the mean of the ith class, m is the overall mean and $N_i = \sum_t r_i^t$. The within class scatter matrix, $\mathbf{S}_W$, says something about the spread of data within each class, and it can be calculated as:

$$\mathbf{S}_W = \sum_{i=1}^{K} \mathbf{S}_i = \sum_t \mathbf{r}_i^t (\mathbf{x}^t - \mathbf{m}_i)(\mathbf{x}^t - \mathbf{m}_i)^T \tag{2.2}$$

where $\mathbf{r}_i^t = 1$ if $\mathbf{x}^t \in C_i$ and 0 otherwise. After projection, the within-class scatter matrix, $\mathbf{S}_W$, is $\mathbf{W}^T\mathbf{S}_W\mathbf{W}$ and the between-class scatter matrix, $\mathbf{S}_B$, is $\mathbf{W}^T\mathbf{S}_B\mathbf{W}$.

To reach maximum separability, we want to maximise the between-class matrix, $\mathbf{S}_B$, to prevent overlapping between the means, and minimise the within-class scatter matrix, $\mathbf{S}_W$, so that the classes are clustered closely together. Thus, we are interested in the matrix $\mathbf{W}$ that maximizes:

$$J(\mathbf{W}) = \frac{\mathbf{W}^T\mathbf{S}_B\mathbf{W}}{\mathbf{W}^T\mathbf{S}_W\mathbf{W}} \tag{2.3}$$

To achieve this, we need to identify the eigenvectors of $\mathbf{S}_W^{-1}\mathbf{S}_B$. The eigenvectors corresponding to the largest eigenvalues are the eigenvectors that maintain most information. [4, page 140-145]

2.2.2 Implementation

Algorithm 1 shows the pseudo code for our implementation. We have implemented LDA as a Class which is initialized with the number of components. After initializing the class the fit method takes two parameters $\mathbf{X}$ an y, where $\mathbf{X}$ is the input feature matrix and y is the class labels associated with each sample in the input. In the **fit()** function we start by calculating the mean for each variable for the whole data set, and the mean within each each class. Then $\mathbf{S}_W$ and $\mathbf{S}_B$ are calculated using equation 2.2 and 2.1. Then we find the eigenvectors of $\mathbf{S}_W^{-1} \cdot \mathbf{S}_B$ using the **inv** and **eig** functions from the **scipy.linalg** library. We then sort the eigenvectors according to the eigenvalues and save the number of eigenvectors corresponding to the value of **n_components**. Lastly we transform $\mathbf{X}$ with the **transform()** method where we take the dot product of $\mathbf{X}$ and the transpose of the linear discriminants.

To verify our results we also implemented LDA with the class **sklearn.discriminant_analysis.LinearDiscriminantAnalysis**.

2.2.3 Results and visualization

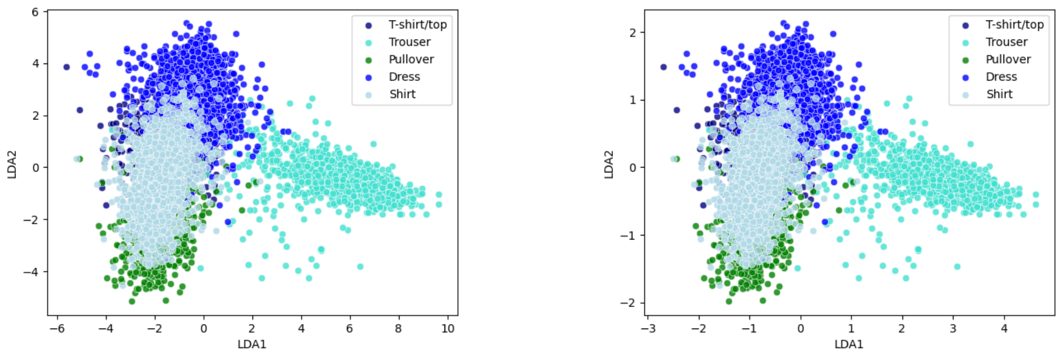
Figure 2.4 pictures a visualization of the data projected on the first two linear discriminant variables. You can see that the scaling is different, but the relation between the eigenvectors are the same.

2.3 Comparison

From figure 2.4 and 2.2a you can clearly see the goal of PCA is to maximise variance while the goal of LDA is to maximise class separability. It is clearly more easy to distinguish between the different classes in the LDA plot, though both methods result in overlapping classes. It is difficult to interpret anything directly from the PCA plot, while we from the LDA plot clearly can see that the trousers are easier to distinguish from than any of the other classes. We therefore all ready expect our classifiers to perform better for this class.

Algorithm 1 LDA Class

```
Constructor: LDA(n_components)
  self.n_components = n_components
  self.linear_discriminants = None

procedure FIT(X, y)
    n_features ← X.get_num_features()
    N ← X.get_num_samples()
    class_labels ← unique(y)

    for c in self.class_labels do
        X_c ← extract_samples_by_class(X, y, c)
        mean_c ← compute_mean(X_c)
        SW ← SW + matrix_multiply((X_c - mean_c),(X_c- mean_c).T)

        n_c ← X_c.get_num_samples()
        mean_diff ← compute_mean_difference(mean_c, means)
        SB ← SB + n_c · matrix_multiply(mean_diff, mean_diff.T)
    end for

    eigenvalues, eigenvectors ← compute_eigen(matrix_multiply(SW_inv, SB))
    eig_values, eig_vectors ← sort_by_eigenvalue(eigenvalues, eigenvectors)
    self.linear_discriminants←first_n_components(eig_vectors, self.n_components)
end procedure

procedure TRANSFORM(X)
    return matrix_multiply(X, transpose(self.linear_discriminants))
end procedure
```



(a) Sklearn implementation

(b) Our implementation

Figure 2.4: Comparison of our own implementation of LDA and LDA using **sklearn** libraries

Methodology

We have implemented three different classifiers, that each tries to solve our classification problem. We have implemented Naive Bayes, KNN and CNN.

3.1 Naive Bayes

The Naive Bayes Classifier is based on Bayes Theorem:

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)} \tag{3.1}$$

where $Pr(Y = k|X = x)$ is the posterior probability that the observation $X = x$ belongs to the kth class given the predictor value for that observation, π_k is the prior probability for the kth class, and $f_k(x)$ denote the density function of X for an observation that comes from the kth class. [5, p. 142]. The Bayes Classifier classifies to the class with the highest posterior probability. By doing this we minimise posterior expected loss. [3, s. 13-14].

$$d(x) = \arg\max_y P(Y = y|x) \tag{3.2}$$

To use Bayes Classifier we need to estimate $\pi_{1,...,K}$ and $f_1(x), ..., f_k(x)$. Assuming that we have a random sample from the population, it is straightforward to estimate π_k, as we can simply compute the fraction of the training observations that belong to the kth class. [5, p. 142-144]

$$\hat{\pi}_k = \frac{n_k}{n} \tag{3.3}$$

To estimate $f_(x), ..., f_K(x)$ we assume that, within the kth class, the p predictors are independent. This assumptions means that for $k = 1, .., K$,

$$f_k = f_{k1}(x_1) \times f_{k2}(x_2) \times \cdots \times f_{kp}(x_p), \tag{3.4}$$

where f_{kj} is the density function of the jth predictor among observations in the kth class. To estimate f_{kj} we can either make assumptions of the distribution of $X_j|Y = l$ or we can use a non-parametric estimate for f_{kj}. There is different non-parametric methods for finding the conditional density functions. A simple way is by making a histogram for the jth predictor within each class. Alternatively, we can use a kernel density estimator, which is essentially a smoothed version of a histogram. For each method you have to choose a step size which defines the bin width.[5, p. 144-156]

3.1.1 Implementation

We have implemented the Naive Bayes classifier as a class. Our implementation has been inspired by an example from a former teaching assistant in the course, Ludek Czinsky [1].

The classifier is initialized with two parameters: stepsize, h and *pdf_estimator_method*, which is the method used to estimate the pdf functions. Our implementation supports using both histograms ('hist') and kernel density estimation ('kde'). After initializing the class, you can fit the classifier by calling the **fit** method, which takes X and y as input. The class prior probabilities is calculated by using equation (3.3). If you choose to use kernel density estimation, then there will be saved a kernel density estimate for each predictor given each class using the class **Scipy.stats.gaussian_kde**. If you choose to use histogram, then there will be computed a histogram for each predictor given each class using the function **numpy.histogram**. For each histogram we save the density value and upper bound for each bin. In both cases the stepsize is used to calculate the binwidth. After fitting the classifier you can call the method **predict**, which takes X as input. It will return the class with the maximum posterior probability as the predicted class for each instance, x_i, in X. The method itself calls the method **predict_proba** which return the posterior probability for each class for each instance, x_i, in X. The posterior probability is calculated using the formula

$$Pr(Y=k|X=x)=\frac{\pi_k \times f_{k1}(x_1)\times f_{k2}(x_2)\times \cdots \times f_{kp}(x_p)}{\sum_{l=l}^{K}\pi_k \times f_{l1}(x_1)\times f_{l2}(x_2)\times \cdots \times f_{lp}(x_p)} \tag{3.5}$$

For each instance, we first calculate the likelihood for each class by multiplying the class prior probability with the class conditional probability for each predictor. If we use histograms to find the conditional probability we simply loop through all the bins until we find a bin where the upper bound is greater than the value of the predictor. The corresponding density is the estimated probability. If we use a kernel density estimate then we simply use the method **pdf** which belongs to the class **Scipy.stats.gaussian_kde**.

3.1.2 Training, cross-validation and test

We have two hyper parameters: h and *pdf_estimator_method*. We use cross validation with ten folds to decide on the optimal parameters. We use the class **sklearn.model_selection.StratifiedKFold** to divide our training data into training data and validation data. It returns stratified folds, that preserve the percentage of samples for each class. For both pdf estimator methods we test models with $h = 0.1, 0.2, ..., 1.0$. In total we test 20 models. For each model we calculate the average accuracy score, the average recall score and the average precision score. To simplify the cross validation, the test scores have been calculated as an average across all classes. We choose the model that scores best. Then we train this model on the full training data and test the model on the test data.

3.2 K-Nearest Neighbours

K-Nearest Neighbour is a classification method that makes predictions for a given observation by finding the nearest K points in the data set and then classify the observation to the class with the highest probability.[5, p.39] Since K-Nearest Neighbours is effective for classifying non-linear patterns and can capture complex decision boundaries in data we selected it as one of our chosen classification models.

3.2.1 Implementation

We implemented this method by using the **KNeighborsClassifier** from **sklearn library**. Using **KNeighborsClassifier** we created a **knn** object with number of neighbours as a parameter. Then we we fit the KNN classifier to the training data. In addition, we fit the classifier to the PCA and LDA transformed training data sets.

3.2.2 Training, cross-validation and test

To identify the optimal number of neighbours we performed cross validation using **cross_validate** from the **sklearn library** with its default 5-fold configuration. To minimize over-fitting and loss of data we wanted to transform the data in each split in the cross validation instead of transforming the data before the cross validation. To do so, we employed the **make_pipeline** function from **sklearn** libraries. We made three different pipelines. In all three pipelines the first step was **StandardScalar**, which standardizes the data, the second step was either LDA,PCA or no transfomation and the last step was **KNeighborsClassifier** for all. Integrating the pipeline for the cross validation ensures that the pre-processing steps are applied correctly during each cross-validation iteration. In order to find the most optimal number of neighbours, we tested for [1,3,5,7,9,11,13,15,17,19,21,23] number of neighbours for the three different data sets. Additionally, we tested for different number of linear disriminants and principal components.

3.3 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of deep learning model. They emerged from the study of the brain's visual cortex, and they are widely used for image recognition, but they are also success full at many other tasks. The core building blocks of a CNN is convolutional layers and pooling layers. In the convolutional layers each neuron is only connected to neurons within their receptive field in the previous layer. This architecture allows the network to concentrate on smaller low-level features in the first layers, then assemble them into larger higher-level features in the next hidden layer. A convolutional layers can consist of multiple filters and corresponding feature maps. All neurons within a given feature map share the same weights and biases, which are updated through back propagation. The goal of the pooling layers is to subsample (shrink) the input image in order to reduce the computational load, the memory usage, and the number of parameters (limiting the risk of overfitting). A pooling layer has no weights, all it does is aggregate the input using a aggregate function. [2, p. 445-458]

3.3.1 Implementation

We have chosen to base our implementation on the LeNet-5 architecture, which has presented in our text book [2] and in lectures [8]. Further, our implementation has been inspired by an example from a former teaching assistant in the course, Ludek Czinsky [1].

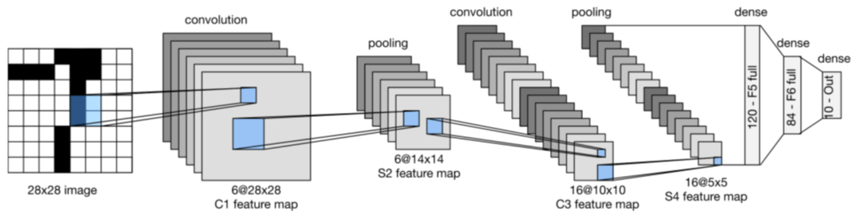


Figure 3.1: Illustration of the LeNet-5 architecture [8]

Modifications

When building the architecture for a CNN, there is a lot of possible variations. To simplify the task we have chosen to base our implementation on the LeNet-5 architecture with a few modifications. We have added batch normalization in each hidden layer before the activation function. The purpose is to mitigate the problem of vanishing and exploding gradients. [2, p. 138-139] There has also been added drop-out to each of the fully connected layers. It reduces the risk of over fitting and has been proven very successful. [2, p. 365] The drop-out rate is one of the hyperparameters that we choose during cross-validation. Since this is a multiclass classification task, we have used the Softmax avtication function for the output layer and Cross entropy as loss function. [2, p. 295] For the hidden layers we have used the sigmoid activation function. As optimizer we have chosen the Adam optimizer. Instead of choosing a specific number of epochs, we have implemented early stopping. The code for the implementation comes from a GitHub repository by Bjarte Mehus Sunde. [7] Early stopping ensures that the training stops when validation loss starts to increase. It saves the model corresponding to the minimum validation loss. During cross validation we choose the batch size, the drop out rate and the learning rate.

3.3.2 Training, cross-validation and test

Through cross validation, we choose the drop out rate, the learning rate and the batch size. We tested for the following values of the three variables: Learning rate = [0.01, 0.001, 0.0001], drop out rate = [0.1, 0.3, 0.5] and batch size = [16, 32, 64]. Again we used the class **sklearn.model_selection.StratifiedKFold** to divide our training data into training data and validation data. For each of these 18 models, we calculate the average loss and accuracy. We choose the model with the highest validation accuracy and train it on the whole training data set. This becomes our final model.

Results

4.1 Naive Bayes

The result of the cross validation can be seen in Figure 4.1.

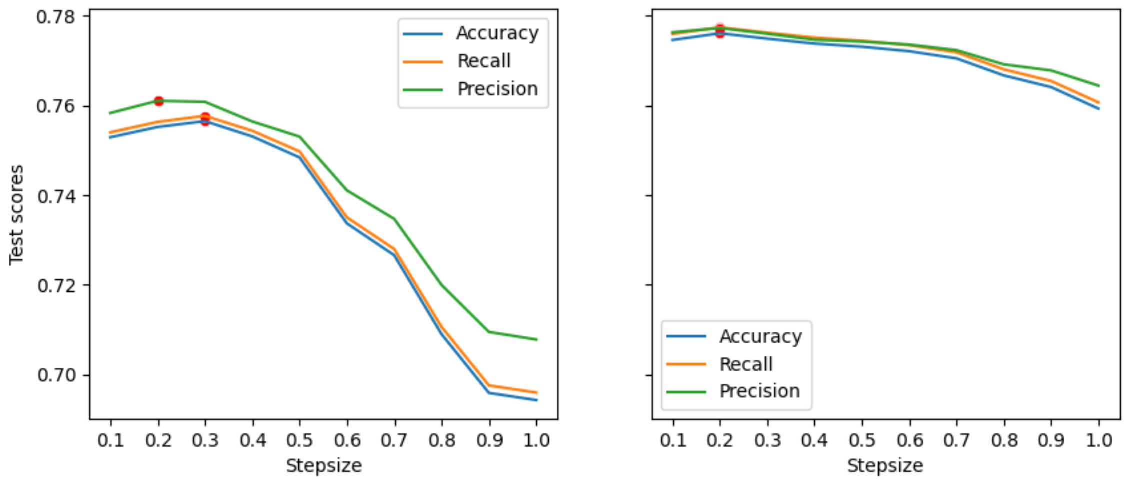


Figure 4.1: Average accuracy, recall and precision score for models with different step size and pdf estimator method.

Based on these results we chose to use the a Gaussian kernel to obtain the pdf estimates and a step size of $h = 0.2$. This gives us a final accuracy score of approximately 73 %. The confusion matrix and the precision and recall score for each class can be seen in table 4.1 and 4.2.

		Predicted					
		T-shirt/top	Trouser	Pullover	Dress	Shirt	Total
True	T-shirt/top	739	8	18	94	141	1000
	Trousers	8	950	6	30	6	1000
	Pullover	45	3	758	13	181	1000
	Dress	159	9	6	806	20	1000
	Shirt	297	8	253	60	382	1000
	Total	1248	978	1041	1003	730	5000

Table 4.1: Confusion matrix for Naive Bayes classifier with $h = 0.2$.

Class	T-shirt/top	Trouser	Pullover	Dress	Shirt
Precision	0.59	0.97	0.73	0.80	0.52
Recall	0.74	0.95	0.76	0.81	0.38

Table 4.2: Recall and Precision scores for Naive Bayes classifier with $h = 0.2$.

4.2 K-Nearest Neighbours

From our cross validation we found that PCA with 83 components, which explain 92% of the variation in the data, gave the highest average accuracy score. For LDA our results showed that utilizing all four linear discriminant variables gave the highest average accuracy score.

In Figure 4.2 we can see that the untransformed and PCA-transformed data set has the highest average accuracy score for $k = 5$, while the LDA has the highest average accuracy score for $k = 21$. We get the overall highest accuracy score when we transform the data in each fold with PCA and lowest when we transform with LDA.

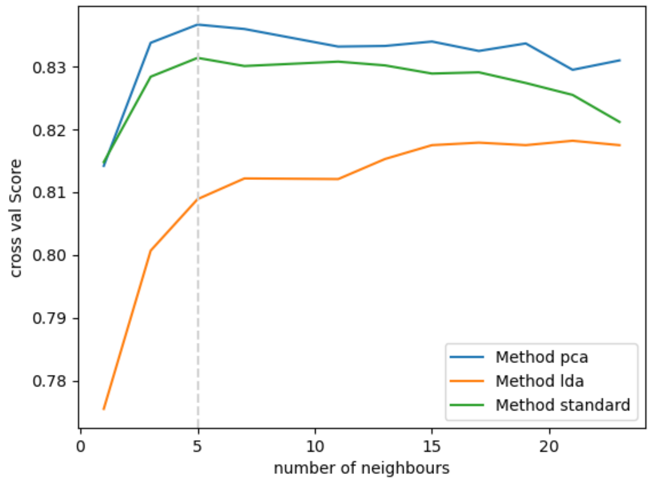


Figure 4.2: Mean accuracy score for different number of neighbours

The final accuracy score for our KNN classifier was 82.2%, and 82.9% when applying PCA for dimensionality reduction. Table 4.4 and 4.3 shows the confusion matrix, precision and recall scores for each class.

		Predicted					
		T-shirt/top	Trouser	Pullover	Dress	Shirt	Total
	T-shirt/top	843	3	17	24	113	1000
	Trousers	8	958	5	18	11	1000
True	Pullover	18	1	821	17	143	1000
	Dress	51	3	19	898	29	1000
	Shirt	189	0	156	28	627	1000
	Total	1109	965	1018	985	923	5000

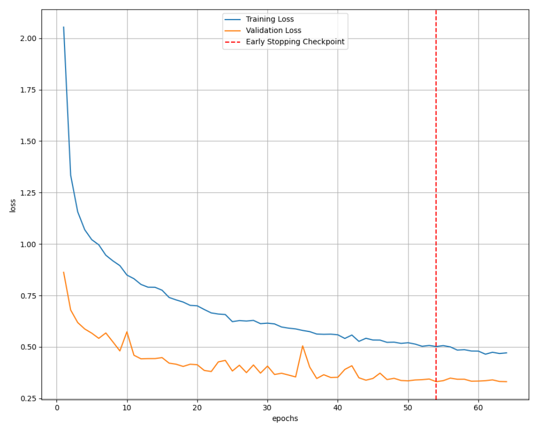
Table 4.3: Confusion matrix for KNN classifier with PCA transformation and $k = 5$.

Class	T-shirt/top	Trouser	Pullover	Dress	Shirt
Precision	0.760144	0.992746	0.806483	0.911675	0.679307
Recall	0.843	0.958	0.821	0.898	0.627

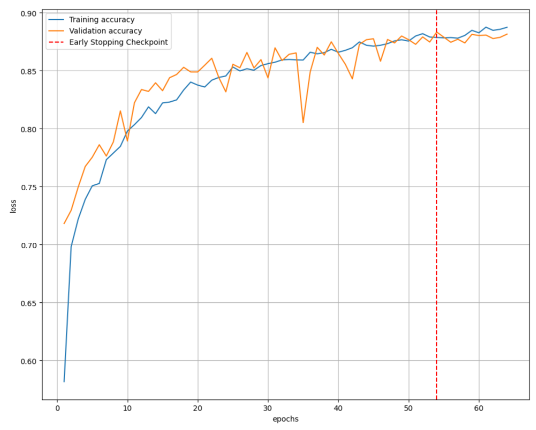
Table 4.4: Recall and Precision scores for KNN with PCA transformation for $k = 5$.

4.3 CNN

The cross validation showed that the model performed best with learning rate = 0.001, drop out rate = 0.3 and batch size = 64. Figure 4.3 shows the loss and accuracy history.



(a) Loss



(b) Accuracy

Figure 4.3

The final accuracy score is 88 %. The confusion matrix can be seen in table 4.1.

		Predicted					
		T-shirt/top	Trouser	Pullover	Dress	Shirt	Total
True	T-shirt/top	843	3	19	26	109	1000
	Trousers	1	976	1	16	6	1000
	Pullover	13	3	864	11	109	1000
	Dress	11	6	5	944	34	1000
	Shirt	126	0	60	33	781	1000
	Total	994	988	949	1030	1039	5000

Table 4.5: Confusion matrix for CNN model.

Discussion

5.1 Naive Bayes

With an accuracy score of 73 % the Naive Bayes Classifier is the classifier performing the worst. One could argue that it is a decent performance taking into account the complexity of the task and the simplicity of the classifier. From figure 2.4, which depicts the data projected on the first two linear discriminant variables, you can see that 'trouser' is the only class which is highly separable from the other classes. While the three classes 't-shirt/top', 'pullover' and 'shirt' are highly overlapping, and the class 'dress' is also somewhat overlapping with these classes. This means the class conditional distributions will also be overlapping. It is therefore not surprising that the NB classifier performs poorly for these overlapping classes. From table 4.2 it can clearly be seen that the classifier performs very well for the class 'trouser' with a precision on 97 % and recall on 95 %, and very poorly for especially shirts and t-shirts/tops. From the confusion matrix in table 4.1 it can be seen that it is especially difficult for the classifier to distinguish between t-shirts/tops, pullovers and shirts. Further, it also has problems distinguishing between dresses and t-shirts/tops. This is to be expected from the visualization of our LDA. But the result is also quite intuitive, since it even for humans can sometimes be difficult to distinguish between these categories.

5.2 KNN

KNN has an accuracy score of 82%, 83% when applying Principal Component Analysis (PCA) as the dimensionality reduction method and 80% when using LDA. Both PCA and LDA stand out as effective techniques for dimensionality reduction but there are some key differences that may be the reason behind the difference in accuracy scores. PCA is an unsupervised method that does not explicitly consider class labels or optimize for class separability, but instead captures the overall structure and variability within the data, while LDA is a a supervised method. Even though LDA also is a great tool for dimensionality reduction, our data set only allow for a maximum of 4 linear discriminants, which might have resulted in a limited ability to capture complex class relationships. PCA, on the other hand, allow us to keep above 92 % of the variation in the data, while still reducing the number of variables fro 784 to 82.

From table 4.4 we can clearly see that the KNN classifier is proficient at identifying the categories; Trousers and Dress, with a precision and recall score of 99% and 96% for Trousers, and 91% and 90% for Dress. Whilst, the categories: T-shirt/top, Pullover and Shirt has much lower scores. From the confusion matrix in table 4.3 we can see that the classifier has trouble when trying to distinguish Shirt from the other categories, especially T-shirt/top and Pullover. For instance the classifier predicted 189 images to be t-shirts/tops and 156 to be pullovers, that were actually shirts. In addition to wrongly classifying images as 'shirt', resulting in a low precision and recall score of 68% and 63%

5.3 CNN

With an accuracy score of 88 %, the CNN classifier outperforms the two other classifiers. This is to be expected, as CNN is developed for image analysis and is a lot more complex than the two other classifiers. Similar to the two other models, this model perform best for the trousers and dresses, and worst for shirts. The model is better at distinguishing between t-shirts/tops, pullovers and shirts than the NB classifier and the KNN classifier. This shows that this model is able to capture more complex relationships in the data than the two others classifiers, as we expected.

The architecture that we implemented is a relative simply compared to some of the newer architectures presented in the book, hence there are room for improvement.[2, p. 460-480] One of the limitations that we faced while training and testing the model was computational power. Due to this we had to test for a limited number of hyperparameters. One of the things that we would have liked to explore is the choice of activation function. We chose to use the sigmoid activation function for all hidden layers. But actually other activation function has been shown to result in better performance. Especially variations of the ReLU action function behave well in deep neural networks. Also, several of the newer architectures like GoogleLeNet, ResNet and SENet, that has been shown to perform better, uses the ReLU activation function. [2, p. 460-480] We would also expect to be able to improve performance by simply making the network larger and deeper like the AlexNet.[2, p. 464-466]

Conclusion

Through our research we have investigated the effectiveness of different classifying methods: Naive Bayes, K-Nearest Neighbours and Convolutional Neural Network. Additionally we looked into dimensionality reduction methods using Principal component analysis and Linear discriminant analysis. From our results we can conclude that CNN, being a method developed specifically for image analysis, was most proficient in classifying types of clothing from an image. Overall, this project contributes to the ongoing discourse on applying machine learning to image classification tasks, with great potential for future research and more effective implementations in clothing classification.

Bibliography

[1] Ludek Czinsky. *ml2022/20 Exercise 20.* 2022. URL: https://deepnote.com/workspace/ludekcizinsky-8f7f55a7-f3c7-4009-b3b2-b7b07fcdf5a7/project/ml202220-44b4318f-b707-4245-8a54-52edfdffa4de/notebook/dd926b1b8ea94ce79538cc9365bc74b7.

[2] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems.* Ed. by Rachel Roumeliotis and Nicole Tache. Second Edition. O'Reiley, 2019. ISBN: 978-1-492-03264-9.

[3] Therese Graversen. *Lecture 6: Bayes Classifiers, K-nearest neighbours [PowerPoint slides].* Sept. 2023. URL: https://learnit.itu.dk/pluginfile.php/364543/mod_resource/content/0/Slides_6.pdf.

[4] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* second edtion. springer.

[5] Gareth James et al. *An Introduction to Statistical Learning with Applications in R Second Edition.* 2. edition. 2021.

[6] Sebastian Raschka. *Principal Component Analysis in 3 Simple Steps.* 2015. URL: https://sebastianraschka.com/Articles/2015_pca_in_3_steps.html?fbclid=IwAR0ErLU497OWz1Y6A4QPwSIzqurO-y2EMSjJJn7V5zyb0HwdHgFuWsPY-tPU.

[7] Bjarte Mehus Sunde. *Bjarten/early-stopping-pytorch.* URL: https://github.com/Bjarten/early-stopping-pytorch/blob/master/pytorchtools.py.

[8] Payam Zahadat. *Lecture 19: Artificial Neural Networks and CNNs [PowerPoint slides].* Nov. 2023. URL: https://learnit.itu.dk/pluginfile.php/371968/mod_resource/content/6/Slides_19.pdf.