Models evaluation for Handwritten Digits Recognition

Machine Learning semester project
Group 09

Jingyan Chen s5015073
& Gonçalo Hora de Carvalho s3450295
& Pedro Rodriguez de Ledesma Jimenez s4745779

Abstract:

This project aims at comparing handwritten digit recognition classifiers using various machine learning algorithms, such as Principal Component Analysis (PCA), Support Vector Machine (SVM), Decision Trees (DT) and K-Nearest Neighbor (KNN). The benchmark task is to classify the widely used MNIST dataset which consists of images of handwritten digits. A Convolutional Neural Network (CNN) model was trained on the MNIST dataset too and used as the gold-standard. The hypothesis was then proposed that despite seeing slight improvements after fitting the algorithms to the PCA data, these would still fail to perform better than a gold-standard algorithm like the CNN. It was found that the KNN and SVM models performed better than the DT model overall. And that there was a slight improvement after processing the data using a PCA. But in the end, all the algorithms underperformed when compared to the CNN, thus the hypothesis held.

1 Introduction

The process of giving machines the ability to detect human handwriting is known as handwriting recognition. This is a complex process due to differences in size, width, and orientation that vary from person to person's writing. As a result, handwriting recognition systems are challenging to research, where pattern recognition and artificial intelligence [8] have contributed significantly to the advancement of this automation process, improving the interface between man and machine [3]. For example, developing Optical Character Recognition (OCR) that is a popular application of handwriting recognition used by many businesses today [10].

In this paper, we are focusing on handwriting digit recognition (*0* to *9*) or classification through automatic machine learning algorithms by using two different datasets, the MNIST (Modified National Institute of Standards and Technology) dataset* and the Digits dataset. Dimensionality reduction techniques are used and evaluated based on the performance. The goal is to train and compare several machine learning algorithms with different configurations. The classifiers used are Decision Trees (DT), K-Nearest Neighbor (KNN) and Support Vector Machine (SVM) as well as a CNN. These will be evaluated on their performance and accuracy on the Digits dataset and the larger MNIST dataset. The codes has been uploaded to the GitHub repository.†

2 Data

For the effects of this report, the dataset given in class will be referred to as the *"small MNIST"* dataset while the larger 60,000-10,000 train-test dataset available used to train the CNN reported in this paper will be referred to as the *"large MNIST"* dataset.

2.1 The Digits dataset

The small MNIST has been provided by the University of Groningen for the purpose of this assignment. This dataset is cleaner and much smaller compared with the large MNIST dataset, and it has

*https://en.wikipedia.org/wiki/MNIST$_database$

†https://github.com/BlueVelvetSackOfGoldPotatoes/NeuralNetworks2021

Figure 2.1: Data sample from Digits dataset or small MNIST.[§]

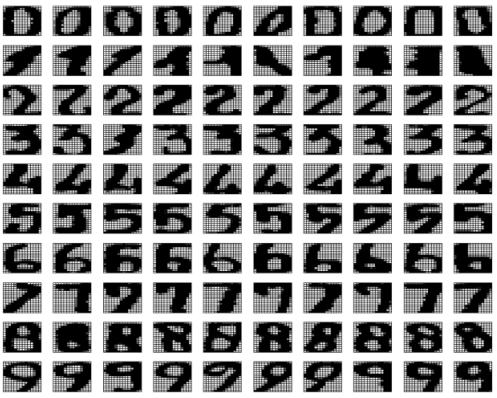
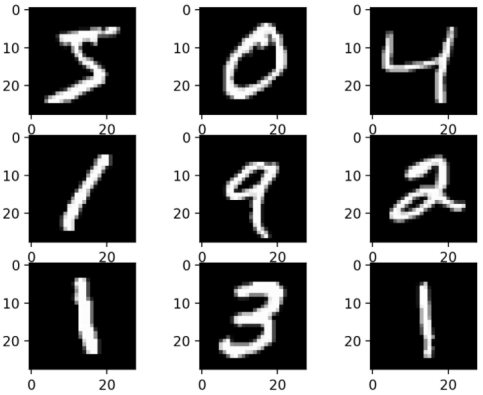


Figure 2.2: Data sample from the large MNIST dataset.



been first used in [5]. There are ten classes in total consisting of the digits from *0* to *9*. Each class has 200 images. Therefore, there is no class imbalance in this dataset. According to the Project Suggestion document, "The data consists of 15 by 16 images, that is, 240 pixels or image vectors. And the grayscale encoding in data file is done by integer steps from 0 (white) to 6 (black)." Figure 2.1 extracted from *Machine Learning* lecture notes shows some examples from the Digits dataset used.[‡]

2.2 The MNIST dataset

The large dataset is a well-known MNIST dataset[¶]. AS As Wikipedia [‖] states, "It is the most widely used benchmark dataset of handwritten digits in machine learning." This dataset is publicly available, and the large MNIST was accessed through deploying *Pytorch* toolbox. This dataset also has handwritten digits from *0* to *9*. Each class has 700 images - there is no class imbalance. Each grayscale image is a square 28 by 28 pixel image. Values in the image vectors can go from 0 (white) to 255 (black) [12][**]. It has a training set of 60.000 examples and a test set of 10.000 examples. Figure 2.2 [2] showcases a sample of the datapoints from the large MNIST dataset.

In conclusion, the difference between these two datasets is their size, where the large MNIST dataset is 30 times larger than the Digits dataset or small MNIST. The decision to use both instead of just one was both technical experiment - to try and work with more than one dataset; and also to obtain a CNN capable of classifying with a higher accuracy. Since these datasets belong to the same distribution of data comparison across algorithms should hold.

3 Methods

As previously mentioned, this paper aims to evaluate different models in the task of handwriting digits recognition that are trained and evaluated on two different datasets. In this section, each model's components will be described and analyzed in depth. First, 3 models are described that had been ran in the small MNIST using a kernel PCA for dimension reduction, to continue with a PCA dimension reduction analysis in both datasets and the description of the CNN model.

3.1 Decision Trees model

The first model used Decision Trees algorithm for classifying, an algorithm that splits the feature space into subsets of decision regions [7]. It has been preselected as suitable due to the expected clustering in the provided dataset. Therefore, a high performance is expected with this algorithm. In this model, the dataset used is the small MNIST, in which a preprocessing step is carried out. The

[‡]https://www.ai.rug.nl/minds/uploads/LN$_{M}L_{R}UG.pdf$
[¶]https://www.tensorflow.org/datasets/catalog/mnist
[‖]https://en.wikipedia.org/wiki/MNIST$_{d}atabase$
[**]http://yann.lecun.com/exdb/mnist/

model performance is evaluated with the original dataset and the reduced dataset obtained through a kernel PCA (non-linear PCA). Non-linear transformation transforms the original data space into a higher-dimensional space, which is the fundamental of a non-linear system. The kernel function takes input vectors in the original space and returns the vectors' dot product in the feature space [10]. Different kernels have been tested: RBF (Radial Basis Function), polynomial, sigmoid and cosine, although the higher performance of the model with the reduced dataset was obtained with the RBF kernel. Hence, the final evaluation uses the best-reduced dataset obtained with the RBF kernel. The performance is evaluated with different split validation: splitting the data 50% train-50% test and k-fold cross-validation. Model parameters are tuned using grid search functions.

3.2 K-Nearest Neighbors model

The model with a similar configuration to the previous one is defined but with a different classifier. KNN [6] is another classifier that computer the distance between input images (different metrics are available and this parameter is tuned). This classifier has a good performance in cluster datasets due to the algorithm's core, where new points are being assigned to the class most common among its k nearest-neighbors [1]. The model is trained and tested with the small MNIST and with the best reduced dataset obtained with the same algorithm (kernel PCA) as the previous model. Similar steps in the reprocessing phase are completed. The performance is evaluated with different split validations: splitting the data 50% train-50% test and k-fold cross-validation. Model parameters are tuned using the grid search function.

3.3 Supported Vector Machine model

Another classifier, the Supported Vector Machine classifier (SVM) is defined and analyzed. The optimal hyperplane is chosen based on its distance from the closest data points in the classes and this hyperplane is then used to classify new data [4]. The model is trained and tested with the small MNIST and reduced with the same algorithm (kernel PCA) as the previous model. Similar steps in the reprocessing phase are completed. The performance is evaluated with different split validation: splitting the data 50% train-50% test and k-fold cross-validation. Model parameters are tuned using the grid search function.

3.4 Principal Component Analysis

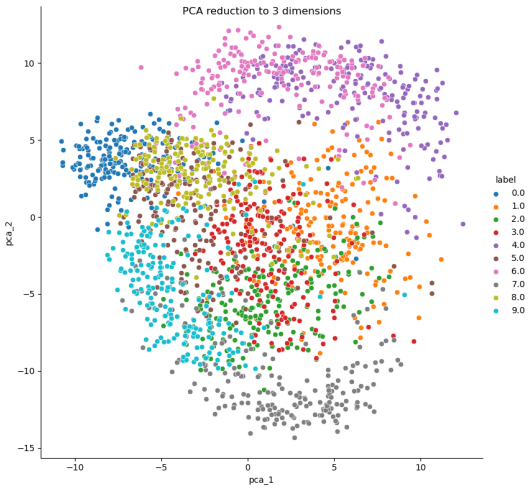
The goal in PCA is to extract the important information from the data and to express this information as a set of summary indices called principal components [11]. These can be taken to be the lines that best fit the data out of all the points that make up, for example, an image. As a rule, the fewer components are used, the less information will be available for reconstructing the data, thus yielding a simplified or summarized object [11].

This technique was deployed to summarize the data used to train the CNN described below. Thus, the following is the data analysis, both the small MNIST and large MNIST using said PCA.

3.4.1 Small MNIST

The following is the PCA fitted from the small MNIST dataset.

Figure 3.1: Visualising the small MNIST data in 3D after fitting a PCA with 3 principle components

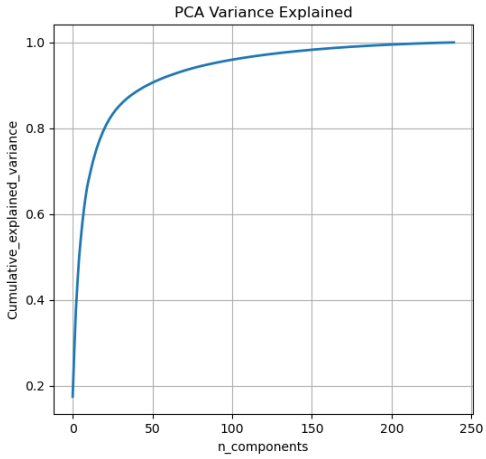


We see PCA 1 and PCA 2, calculated using the eigenvectors for the corresponding eigenvalues 238

and 239. The total data is a 240-dimensional manifold, and the points in Figure 3.1 are the projection of each data point along with the directions with the largest variance. This generates an optimal stretch and rotation in 240-dimensional space that enables the visualization of the digits in two dimensions.

The classes of numbers, or digits, seem to be well linearly separated to some extent. There is a strong overlap between the digits *1*, *3* and *2*, clustering together in the middle. While digits *7*, *9*, *0* and *8* are much more neatly clustered away from the former core. Digits *6* and *4* seem to cluster together too.

Figure 3.2: Explained Variance of the small MNIST data after fitting a PCA with 3 principle components



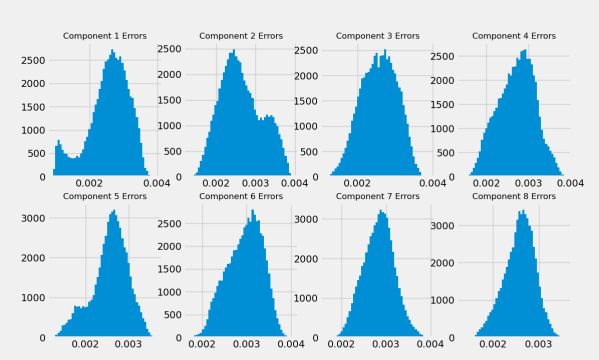
In Figure 3.2 we see that the first 50 components out of a 240-component fitted PCA explain more than %80 of the variance in the data. After these, the added components do not add much more information.

3.4.2 Large MNIST

Since we decided to try another dataset consisting of the same data but more data than the former, we also analysed this using different techniques.

First, we plotted the component error rate. As shown in Figure 3.3, each bin is a composite of measurements of the mean absolute error between a record in the dataset and a component (which are numerically labelled).

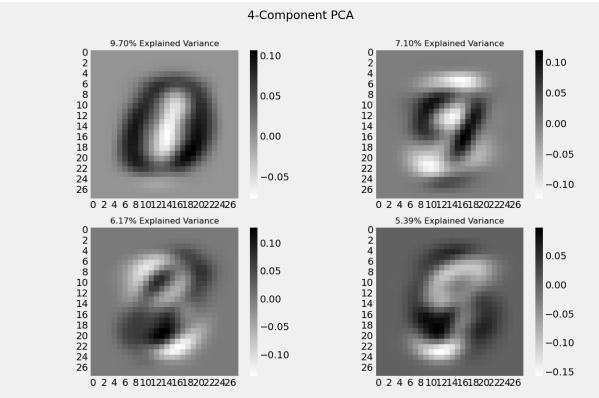
Figure 3.3: Explained Variance of the small MNIST data after fitting a PCA with 3 principle components



A bad component has uniformly distributed errors or is distributed all across the data. A good component has a bimodal distribution, as does the first five components in the above graph, specifically components 1, 2 and 5. This indicates differentiable data—two classes of data that cluster apart.

Next, to visualize a reconstruction of a sample of the data using the first four components we plotted Figure 3.4.

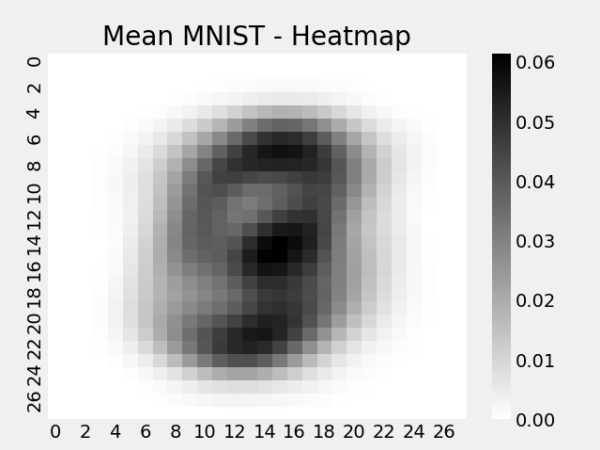
Figure 3.4: 4 component reconstruction - large MNIST samples



Another interesting visualization using the fit-

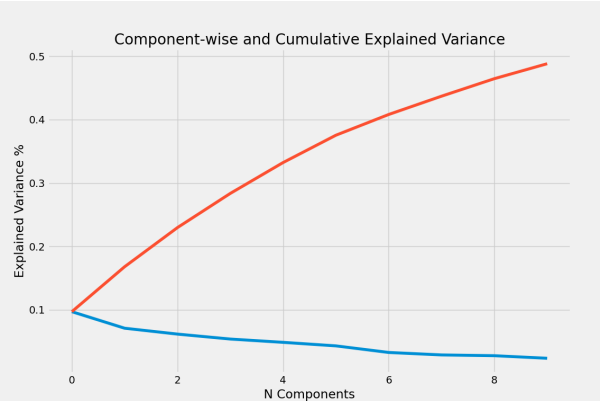
ted PCA is that a mean image can be plotted using Python's library Seaborn, more specifically, its *heatmap* method. As shown in Figure 3.5, it summarizes the most relevant distribution of pixels or their density across the data.

Figure 3.5: Mean Image - large MNIST sample



Finally, we plotted the explained variance of the large MNIST in Figure 3.6

Figure 3.6: Explained Variance - large MNIST. Red: cumulative explained variance. Blue: component-wise explained variance.



There is no clear cutoff point from observing Figure 3.6 - the first component seems to offer somewhat of a cutoff, as is evidenced but the sharp angle of the blue curve between components 1 and 2.

3.5 Convolutional Neural Network

We decided to train a CNN on the large MNIST data.

Supervised learning are amongst the most common systems in machine learning. These are of the type where the program is given a set of already labeled or annotated data, and asked to produce correct labels on new data. From all the supervised learning methods available, deep learning has been the productive thus far. Where models learn useful representations and features automatically, directly from the raw data, bypassing an otherwise manual and difficult conceptual step of feature design and extraction.

CNNs are then the preferred algorithm in visualisation tasks [9]. In the fully convolutional network architecture, prediction results from a pixel-wise image segmentation by applying a number of convolutional filters. A CNN contains layers with locally spatially encoded connections. A kernel is a pattern of weights that is replicated across multiple local regions. The process of applying the kernel to the pixels of the image (or to the mapping spatially organized units in a subsequent layer) is called convolution, a term borrowed from signal processing. Any signal can be represented as the sum of scaled and shifted impulse signals. The term is quite synonymous to cross-correlation, which is a measure of similarity of two series as a function of the displacement of one relative to the other. This is also known as a sliding dot product or sliding inner-product. In sum, the model learns features from fine to coarse levels using these mechanics while combining multi-scale features for predicting the label class at each pixel.

The following is then the architecture used for the CNN trained on MNIST. This is a well known MNIST-tested simple architecture. It only needs two convolutional layers, each followed by max pooling, to learn the data. Max pooling layers calculate the maximum value for patches of the feature maps obtained in the convolutional layer before it.

The loss function used is the cross-entropy loss shown bellow:

$$H(p,q) = -\sum p(x)\log(q(x)), \quad (3.1)$$

and a learning rate $\alpha = 0.01$.

Listing 1: Python standard image loading and resizing

```
--------------------------------------------------
Architecture:
    CNN(
    (conv1): Sequential(
        (0): Conv2d(1, 16, kernel_size=(5, 5),
            stride=(1, 1), padding=(2, 2))
        (1): ReLU()
        (2): MaxPool2d(kernel_size=2, stride=2,
            padding=0, dilation=1, ceil_mode=
            False)
    )
    (conv2): Sequential(
        (0): Conv2d(16, 32, kernel_size=(5, 5),
            stride=(1, 1), padding=(2, 2))
        (1): ReLU()
        (2): MaxPool2d(kernel_size=2, stride=2,
            padding=0, dilation=1, ceil_mode=
            False)
    )
    (out): Linear(in_features=1568, out_features
        =10, bias=True)
    )
--------------------------------------------------
--------------------------------------------------
Loss function:
    CrossEntropyLoss()
--------------------------------------------------
--------------------------------------------------
Optimizer:
    Adam (
    Parameter Group 0
        amsgrad: False
        betas: (0.9, 0.999)
        eps: 1e-08
        lr: 0.01
        weight_decay: 0
    )
--------------------------------------------------
--------------------------------------------------
Training - note that the remaining Epochs and
    subsequent steps are available on the Github:
    {'train': <torch.utils.data.dataloader.
        DataLoader object at 0x7f1b2c954670>,
        'test': <torch.utils.data.dataloader.
        DataLoader object at 0x7f1b2c9549d0
        >}
    Epoch [1/10], Step [100/600], Loss: 0.1085
    ...
    Epoch [10/10], Step [600/600], Loss: 0.0830
--------------------------------------------------
```

Figure 3.7: Training Loss line - Large MNIST

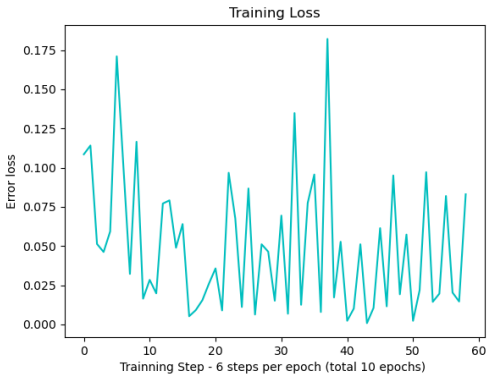
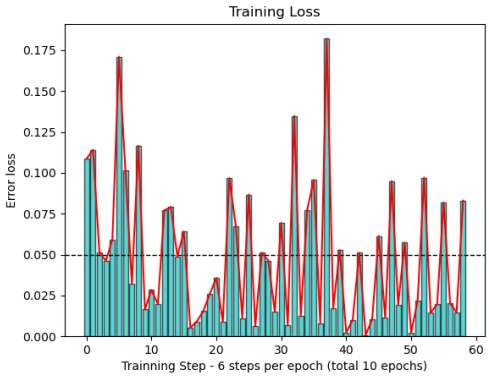


Figure 3.8: Training Loss barplot - Large MNIST



The model's training loss plots, Figure 3.7 and Figure 3.8 do not look optimal, but with an average training loss of 0.05 learning did probably happen (and as will be verified using the test cases in results, it did). The model probably did not overfit since the dataset used had many datapoints and the model was quite simple.

4 Results

In this section, results of the performance of each model with the corresponding datasets are presented. First, the results of the KNN, DT and SVM

pipelines are shown, ending with the performance of the CNN model.

As mentioned before, all models (DT-KNN-SVM) are experimented with the digits dataset without any dimension reduction, with the best-reduced dataset and after tuning the parameters. The number of features reduced in the dataset has been selected by computing the performance with all possible dimensions. The best-reduced dataset in terms of performance has 12 features for the DT model and 18 for the KNN model. This feature represents enough relevant information of each digit image to classify it with a low percentage of misclassification. In addition, the accuracy of the SVM model decreased after the dimension reduction, requiring almost all the features to make a high performance. Therefore, using the dimension reduction algorithm was pointless. Table 4.1 shows the results of the different models defined with before and after dimension reduction and after tuning the parameters. Although all models achieved high performance on classifying digits, the performance of the DT model was lower in contrast with the KNN and SVM models. Despite this, the model's performance did not drop after decreasing the feature space from 240 features to 12 features. Identical results were obtained with the KNN model (18 features).

Table 4.1: Accuracy of the classification with different reduced datasets and classification algorithms.

Models	50% Data Split	Best Reduced Data	Tuned models
DT	85.6%	85.9%	87.5%
KNN	96.3%	96.3%	97.6%
SVM	96.6%	–	97.3%

K-fold cross-validation with 3 and 5 fold was applied to evaluate each model, obtaining higher mean accuracy in all the models (1% -2% higher). This improvement is because the models are using half of the sample for testing in the first approach. Therefore, the model has fewer points for training, although this prevents us from overfitting the model.

Below we test the CNN trained on the large MNIST on a sample of the test cases, and then we run the model on the whole dataset of test cases 10 times and average these to obtain a final performance metric.

Listing 2: Python standard image loading and resizing

```
--------------------------------
Testing prediction with one example:
    Prediction number: [6 5 5 1 7 3 7 9 6 7]
    Actual number:     [6 5 5 1 7 3 7 9 6 2]
--------------------------------
--------------------------------
Testing on test dataset -- note that the
    remaining 8 test cases are available on the
    Github:
    Test Accuracy of the model on the 10000 test
        images: 1.00
    ...
    Test Accuracy of the model on the 10000 test
        images: 0.99
    Average Accuracy: 9.87/10 = 0.987 -> 98%
        correct classification rate
--------------------------------
```

The model thus obtained a 98% classification rate.

5 Discussion

Overall, the experiments dive into PCA data analysis and the subsequent visualization opportunities that arise from this summarization technique. Three typical Machine Learning algorithms are deployed to classify the data: Decision Trees, Support Vector Machines and K-Nearest Neighbors. The CNN model was trained on a super-set of the data offered in the course - this was done to visualize learning better and as an experiment - any comparison drawn across algorithms should still be valid since the data follows from the same distribution. An experiment that could be done in the future is to generate data passed through the PCA, create a simplified or information-concentrated MNIST dataset, and then train another CNN (with the same architecture) on this. Finally, these CNNs should be compared using the correct test set, that is, a test set that follows from the distribution of training data used (e.g. if PCA-translated data was used for training one and the normal MNIST was used to train the other, then use normal MNIST for the latter and PCA-translated data for the former). This could be used as an experiment to show if using a PCA in a machine learning pipeline aids in training this CNN architecture on the MNIST

dataset or not. An assembly of models could also be concatenated where each model offers an 'opinion' in the processing pipeline with the final result being a classification that was averaged out from all the models, each with their own particular 'opinion-weight' [5].

References

[1] Naomi S Altman. "An introduction to kernel and nearest-neighbor nonparametric regression". In: *The American Statistician* 46.3 (1992), pp. 175–185.

[2] Jason Brownlee. *How to Develop a CNN for MNIST Handwritten Digit Classification*. 2019. URL: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/.

[3] Horst Bunke and Kaspar Riesen. "Recent advances in graph-based pattern recognition with applications in document analysis". In: *Pattern Recognition* 44.5 (2011), pp. 1057–1067.

[4] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20.3 (1995), pp. 273–297.

[5] Josef Kittler et al. "On combining classifiers". In: *IEEE transactions on pattern analysis and machine intelligence* 20.3 (1998), pp. 226–239.

[6] Yann LeCun et al. "Learning algorithms for classification: A comparison on handwritten digit recognition". In: *Neural networks: the statistical mechanics perspective* 261.276 (1995), p. 2.

[7] J. Ross Quinlan. "Induction of decision trees". In: *Machine learning* 1.1 (1986), pp. 81–106.

[8] Lambert Schomaker, Katrin Franke, and Marius Bulacu. "Using codebooks of fragmented connected-component contours in forensic and historic writer identification". In: *Pattern Recognition Letters* 28.6 (2007), pp. 719–727.

[9] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[10] Due Trier, Anil K Jain, and Torfinn Taxt. "Feature extraction methods for character recognition-a survey". In: *Pattern recognition* 29.4 (1996), pp. 641–662.

[11] Svante Wold, Kim Esbensen, and Paul Geladi. "Principal component analysis". In: *Chemometrics and intelligent laboratory systems* 2.1-3 (1987), pp. 37–52.

[12] Christopher J.C. Burges Yann LeCun Corinna Cortes. *THE MNIST DATABASE of handwritten digits*. URL: http://yann.lecun.com/exdb/mnist/.