IN4085 Pattern Recognition

- NIST Digits Classification-

Project Report

Group 38

Alexander Berndt	4698959	a.e.berndt@student.tudelft.nl
Ibrahim Chahine	4737792	bchahine@student.tudelft.nl
Georgios Skaltsis	4248425	g.skaltsis@student.tudelft.nl

TU Delft

Contents

Chapter 1

Introduction

In this report, we discuss the design and implementation of numerous classifiers in order to automatically read and correctly interpret handwritten digits. As per the client, two scenarios are investigated. These scenarios are aimed at representing two case-studies which the client often encounters. The scenarios addressed in this report are as follows

- **Scenario 1**
  A training set of more than 200 objects per class. This can be seen as a classifier trained once on a large amount of data after which it is used for its purpose e.g. reading bank cheques.

- **Scenario 2**
  A training set of maximum 10 objects per class. This simulates a classifier which is used to classify small, unique batches of numbers.

The report presents the final classifier design for these two scenarios supported by a detailed motivation on every step taken to reach the final designs. This involves the image pre-processing in Chapter 2, a brief overview of the methods used in the designs in Chapter 3, setting certain design parameters in Chapter 4 and finally, the final designs for Scenario 1 and 2 in Chapters 5 and 6 respectively.

Additionally, a live test is shown in Chapter 7 to compare the trained classifiers on the *NIST* data to real-life hand-written digits by the authors.

Chapter 2

Image Pre-Processing

In this chapter we discuss the image pre-processing methods used throughout the assignment. These pre-processing methods are then used in different combinations to train and test the classifiers in the following chapters. The pre-processing we follows is

1. Morphological Operators
2. De-slanting Image
3. Choosing Pixel Resolution

2.1 Morphological Operators

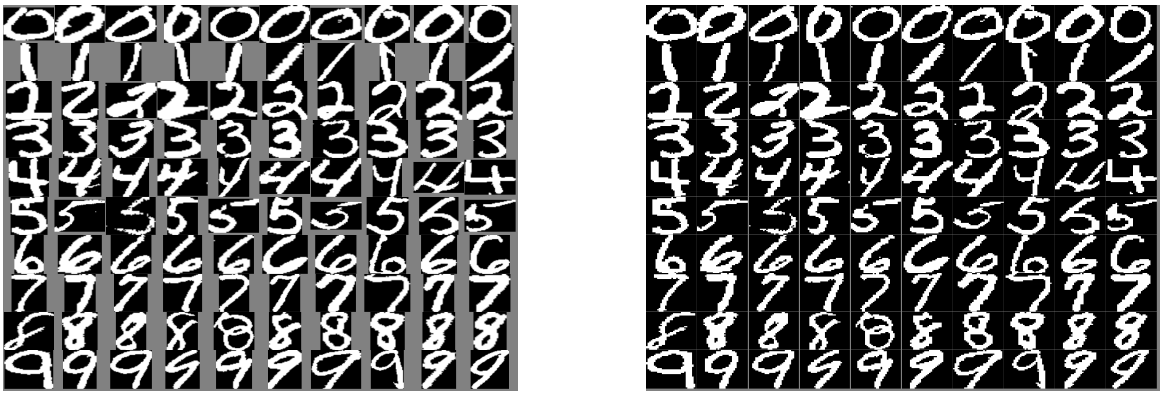

Figure 2.2: After Applying Morphological Operators

As seen in Figure 2.1, the raw NIST data digits have different aspect ratios as well as largely varying line thicknesses, slant angles and general clarity. Firstly, we use *im_center*, *im_stretch* and *im_box* to center the image and force a 1 : 1 aspect ratio. After this, the *im_berosion* and *im_dilation* morphological operators are used with a very low threshold of 2 in an attempt to filter out any irregularities potentially caused by a noisy camera sensor. The order of operation is erosion, dilation, dilation, erosion - the idea being to remove small white dots (noise) but keep thinly written numbers. Finally, the image is re-sized to the desired resolution (e.g. 32x32 or 48x48).

Initially, the dilation and erosion operators were run with higher thresholds in order to filter out noise as well as emphasize thinly written numbers. However, it proved difficult to find thresholds which would not render the image unrecognizable while adequately removing noise. To avoid this risk of rendering images unrecognizable, the threshold was kept low at 2.

2.2 De-Slanting Images

Another issue evident in the NIST data shown in Figure 2.1 is that the numbers can have large variance in their angle. To avoid this, a de-slanting operation is performed. This operation involves determining the image moments [1] of the digits shown in Figure 2.2. These image moments σ_{11}, σ_{20} and σ_{02} can be used to approximate the principal angle θ of the image.

$$\theta = 0.5 \arctan\left(\frac{2\sigma_{11}}{\sigma_{20} - \sigma_{02}}\right)$$

With this angle θ, the image can be de-slanted using a linear transformation as described by $T(\theta)$

$$T(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ \sin(\theta) & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 2.4 shows the significant rotational improvement resulting from this de-slanting operation on the digits shown in Figure 2.3.

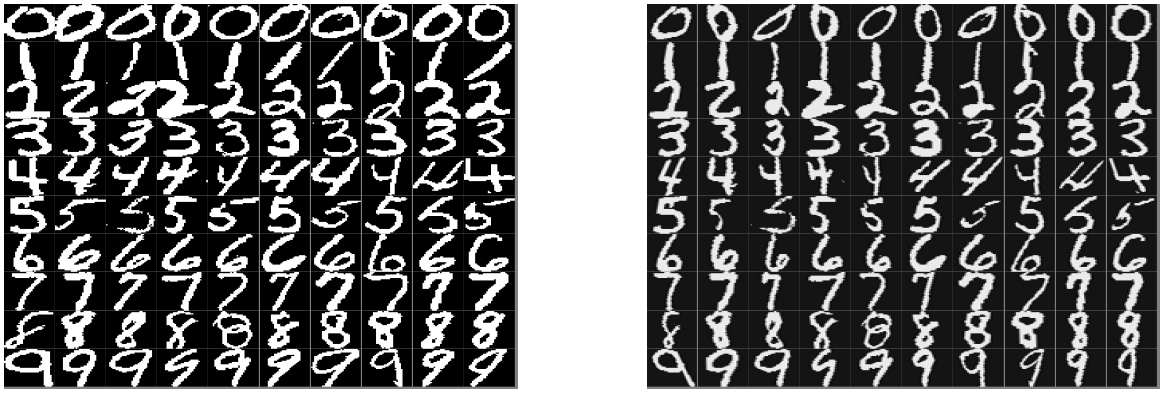

Figure 2.4: After De-Slanting

2.3 Pixel Resolution Decision

An important decision is the final image resolution used for the classifier. However, since different image resolutions can be combined with different PCA variance percentages, the image resolution is considered as a variable design parameter. The important consideration is that a high image resolution ensures a lot of the information is kept, albeit with more noise and higher object dimensionality. Conversely, a low image resolution ensures a lower affect of noise on the object, but at the cost of information. This aforementioned trade-off is considered throughout this report and addressed in later sections.

2.4 Dimensionality Reduction

Images are objects with very high dimensionality due to the relatively high resolution in which the images of the digits are captured. To avoid the affects of the curse-of-dimensionality, we decided to apply a dimensionality reduction technique.

We choose Principal Component Analysis (PCA) since it maximizes the variance of the reduced feature space, meaning that the most information is kept for a reduction in dimensionality. Since image noise can be seen as low variance aberrations, PCA also acts as a noise filter.

We also made sure to first normalize all dimensions and remove the mean of the data before applying PCA so that largely scaled variables don't add a bias to the variance.

As an illustration, the different PCA variances were remapped to the original feature space to show the reduction in image information for decreasing PCA variances (the amount of variance kept by the PCA mapping). This was done using *PRtools* and is shown in Figure 2.5.

Figure 2.5: Re-mapped PCA Representation for 25%, 75%, 95% Variance

Note how the certain digit characteristics are noticeable in the 25% variance image such as the vertical, straight profile of the one and the curve of the three. However, at this low variance we also note the similarities between the seven and the nine. This implies that the components with variance less significant than 25% contain the information which could be used to discern between sevens and nines. If we consider the 75% variance image, we note the digits are far more defined, albeit at a cost of higher dimensionality. The 95% variance image is even clearer to the human eye, but could potentially contain more unnecessary information (noise) which will adversely affect the classification performance.

Naturally, it is important to note the combination of PCA variance to keep, original image resolution (in pixel or HOG feature space) and classifier complexity to ensure accurate classification. Managing this trade-off will be addressed in Chapter 4.

Chapter 3

Overview of Methods

This chapter serves to provide a brief overview of the feature representations and classifiers used on the $NIST$ dataset. A brief motivation is also provided why these specific representations and classifiers were used.

3.1 Image Representations

Pixel Representation:

Representing images using pixels results in a feature of size n^2 where n is the amount of pixels per side of the squared image. This representation is the most basic way of representing an image to a classifier and is similar to the way humans interpret images.

Histogram of Oriented Gradients:

Histogram-of-Orientated-Gradients (HOG) is a method for extracting features from an image widely used for character recognition [1]. HOG feature extraction basically computes the gradient of subsections of an image initially represented by pixels. The different directions of the maximum gradient in each image subsection are then binned (e.g. 0^o to 45^o, 46^o to 90^o etc) into a histogram of angles. The image is then represented by these histogram bins.

Dissimilarity Representation

Dissimilarity representations make use of the relationship between different images from different classes as opposed to their absolute properties such as pixel intensity or HOG histogram bins. Using a training set, a dissimilarity matrix P is created from images already in a certain representation (pixel or HOG). Such a matrix P is visualized in Figure 3.1. Methods of determining the dissimilarity between two images include the euclidean distance between each corresponding feature of two images.

Using Figure 3.1 with euclidean dissimilarity measure as an example, every cell $P(i, j)$ of the dissimilarity matrix P takes its value from the euclidean distance between the i^{th} and j^{th} images of the dataset, represented in a pixel representation form. The diagonal blocks of the matrix represent the distance of objects within the same class. One would expect/want objects in the same class to have a smaller dissimilarity value meaning they would be darker blue in Figure 3.1.

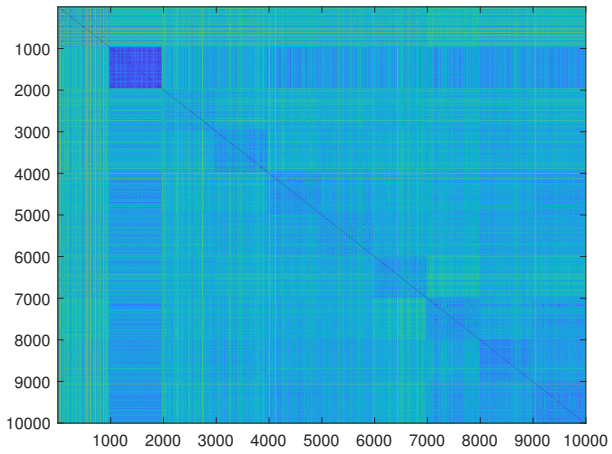

Figure 3.1: Visualized Dissimilarity Matrix of 1000 Objects per Class

3.2 Overview of Classifiers

A variety of different classifiers and classifier types are tested on the provided $NIST$ data in order to ensure the final designs for Scenario 1 and 2 are optimal. Classifiers are differentiated as either parametric, non-parametric and advanced. Each of these classifier types have certain strengths and weaknesses, which is why they are all considered throughout the design process.

Parametric Classifiers

Naive Bayes Classifier

Naive Bayes uses a conditional probability model, where features representing an object are assumed to be independent variables. This classifier is very efficient, but is largely affected by the curse of dimensionality meaning large training set are required to ensure accuracy.

Nearest Mean Classifier

NMC is a fast classifier which computes the mean between an object and the respective feature means of every class. The object is classified to the class with the lowest mean distance. This classifier is scale sensitive. [2]

Linear Discriminant Classifier

LDC is more precisely called Linear Bayes Normal Classifier. This method starts by fitting a normal distribution to the classes assuming equal covariance matrices. Following that a Nearest Mean Classifier is applied to the objects.

Fisher Lest Square Linear Discriminant

FisherC is a special case of linear discriminant classifiers. FisherC does not assume a class normal distribution and equal covariances. Instead, it computes the means and the scatter matrices for each class. Using the results and a least squares approach, FisherC finds the optimal direction where the data is projected with the optimal separability, based on which classification is done [3].

Logistic Classifier

The Logistic Classifier is a simple linear binary classifier that uses regression of logistic sigmoids to return probability values mapped to a number of discrete classes. Its performance has been noted to have little difference with Linear Discriminant Analysis but does not assume Gaussian distribution for classes [4].

Quadratic Discriminant Analysis

Quadratic Discriminant Analysis is a classifier that fits conic surfaces (circles, ellipsoids, parabolas) to separate classes. This classification assumes normal conditional probability density to every class with heterogeneous variance-covariance matrices [3].

Perceptron and Voted Perceptron

The perceptron follows the basic principles of a biological neuron and yields good results for both linearly separable and non-separable classes. The disadvantage of this method is is that, it randomly chooses its weights at first which could cause the fall into a local optima especially when classes are not linearly separable. The voted perceptron is a variant of the perceptron algorithm that uses multiple weighted perceptrons [5]. In other words, this algorithm starts with a new perceptron each time an object is misclassified, initializing the weights to the last weights obtained in the last perceptron, thus avoiding local minimas.

Non-Parametric Classifiers

K Nearest Neighbours Classifier

kNN uses the basic principle of classifying an object based on the votes of its k nearest neighbours, where k is an integer and design parameter. This method can also induce a erroneous classification when training set features are noisy or irrelevant. Different k values greatly influences the classification results: a low value of k causes over-fitting, whereas a high value of k causes class boundaries to be less distinct [6].

Parzen Classifier

The Parzen classifier estimates a Parzen density for every class using kernels for smoothing. The kernel density obtained becomes smooth for a suitable choice of the smoothing parameter h. This classifier needs large quantities of training data to ensure the density estimate converges to the true probability density. [7].

Random Forest Classifier

Random Forest is a decision tree based classifier. The methods starts by creating randomly bootstrapped datasets from the available training data. The varieties of decision trees, allows for a fairer class vote that results in a more democratic and accurate classification [8]. Random Forest classifiers find it hard to predict the model and are computationally expensive, but are less prone to overfitting like single-decision trees.

Advanced Classifiers and Ensemble Methods

Boosting and AdaBoost

Boosting is an ensemble method which uses the classification error of a learned weak classifier on a training set to iteratively re-train that same classifier with a revised training set which emphasizes the previous errors [9]. Adaboost is a specific type of boosting algorithm which combines weak classifiers to classify an object based on a weighted sum of classification results of these weak classifiers. In other words, each classifier contributes to the final classification based on the classification error of each iteration. In addition, the weights on each sample for consecutive weak classifications change. When a first weak classifier misclassifies a sample object, this object's weight is increased for the next weak classification.

Support Vector Machine

SVM is an advanced technique that trains a linear classifier that fits a hyperplane between classes which maximizes the margin between each class of the training set. By using kernel methods, the inputs can be mapped to a higher dimension resulting in a non-linear classification boundary in the original input dimension. Even though the computational effort of kernel methods is vastly decreased by using the kernel trick, it is still not scalable, meaning that training on large training sets takes up a lot of computational resources [3].

3.3 Additional Techniques

Principal Component Analysis

PCA is a dimensionality reduction method. Essentially, PCA is a mapping which maps the original objects to a reduced feature space ensuring maximum variance. This is done by finding mutually orthogonal bases of the scatter matrix of the original objects and mapping the original data on a subset of these orthogonal bases. Since the amount of variance of the original data kept is directly proportional to the amount of orthogonal bases vectors, objects can easily be mapped to a subspace knowing the decrease in variance it will undergo.

Chapter 4

Design Parameters

In this chapter, we specify certain design parameters which will be used throughout the rest of the assignment. With that being said, one may ask:

"Why not investigate all combinations?"

This is a valid question. Why would you set certain constraints when you cannot be sure that they are optimal for the overall, final design? The problem is the sheer amount of variables we can change. The following is a list of options we need to decide on

- Image Representation (Pixels, HOG features, dissimilarity representation)
- Image Resolution (Pixel or HOG resolution)
- PCA variance kept
- Training Set Size
- Classifier Choice

It will be impossible to evaluate combinations of all the aforementioned design parameters. We therefore fix the image resolution, PCA variance and training set size in this chapter, motivated by intuitive and qualitative evidence, and then use these fixed parameters throughout the assignment. For the final classifier design, we will reconsider these choices and adjust them accordingly.

4.1 Training Set Size

For Scenario 1, we are allowed to use between 200 and 1000 training objects per class. However, we need to determine the best amount of training data per class to avoid over-training. For this test, we consider a simple linear classifier, the logistic linear classifier *loglc* classifier, and evaluate the classification error for different training set sizes. This is shown in Figure 4.1.

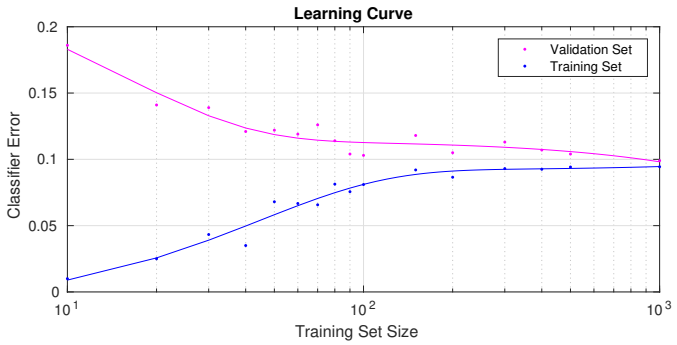

Figure 4.1: Learning Curve of Logistic Linear Classifier *loglc*

From these results, we see that the training and validation curves don't really converge before a training set size of 1000. We therefore decide to use a *training set size of 900* samples per class for Scenario 1 allowing us to validate the classifiers with the remaining 100 samples as a test set.

4.2 PCA vs Image Resolution

In order to determine a good combination of PCA variance percentage kept and image resolution, a simple classifier, the KNN classifier with $K = 3$, was run on different percentage-resolution combinations and the classification error plotted as shown in Figure 4.2. The idea is to find a resolution and PCA dimensionality reduction combination which will keep the relevant information used to discern between the different classes without the excessive information seen in higher image resolutions. Additionally, this is also done to see how much of the dimensionality we can reduce to (with the PCA mapping) without loosing class-dividing information.

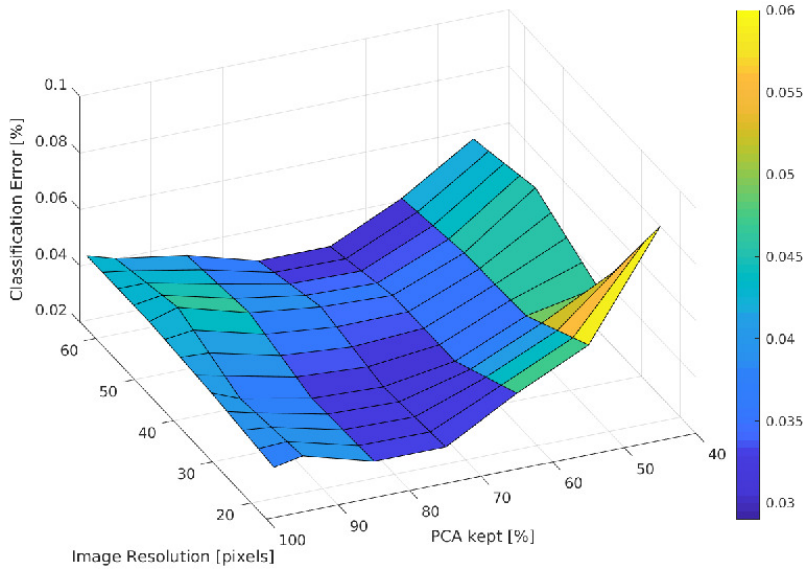

Figure 4.2: Pixel: Graphical comparison of PCA variance percentage and image resolution

From Figure 4.3, it is clear that the optimum PCA-percentage-to-resolution relation follows a linear trend denoted by the dotted white lines. The minimum classification errors are emphasized by the green, red and yellow dots. We decide to use the red dot (85% *variance with PCA and* 32*x*32 *image resolution*) as our design specification. This is motivated as follows

1. The green dot represents a high resolution, implying more computational effort for seemingly the same result
2. These results were obtained using a very simple classifier, meaning the plot will be biased towards smaller feature-counts. To compensate for this bias we choose the red dot over the yellow dot.

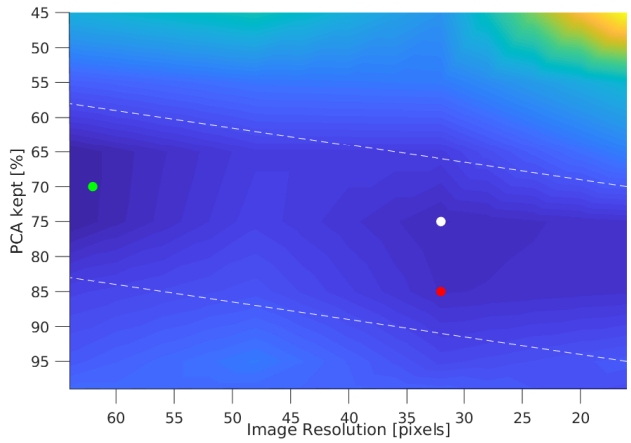

Figure 4.3: Top-view of Figure 4.2 showing general trend (white bounds) and optimum locations (colored dots)

4.3 PCA vs HOG Features Dimensions

The same analysis can be performed for the dimensions of the HOG features as performed in Section 4.2. HOG features are a collection of histograms of the gradients of subsections of the original image in pixel representation. For example, a 64x64 image, which in image resolution translates to a vector of 4096 features, has 1764 features in HOG features (using 14x14 subsections and 9 bins).

The analysis is repeated using the HOG features and the result is shown in Figure 4.4. Note the same valley trend as seen in Figure 4.2. Using the same motivation as described in Section 4.2, we decide to use 95% *variance with PCA and 32x32 image resolution* for the HOG feature representation.

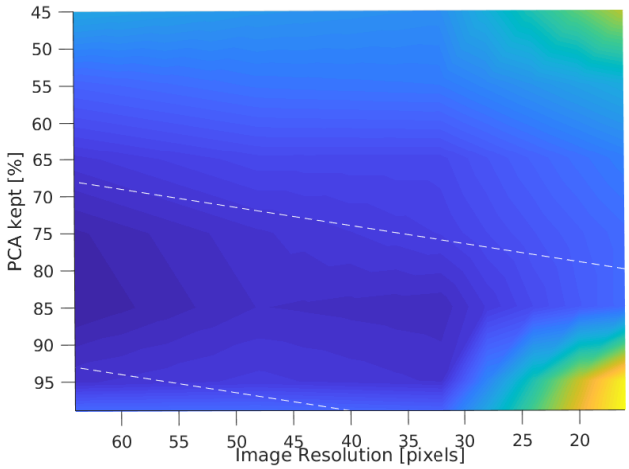

Figure 4.4: Top-view comparison to Figure 4.3 for HOG features showing general trend (white bounds)

4.4 Summary of Design Parameters

The design parameters are summarized as follows

1. Training set size of 900 samples per class.
2. PCA percentage of 85%, image resolution of 32x32 for Pixel Representation.
3. PCA percentage of 95%, image resolution of 32x32 for HOG Representation.

Chapter 5

Scenario 1: Training Set Size $\geq$ 200

In this chapter we evaluate different classifiers on the pre-processed training data described in Chapter 2. The training set size is 900 samples per class as explained in Section 4.4. We train classifiers using this training set and then evaluate it on the remaining 100 samples per class. After obtaining the results from applying different classifiers to three different feature representations, a final optimization step is followed to ensure the entire process is optimal in terms of classification error. Finally, the final classifier is tested with the *nist_eval* command to confirm the estimated classifier performance.

5.1 Pixel Features

As described in Section 4.2, we will consider images of size 32x32 and a PCA variance of 85%. We then evaluated the classifiers described in Section 3.2. The classification errors are tabulated in Table 5.1.

Classifier	**Error %**
K-Nearest Neighbors ($K = 3$)	**3.2**
Parzen	**3.2**
Adaboost KNN	**3.3**
Adaboost LDC	**3.2**
Linear Bayes NC	9.9
Nearest Mean Classifier	20.1
Quad Bayes NC	**3.5**
Naive Bayes	19.5
Random Forest	21.2
Fisher	12.2
Log Linear	7.4
Perceptron	8.8
Voted Perceptron	25.1

Table 5.1: Pixel Representation:
100 test samples per class, Pixel Features, 32x32, 85% PCA

We note good results, especially by the simpler classifiers (KNN, Parzen, Linear Bayes NC) as well as more advanced classifiers such as the Adaboost with either K-NN with $K = 3$. Some classifiers performed poorly, probably because of their non-linear nature (see the Voted Perceptron and Random Forest results).

5.2 HOG Features

As described in Section 4.3, we use an image resolution of 32x32, resulting in 324 HOG features, and a PCA variance of 95%. The classifiers discussed in Section 3.2 are then trained on the 900 HOG-featured, PCA-reduced training samples and tested on the remaining 100 test samples. The results are shown in Table 5.2.

Classifier	**Error %**
K-Nearest Neighbors ($K = 3$)	**2.0**
Parzen	**2.4**
Adaboost KNNC	**2.4**
Adaboost LDC	**2.4**
Linear Bayes NC	**3.0**
Nearest Mean Classifier	n/a
Quad Bayes NC	**2.6**
Naive Bayes	9.8
Random Forest	n/a
Fisher	**3.8**
Log Linear	**2.6**
Perceptron	**3.7**
Voted Perceptron	**2.6**

Table 5.2: HOG Representation:
100 test samples per class, Pixel Features, 32x32, 95% PCA

Using the HOG features, we note an overall improved performance compared to the the pixel representation. The simple classifiers (KNN, Parzen, Linear Bayes NC) all improved and, most notably, the more complex classifiers (Adaboost ensembles) also performed very well. This could lead to an indication that digits are more robustly represented using HOG features compared to the normal pixel features.

5.3 Dissimilarity representation

As described in section 3.1 the dissimilarity representation is being done on pre-processed data. We used an image resolution of 32×32 and applied scaling and a PCA variance of 85%. A 20% of the initial 900 samples per class dataset was used to define a dissimilarity measure, based on which we generated the final training set. This full training set was used to train the classifiers mentioned in Section 3.2 and the results are presented in Table 5.3.

By checking figure 5.1 it is hard to conclude on whether good classification errors should be expected or not. Table 5.3 is of course more informative and reveals that only a few classifiers achieve an acceptable classification error, less than 5%. Namely, Fisher and Log Linear achieve even less than 3%, whereas other simple classifiers, such as KNN and Adaboost KNNC, also perform well with an error slightly greater than the maximum allowed.

Similar tests were done using the HOG features to determine the dissimilarity matrix, but this yielded very poor results for all classifiers. We therefore did not expand on this option.

5.4 Optimizing the Overall Classifier

By comparing the classification errors using the Pixel, HOG and Dissimilarity representations, we note that the best performance is obtained by using HOG feature representations. This is concluded not only because HOG features yielded the lowest classification error (2.0% with KNN-3), but also

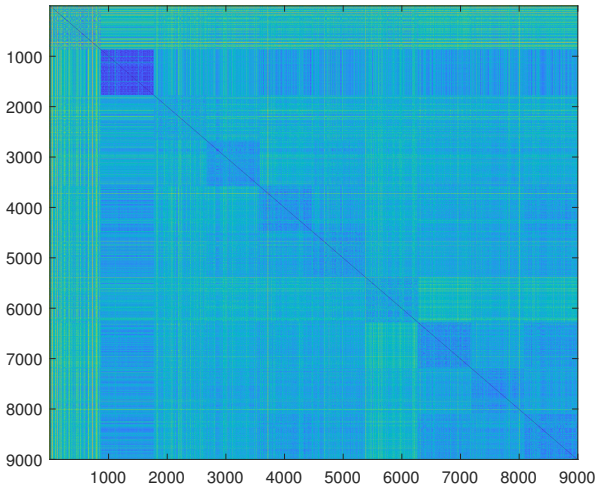

Figure 5.1: Visualized dissimilarity matrix

Classifier	**Error %**
K-Nearest Neighbors ($K = 1$)	5.8
Parzen	90.0
Adaboost KNNC	6.4
Adaboost LDC	84.0
Linear Bayes NC	89.0
Nearest Mean Classifier	90.0
Quad Bayes NC	11.4
Naive Bayes	90.0
Random Forest	16.1
Fisher	**2.8**
Log Linear	**2.1**
Perceptron	73.0
Voted Perceptron	10.9

Table 5.3: Dissimilarity Representation:
100 test samples per class, Pixel Features, 32x32, 85% PCA

because the results were the most consistent over a large variety of simple and more complex classifiers. This implies that the HOG operation extracts the most discerning features of the original pixel images for classification.

From the HOG results in Table 5.2, we choose to further analyze the *KNN*-3, Adaboost with Linear Bayes Normal Classifier (LDC) and Log LC classifiers since they all yielded good results. However, since we set constraints on the PCA variance and image resolution, we are not sure that our final design is globally optimal in terms of all the design variables. Therefore, we consider these three classifiers on different PCA percentages and image sizes. The resultant classification errors are shown in Table 5.4.

From Table 5.4 we notice that the *KNN*-3 obtained the best classification error. However, *KNN*-3 is largely prone to over-fitting due to its low K value, meaning that these results are less likely to occur on a real test set. The *Log LC* classifier did not really yield significant improvement. Finally, the *Adaboost ensemble* with *Linear Bayes NC* (*LDC*) showed improved results for 85% PCA, 64x64 image size and 95% PCA with 32x32 image size.

		Classification Error %								
		KNN $K = 3$			**Adaboost LDC**			**Log LC**		
		PCA Percentage								
		75	85	95	75	85	95	75	85	95
image size	16x16	22.2	12.4	9.7	27.4	17.6	12.7	28.9	16.1	10.7
	32x32	2.3	1.8	2.0	3.1	2.5	**2.4**	4.2	2.5	2.6
	48x48	2.0	**1.6**	2.2	2.4	2.7	2.5	3.4	4.5	23.0
	64x64	**1.6**	**1.5**	2.0	2.5	**2.0**	2.4	2.2	9.4	3.6

Table 5.4: Overall Classifier Optimization for Scenario 2 using HOG features

5.5 Final Classifier Design

With a fear of considering over-training by the *KNN*, we choose the more realistic *Adaboost* with *Linear Bayes NC* (*LDC*) as our final design. Additionally, favoring lower computational effort, we also choose the 32x32 image size with 95% PCA variance. Naturally, if computational effort is not a limiting factor, the 64x64 image size with 85% PCA variance would be a better choice. The final classifier design is as follows:

1. Image representation using HOG features
2. 32x32 image resolution (resulting in 324) HOG features
3. PCA variance reduction of 95%
4. Adaboost Ensemble using Linear Bayes Normal Classifier

This final design was then tested using 10-fold cross validation of the training set data. This was done by splitting the provided *NIST* data into a 900-100 training-test ratio 10 times. Table 5.6 shows the average confusion matrix (for one of the ten cross-validation iterations) of this final design on the 100 samples per class. Table 5.5 shows the estimated error using cross validation and actual error obtained using *nist_eval*.

Scenario 1: Final Design	
Estimated Error	Error using *nist_eval*
2.4 %	2.5 %

Table 5.5: Classification Error Estimation and *nist_eval* Comparison

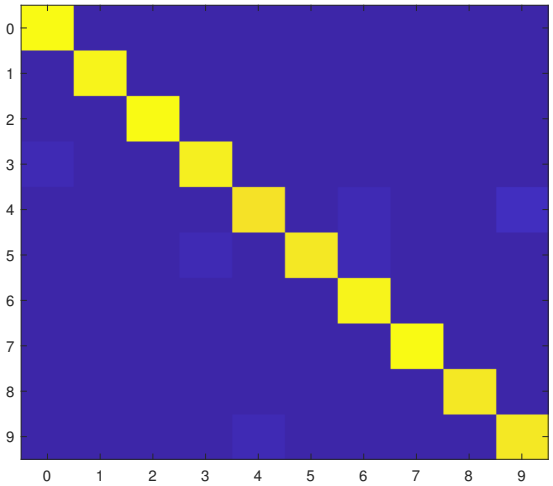

Figure 5.2: Visualized confusion matrix with a 100 samples per class testing set

Scenario 1 Final Design - Adaboost with LDC										
	Classified Label									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**100**	0	0	0	0	0	0	0	0	0
1	1	**97**	1	0	0	0	0	0	1	0
2	0	0	**100**	0	0	0	0	0	0	0
3	2	0	1	**96**	0	1	0	0	0	0
4	0	0	1	0	**93**	0	2	0	0	4
5	0	0	0	3	0	**95**	2	0	0	0
6	1	0	1	0	0	0	**98**	0	0	0
7	1	0	0	0	0	0	0	**99**	0	0
8	1	1	0	1	0	0	1	1	**95**	0
9	0	0	0	0	3	0	1	1	1	**94**

Table 5.6: Confusion Matrix
HOG features, 32x32 image size, 95% PCA variance

Chapter 6

Scenario 2: Training Set Size $\leq$ 10

This chapter details the same process as was followed in Chapter 5, except that we assume a far smaller training set size. The idea being that the classifier could be trained on a portion of a batch of images and then classify the rest of that batch. The training set size is limited to 10 samples per class. Different classifiers are trained on this reduced training set and then evaluated in the same way as in Scenario 1, that is, on a test size of 100 samples per class from the provided $NIST$ data. The classification performance resulting from three different feature representations is explored, followed by an overall optimization step to determine the final classifier design. Finally, the final classifier is tested with the *nist_eval* command to confirm the estimated classifier performance.

6.1 Pixel Features

As described in Section 4.2, we consider images of sizes of 32x32 and a PCA variance of 85%. We analyze a collection of simple and more complex classifiers as described in Section 3.2. The results are tabulated in Table 6.1

Classifier	**Error %**
K-Nearest Neighbors ($K = 3$)	27.1
Parzen	27.1
Adaboost KNN	30.6
Adaboost LDC	30.2
Linear Bayes NC	**18.6**
Nearest Mean Classifier	29.3
Quad Bayes NC	79.5
Naive Bayes	56
Random Forest	29.4
Fisher	**23.7**
Log Linear	26.8
Perceptron	31.2
Voted Perceptron	27

Table 6.1: Pixel Representation:
100 test samples per class, Pixel Features, 32x32, 85% PCA, 10 training samples per class

6.2 HOG Features

Table 6.2 shows the results obtained from the HOG feature representation and PCA variance of 95% as motivated in Section 4.3. Using HOG features, we note a significant improvement in classification results compared to the pixel representation. This once again confirms the fact that HOG features

really emphasize the differences between the different digits and are therefore a good representation for handwritten digits.

Classifier	Error %
K-Nearest Neighbors ($K = 3$)	**12.6**
Parzen	**12.6**
Adaboost KNNC	**15.6**
Adaboost LDC	**10.4**
Linear Bayes NC	**8.2**
Nearest Mean Classifier	**12.5**
Quad Bayes NC	70.2
Naive Bayes	48.8
Random Forest	**15.5**
Fisher	**10.9**
Log Linear	**11.2**
Perceptron	**18.0**
Voted Perceptron	**12.5**

Table 6.2: HOG Representation:
100 test samples per class, Pixel Features, 32x32, 95% PCA, 10 training samples per class

6.3 Dissimilarity representation

As discussed in Section 5.3, the idea behind a dissimilarity representation is to represent a data set based on relations between objects as opposed to absolute values. Using the smaller training set of 10 objects per class, the resulting dissimilarity matrix was constructed and is visualized in Figure 6.1. Table 6.3 shows the results obtained by different classifiers.

Classifier	Error %
K-Nearest Neighbors ($K = 1$)	30.6
Parzen	30.3
Adaboost KNNC	33.3
Adaboost LDC	33.8
Linear Bayes NC	**22.6**
Nearest Mean Classifier	54.5
Quad Bayes NC	90
Naive Bayes	61.8
Random Forest	38.8
Fisher	**19.5**
Log Linear	**20.7**
Perceptron	30.4
Voted Perceptron	48.1

Table 6.3: Dissimilarity Representation:
100 test samples per class, Pixel Features, 32x32, 95% PCA

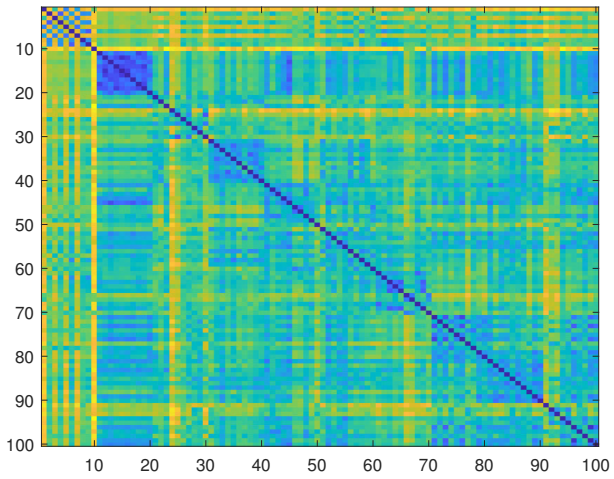

Figure 6.1: Visualized dissimilarity matrix for 10 training samples per class

6.4 Optimizing the Overall Classifier

For the same reasons as described for Scenario 1 in Section 5.4, by comparing the classification errors using the Pixel, HOG and Dissimilarity representations, we note that the best performance is obtained from the *Adaboost ensemble with LDC*, *Linear Bayes NC (LDC)* and *Fisher* classifiers on the HOG features. Once again, since we set constraints on the PCA variance and image resolution, we are not sure that our final design is globally optimal in terms of all the design variables. Therefore, we consider the three aforementioned classifiers on different PCA percentages and image sizes. The resultant classification errors are shown in Table 6.4.

		Classification Error %								
		Adaboost LDC			**Linear Bayes NC**			**Fisher**		
		PCA Percentage								
		75	85	95	75	85	95	75	85	95
image size	32x32	9.7	**8.4**	10.4	13.2	9.6	**8.2**	13.4	12.0	10.9
	48x48	9.6	11.5	16.3	10.9	**8.7**	12.1	12.4	9.4	21.8
	64x64	11.4	14.1	26.7	13.4	13.0	15.8	12.0	12.8	40.2

Table 6.4: Overall Classifier Optimization for Scenario 2 using HOG features

Note that we do not consider the 16x16 image size since the results of Figure 4.4 clearly indicate far worse performance when the image size is so small. This makes intuitive sense because the number of HOG features of a 16x16 are far fewer than 256, meaning that the HOG gradient operation is largely distorted by image irregularities making classification more challenging. These effects are only exacerbated when there are only 10 training samples per class.

We note that the best results are obtained with the *Linear Bayes NC* with a PCA variance of 95% and an image size of 32x32. This is an indication that our initial constraints set in Section 4.3 were defined correctly.

6.5 Final Classifier Design

For the final classifier design, we choose the simple, **parametric** *Linear Bayes NC* with a PCA variance of 95% and an image size of 32x32. In order to obtain a good estimate of the classifier performance, the classifier was trained on a set of 10 samples per class and then validated with 100 different samples a total of ten times. The idea was to perform a validation method similar to 10-fold cross

validation, but without testing on the entire *NIST* dataset as this would take too much time.

The final design for Scenario 2 is

1. Image representation using HOG features
2. 32x32 image resolution (resulting in 324) HOG features
3. PCA variance reduction of 95%

Table 6.6 shows the average confusion matrix of the 10-fold cross-validation scheme. Figure 6.2 shows the final confusion matrix. The final estimated classification error and error using *nist_eval* are shown in Table 6.5. We note that the estimated classification and actual error are very close, indicating that our cross-validation scheme was representative of the actual error.

Scenario 2: Final Design	
Estimated Error	Error using *nist_eval*
8.2 %	8.5 %

Table 6.5: Classification Error Estimation and *nist_eval* Comparison

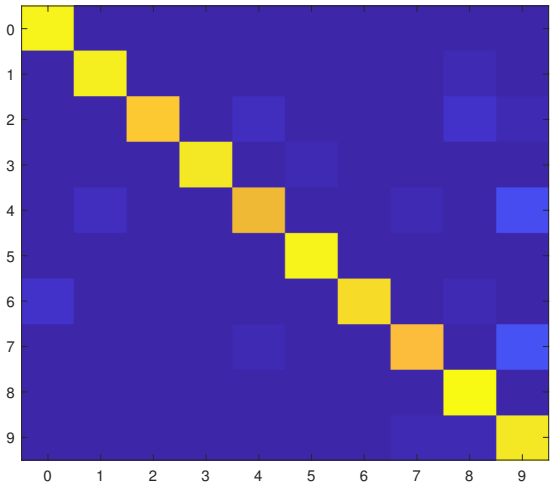

Figure 6.2: Visualized confusion matrix with a 100 samples per class testing set

Scenario 2 Final Design - Linear Bayes NC										
	Classified Label									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**97**	1	0	0	1	1	0	0	0	0
1	1	**95**	0	0	1	0	1	0	2	0
2	0	1	**86**	1	4	0	0	1	5	2
3	1	0	1	**94**	0	2	0	0	1	1
4	0	4	0	0	**80**	0	0	3	0	13
5	0	1	0	0	0	**97**	0	0	1	1
6	6	1	0	0	0	0	**91**	0	2	0
7	1	1	0	0	2	0	0	**82**	0	14
8	0	0	0	0	1	0	0	0	**99**	0
9	0	0	1	0	0	0	0	3	2	**94**

Table 6.6: Confusion Matrix
HOG features, 32x32 image size, 95% PCA variance, 100 test samples per class

Chapter 7

Live Test

7.1 Data Preparation for Live Test

In order to obtain better insight into the practical performance of the classifier design in Chapters 5 and 6, a live test is performed. A set of 100 self-made handwritten digits (10 per class) are labelled and imported into the classifier test framework. The best classifier designs of Scenario 1 and 2 are then tested on this newly created test set and the results compared to the classification error obtained using the *NIST* test and *nist_eval*.

Figure 7.1 shows the handwritten digits used as input to the test framework. A bounding box script was written to locate clusters of dark lines (digits) and tuned to locate the digits on a white sheet of paper.

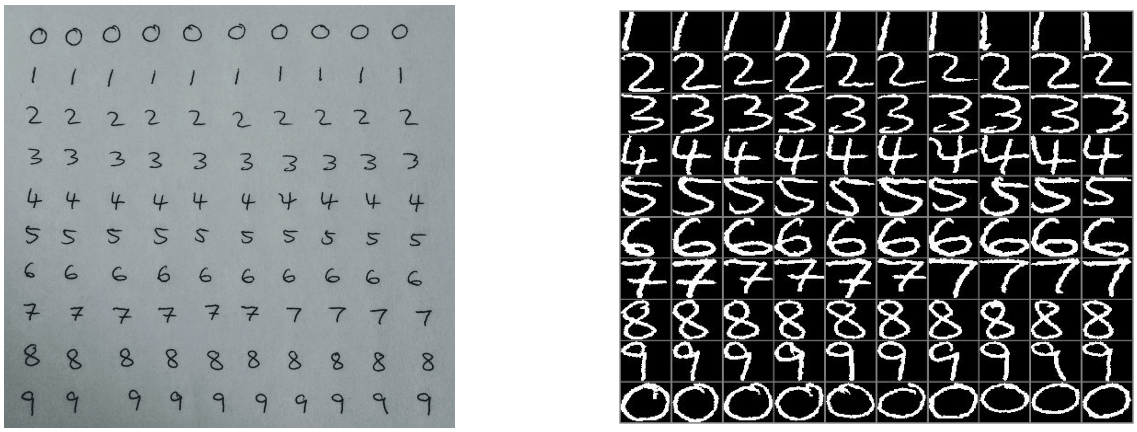

Figure 7.1: Raw Handwritten Digits and Imported in Testing Framework

7.2 Classification Results and Comparison

We compare the performance of the classifier design of Scenario 1 and 2 to the handwritten digits. These results are shown in Table 7.1. For both Scenarios, we note a significant difference in the estimated performance and that of the Live Test data. The confusion matrices of the Live Test results shown in Table 7.1 are shown in Table 7.2 and 7.3 respectively.

Scenario	Classifier	Estimated Error	NIST Error	Live Test Error
1	*Adaboost with LDC*	2.4 %	2.5 %	11 %
2	*Linear Bayes NC*	8.2 %	8.5 %	25 %

Table 7.1: Classification Error Comparison:
Testing set, *NIST* test set and handwritten test set

Scenario 1 Adaboost with LDC										
	Classified Label									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**9**	0	0	0	0	0	0	0	0	1
1	0	**10**	0	0	0	0	0	0	0	0
2	0	0	**10**	0	0	0	0	0	0	0
3	0	0	0	**9**	0	1	0	0	3	0
4	2	0	0	0	**7**	0	0	0	0	1
5	0	0	0	0	0	**10**	0	0	0	0
6	0	0	0	0	0	1	**9**	0	1	0
7	1	0	0	2	0	0	0	**7**	0	0
8	0	0	0	0	0	0	0	0	**10**	0
9	0	1	0	0	0	0	0	0	1	**8**

Table 7.2: Live Test using Design of Scenario 1
HOG features, 32x32 image size, 95% PCA variance

Scenario 2 Linear Bayes NC										
	Classified Label									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	**8**	0	0	0	0	0	0	0	0	2
1	0	**9**	0	0	1	0	0	0	0	0
2	0	0	**10**	0	0	0	0	0	0	0
3	0	0	2	**4**	0	1	0	0	3	0
4	2	0	0	0	**8**	0	0	0	0	0
5	0	0	0	0	0	**10**	0	0	0	0
6	0	0	0	0	0	1	**8**	0	1	0
7	0	0	0	0	5	0	0	**4**	0	1
8	0	0	0	0	1	0	0	0	**9**	0
9	0	2	0	0	0	0	0	3	0	**5**

Table 7.3: Live Test using Design of Scenario 2
HOG features, 32x32 image size, 95% PCA variance

Chapter 8

Recommendations

In this chapter, we consider a list of recommendations for the client given the classifier designs developed for Scenario 1 and 2 and the results of the Live Test.

- *Applicability of HOG Features*
  The results indicate that HOG features are a very good feature extraction method for digits classification and are highly recommended for the final design decision.

- *Layered Classification Scheme*
  From the confusion matrices of the final classifiers for both Scenario 1 and 2, it was noticed that classifiers sometimes have difficulty discerning between two similar looking digits such as 3s and 8s. For this example, an option would be to train a second classifier on only 3s and 8s and when the first classifier classifies a digit as a 3 or an 8, the second, more complex/accurate classifier can determine which one it is.

- *Use Active Learning Techniques*
  From the results we noticed that some digits are always problematic for certain classifiers. An example is the 9 which is often incorrectly classified. To overcome this, an active learning technique could be devised which requires user input when the classifier's confidence of a certain class is below a user-specified threshold.

- *Lack of Diversity in NIST set*
  From the Live Test in Chapter 7, we noticed that the classification error of handwritten digits was much higher than with the *nist_eval* test data. This could indicate that the $NIST$ dataset does not encapsulate a wide enough variance in handwriting styles. For the final design, it would be highly recommended to consider more varied training data.

- *Training Set Size*
  From the analysis in Section 4.4, a larger training set would not necessarily improve the results. This is motivated by the good results obtained using the *nist_eval* testing set. However, as mentioned in the previous point, a revised training set might be necessary.

- *Dissimilarity Features and Time Constraints*
  For large training sets, the dissimilarity matrix take relatively long to compute. This could be a problem if the classifier needs to be trained on-the-go such as the batch training of Scenario 2.

- *Using a Single Classifier*
  Our design notes two different classifiers which were optimum for their respective scenarios. However, we do note that the Adaboost LDC classifier performed well in both scenarios using the same features (HOG) and PCA variance. That being said, it is still recommended that the optimal classifier be used for each respective scenario.

- *Reject Options*
  For digits that are difficult to classify, it could be advisable to make use a reject policy which requires the cheque to either be read manually, or ask the customer to rewrite it in the case that too many digits have a low classification certainty.

Chapter 9

Conclusion

In this report we discuss the steps required to create a digit classification system. Following a detailed analysis and discussion on image pre-processing and classifier choice, the two provided scenarios were analyzed for which a globally optimal classifier was designed for each.

The final classifier designs were trained on Histogram-of-Oriented-Gradients (HOG) feature representations in a 32x32 image size with a PCA variance reduction of 95%. For Scenario 1, a linear bayes NC classifier was trained on 1000 objects per class and for Scenario 2, an ensemble method, specifically, AdaBoost with LDC, was trained on 10 objects per class. The final classification errors for these two scenarios were 2.5% and 8.5% using the *nist_eval* (hidden) test dataset.

The two classifiers were tested on a self-made handwritten digits dataset of 10 objects per class in a so-called Live Test. The classification error was 11% and 25% for the two scenarios respectively. The decreased accuracy for the Live Test was attributed to a bias in the handwritten digits in the $NIST$ dataset.

Finally, a list of recommendations detailing important considerations and possible improvements for a real-life application of these classifiers is given such that these designs can be used in practice.

Bibliography

[1] Guido Gerig. *Shape Analysis - Moment Invariants*. http://www.sci.utah.edu/~gerig/CS7960-S2010/handouts/CS7960-AdvImProc-MomentInvariants.pdf. [Online; accessed 11 Dec 2018]. 2010.

[2] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. DOI: 10.1017CBO9780511809071.

[3] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition*. 4th. Orlando, FL, USA: Academic Press, Inc., 2008. ISBN: 1597492728, 9781597492720.

[4] Trevor Hastie et al. "The Elements of Statistical Learning: Data Mining, Inference, and Prediction". In: *Math. Intell.* 27 (Nov. 2004), pp. 83–85. DOI: 10.1007/BF02985802.

[5] Yoav Freund and Robert E. Schapire. "Large Margin Classification Using the Perceptron Algorithm". In: *Machine Learning* 37.3 (1999), pp. 277–296. ISSN: 1573-0565. DOI: 10.1023/A:1007662407062. URL: https://doi.org/10.1023/A:1007662407062.

[6] Brian Everitt. *Cluster analysis*. Wiley, 2011. Chap. Miscellaneous Clustering Methods.

[7] Zhi-Wei Pan et al. "Parzen windows for multi-class classification". In: *Journal of Complexity* 24.5 (2008), pp. 606 –618. ISSN: 0885-064X. DOI: https://doi.org/10.1016/j.jco.2008.07.001. URL: http://www.sciencedirect.com/science/article/pii/S0885064X08000526.

[8] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[9] Yoav Freund and Robert E Schapire. "A Short Introduction to Boosting". In: *Journal of Japanese Society for Artificial Intelligence* 14 (Oct. 1999), pp. 771–780.