Recognizing Hand-Written Digits

Christine Vinck (4627946), Arthur Brussee (4759680), Husain Kapadia (4707915)

February 2, 2018

Abstract

In this report we explore the possibilities of using pattern recognition techniques, to recognize hand-written digits on bank cheques. Several classification models are implemented using different representation of the data. As a result a optical character recognition tool is developed along with recommendations for its application in practice.

1 Introduction

The process of clearing handwritten bank cheque's can take up to a few days of manual work. Automatic bank cheque processing applications, based on pattern recognition techniques, offer a simplification to this process, making it faster and more efficient. Optical character recognition, or *OCR*, is the process of converting scanned document files into machine readable and searchable format files. To clear handwritten digits on a bank cheque, the account number and the amount of money have to be recognized with high accuracy in order to transfer money to the right address. These digits may look different in every persons handwriting. To automate this task, it is required to find a classifier that generalizes well to all variations in hand writing styles. In this report, we explore multiple classifiers and representations of the data, to make recommendations about a good setup for an automatic bank cheque processing application. The performance of the classifiers will be evaluated in two scenario cases:

- Scenario 1: all training data is available at once.
- Scenario 2: the training data is only available in small batches, say, approximately the size of the current batch of cheques to be processed.

The data used for this problem, is the NIST dataset - a standard dataset of handwritten digits provided by the US National Institute of Standards & Technology (NIST). This dataset consists 2800 images of 128 x 128 pixels for each of the 10 classes, representing the 10 digits (0 - 9). The training NIST dataset will be used to train the models, and the provided test set will be used as a benchmark for testing the results of the models. To further validate the results in a practical setting, a 'live test' will be incorporated using scans of our

own handwriting. Finally, a comparison will be made between the actual (live) performance, the expected (training) performance and the benchmark (test) performance for all classifiers, data representations, and scenarios.

2 Methodology

To explore the possibilities of automatically recognizing hand-written characters using pattern recognition techniques, five classifier models has been applied to three different representation of the data. Additionally, the performance of the classifiers will be evaluated for two scenarios: in the first scenario the classifiers are trained on a large amount of data (up to 200 objects per class) at once, in the second scenario the classifiers are trained on a small set of data (up to 10 objects per class). The models are implemented in MATLAB, with the use of the PRtools library. The default classifiers for each type from the PRtools library were used. The following paragraphs will discuss the data representations, the models used, the selection of features and the testing and validation of the models.

Data representation The first step is to determine how to represent the data to the model, since this can have a substantial impact on the performance of different classifiers. The following three representations for image data will be compared:

1. *Features*: Several features are extracted from the input data using the `im_features` from the PRTools. These features include the Area, Perimeter, Eccentricity, Solidity, etc. see Appendix **??** for a complete list of features that are measured.

2. *Raw*: The raw pixels of the image is used, where each pixel is interpreted as a feature. The images are all re-sized to 8 x 10 pixels. The reason for resizing the image from 128 x 128 to 8 x 10 was to avoid the *Curse of Dimensionality*, since every pixel would be treated as a feature and we do not have enough data samples given a large feature space.

3. *Dissimilarity*: This representation is uses measures that estimate the dissimilarity between pairs of objects based on the Euclidean distance metric.

Data transformation The scaling of the features can have a negative effect on the performance of some models. So, for each representation the data is normalized - the final features are re-sized and transformed such that the mean is 0, and the variance is 1. Similarly, having too many features can also affect the performance of the models due to the risk of over-fitting, also known as the *curse of dimensionality*. For this problem, a principal component analysis (PCA) routine was used as a feature extraction technique, to linearly map the features to a reduced feature space. To optimize the final PCA dimension, feature curves were made showing the classifier performance as the PCA dimension increases.

Models A wide variety of classifier models is considered for each of the data representations. The models were chosen to represent the distinct techniques and types of classifiers. The types of classifiers covered are parametric, non-parametric, density-based, and non-density-based models. As some classifiers may perform better than other in specific parts of the feature space, also a combined classifier approach is considered. For this problem, only the stacked mean approach to combining classifiers is applied, as other ways of combining classifiers did not show significant difference in results. A brief description of the classifiers used is given in the following list.

Parametric, Density-based Models

1. *Linear classifier*: The Linear Bayes Normal Classifier *ldc* is a simple parametric classifier which assumes that the data is according to a Gaussian distribution. The parameters to be estimated are the mean μ and the co-variance matrix Σ. Though PRTools guide mentions that regularization parameters R=0, S=0, M=NaN yields best results generally, we use R=0.5, S=0.5, M=NaN so as to prevent overfitting. No rigorous examination was done to choose these values.

2. *Quadratic classifier (qdc)*: Quadratic Bayes Normal Classifier *qdc* is the same as the linear classifier, except that it takes on a non-linear decision boundary. The parameters to be estimated is also the same as the linear classifier. Though PRTools guide mentions that regularization parameters R=0, S=0, M=NaN yields best results generally, we use R=0.5, S=0.5, M=NaN so as to prevent overfitting. No rigorous examination was done to choose these values.

Non-Parametric, Density-based Models

3. *Parzen classifier*: The Parzen classifier *parzenc* is based on the parzen density estimation, which is dependent on the radius hyper-parameter h. This parameter is not to be estimated, but should be optimized to achieve the best performance. In these experiments it was chosen to be 5 as it seemed to perform well, though no rigorous examination was done.

Non-Density-based Models

4. *Fisher classifier*: The Fisher's Linear Discriminant *fisherc* is not based on density estimation, but instead it tries to optimize the Fisher's criterion.

5. *Decision tree*: The decision tree *treec* takes on a splitting criteria. The default splitting criteria has been used which is the *purity*.

6. *Neural network*: Back-propagation trained feed-forward neural net classifier *bpxnc*. In this case a single layer with 30 neurons was used, as it was found to work well while being relatively simple. In the results an analysis is shown of the performance of different neural network types and sizes.

7. *Support vector machine*: The *svc* classifier generalizes very well and doesn't need much training data. A disadvantage is that the kernel and the trade-off parameter has to be optimized which is computationally expensive. In this case the SVM was kernelized by a 3rd degree polynomial kernel.

Combined classifiers

8. *Stacked Mean, (Neural Network + SVM)*: By stacked combining a set of classifiers in the same space are combined by horizontal concatenation. For this project they are trained by the same training sets. A test object to be classified is applied to all classifiers and results are combined.

To get the best performance, it is necessary to tune the models to its input data. This is done in several steps: (1) selecting the appropriate number of training samples, (2) determine the optimal number of features, and lastly (3) to select the optimal hyper-parameter settings for each model. For the first step a learning curve is constructed, which plots the classification errors against different sizes of the training set. This way we can analyze the impact of having more data and compare different classifiers with each other. The second step is done using the PCA, as discussed in a previous section. Lastly, the hyper-parameters of each classifier were chosen as they showed adequate performance on most examples. A more rigorous analysis of these parameters like doing a full gridsearch was not performed however.

Model Validation For the purpose of optimizing the performance of the models, the provided NIST testset is used. The reason for using a test set for this purpose is because tuning the models based on the training data (data that it has already seen), will result in over-fitting and the model will not generalize well. To ultimately test and validate the performance of the classifiers, two different test sets are used: (1) the original NIST testset containing standard scans of hand-written integers, (2) a generated dataset containing scans of our own hand-written integers, which will be discussed in the 'Live test' section that follows.

Live test Handwriting is different and unique for each person. To test the generalization capability of the resulting model, a real live test is performed on self generated handwritten digits. Each of the team members has written 10 samples of each of the 10 digits on a piece grid paper. These digits were scanned to a digital file. The image was processed to remove the gridlines and binarize the image. A script written in C# was used to determine a bounding box around each digit and save this region to a separate file. The representation and classifiers were otherwise kept the same for the live test. One small difference however is the way scaling and PCA is applied. To make sure the classifier trained on the full NIST data understands the data from our handwritten digits, the same scaling and PCA map was applied to the handwritten data as to the NIST data.

3 Results & Analysis

In this section we present the results of the experiments, along with an analysis and interpretation of these results.

3.1 Feature Curves

We use PCA to extract features from the different representations. In order to decide on the best dimension, we plot a graph of number of PCA dimesnions vs. classifier performance. We collected this result only for the neural network classifier. Size of the training set while gathering this result was N=125 samples per class.

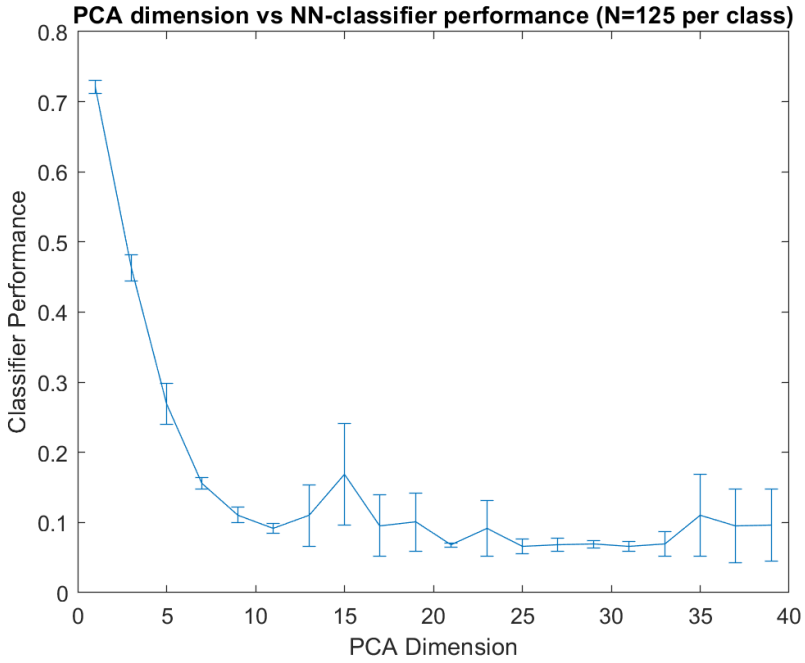

Figure 1: Feature curve: varying PCA dimension for Raw Pixel representation, neural network classifier.

Figure 1 justifies some of the choices for our particular PCA dimensionality. While this curve was only measured fully for the raw pixel representation and neural networks, small experiments showed the behaviour is broadly the same for other representations and classifiers. Based on this curve, we decide to choose a PCA dimension of 30 for the raw pixel representation for training sizes greater than 125 because the curve tends to go upwards after this point. Moreover the variance in the performance is also very less.

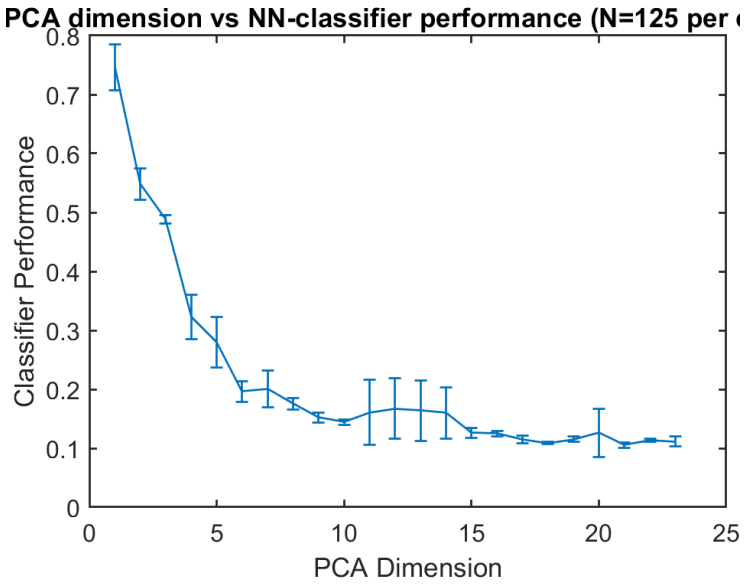

Figure 2: Feature curve: varying PCA dimension for feature representation, neural network classifier.

The feature representation returns 24 features that are listed in the appendix. From figure 2, we infer that PCA reduction is not really needed. Thus for training size greater than 125 samples, no dimensionality reduction is required. Since, dissimilarity representation does not suffer from the *Curse of Dimensionality*, no dimension reduction has been performed.

3.2 Structure of Neural Network

In order to decide on the structure of the Neural network, we trained multiple networks with different base structures and number of neurons within the layers. This was done by looping over a factor scale from 0.5 to 4.5 to vary the number of neurons within a layer. Their performance was recorded and compared with every structure.

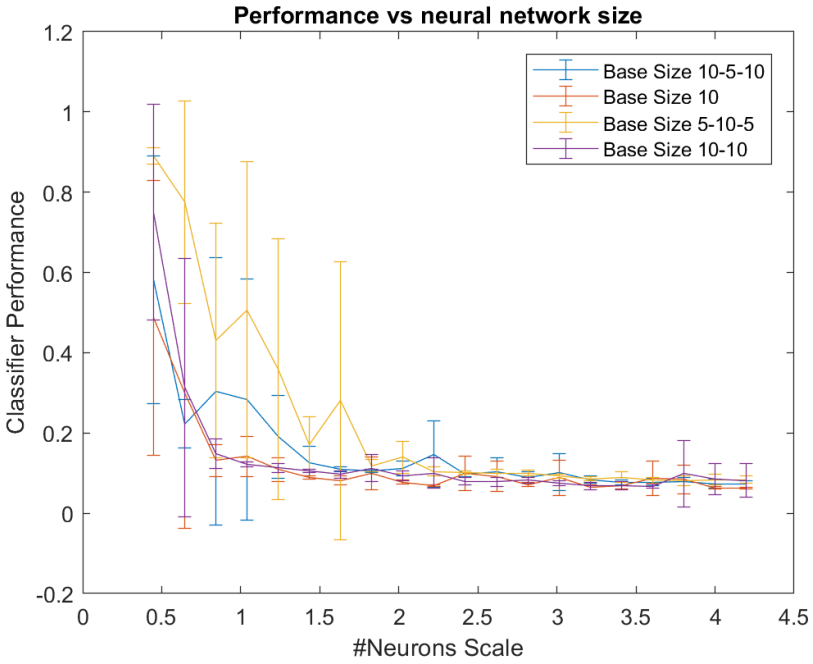

Figure 3: Different neural network sizes and configurations vs their performance.

Figure 3 is an errorbar which shows why for the neural network a single layer network with 30 neurons was used throughout, as this seemed to perform reasonably well while being relatively simple. Moreover, the variance in it's performance is also very less which indicates that it is stable.

3.3 Learning Curves

These curves were plotted in order to analyze the performance of various classifiers by varying training set size for every representation. We could plot training curves only till a maximum training size of 200 for raw pixel and features representations since it is very time consuming to train over data samples more than this. Whereas for dissimilarity representation, we could train for 400 samples since it is faster.

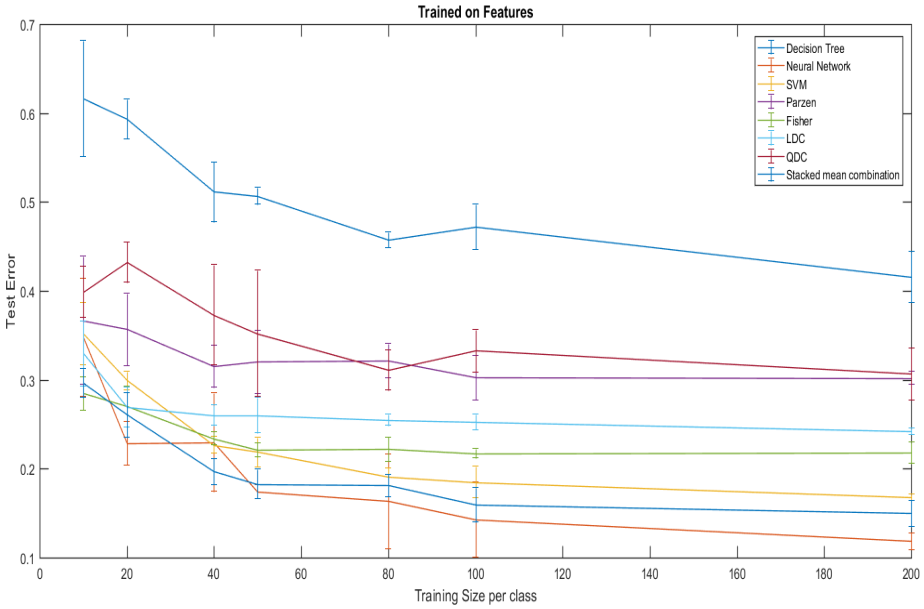

Figure 4: Learning curve for Features representation.

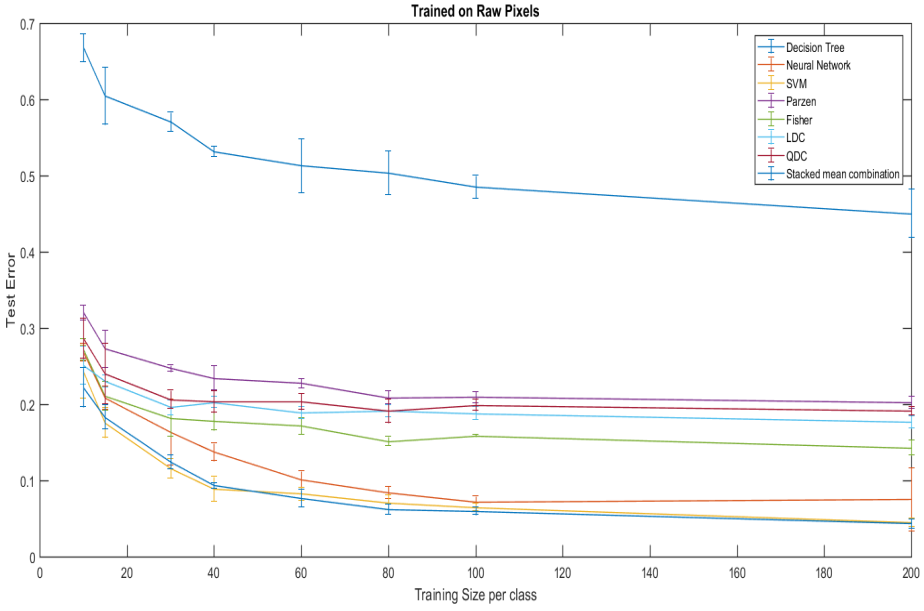

Figure 5: Learning curve for Raw Pixels representation.

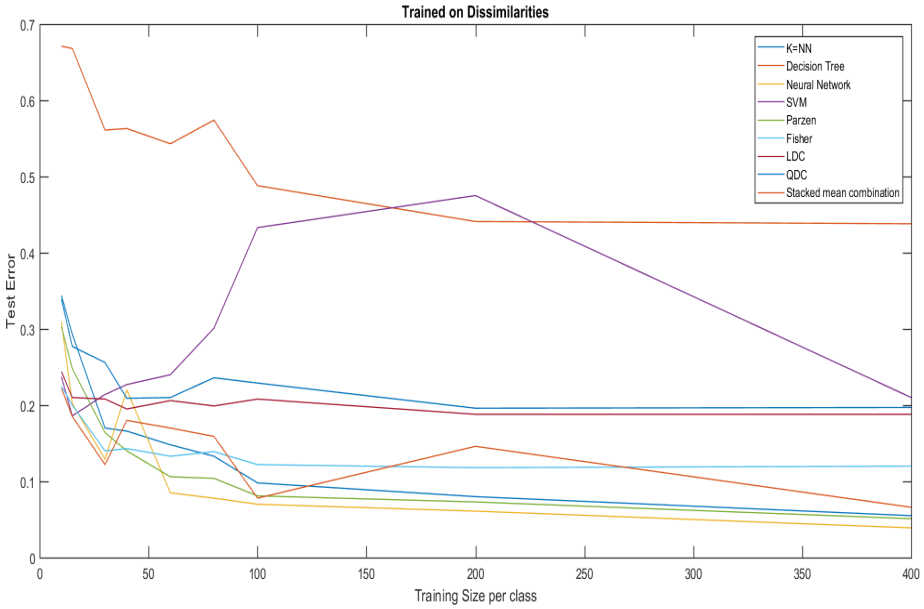

Figure 6: Learning curve for Dissimilarities representation.

On comparing the curves for training over raw pixels and training using features, we observe that raw pixels yield better results as compared to image features for almost all classifiers. The dissimilarities representation outperforms all other representations for neural networks and parzen classifiers. On observing the general trend in the curve obtained, the test error decreases for every classifier as we train over more data samples as expected.

Also it can be observed that almost all learning curves flatten out for larger training set sizes. This indicates that training over more data samples would not help tremendously in further increasing classifier performance.

For different representations different classifiers perform the best. Figure 4 shows that for raw pixels, the simplest representation, neural networks and support vector machines generally perform the best and also a mean stacked combination of the two perform the best. Moreover, the less variance in error shows that these classifiers are stable.

Figure 5 shows that the features representation was generally not very effective - with only a neural network classifier performing close to 10% error. Also, there is a very large variance in their errors, thus the performance is also not reliable for feature representation. A drawback of the use of features is that different objects may have the same representation as they differ by properties that were not expressed in the chosen feature set which results in class overlap [1]. Thus, the features we chose to measure are not effective measures of the image data. One possible cause could be that after resizing the images were too small to determine measures such as eccentricity successfully.

Lastly the dissimilarity representation performs quite well as seen in figure 6.

In this representation a neural network achieves the lowest overall performance of all classifiers and representations at 3.8% error for training over 400 samples.

When comparing learning curves in the different representations it can additionally be seen that the dissimilarity trains faster than the direct representation and more training samples may not be beneficial because the curve flattens out for almost all classifiers. It's also interesting to note that SVM, while generally performing well in raw pixels or features representations, performs quite badly in this representation independently of the training size. Similarly the stacked mean combiner also performs worse though its reduction in performance is not extreme. Additionally, the parzen classifier performs extremely well when trained over the dissimilarities space despite not performing well in other representations.

3.4 Scenario 1: Large data

As described in the introduction, the model for the first scenario is trained on a large data set. At least one result for this system should have a test error rate that is lower than 5% for the system to be considered successful. In this scenario 400 samples per class were used to train each classifier. The table below illustrates the errors obtained for different classifiers trained over different representations.

Table 1: Scenario 1 Results

Classifiers	Representations		
	Raw Pixels	Features	Dissimilarity
Neural Net (NN)	0.063	0.080	0.038
Decision Tree	0.386	0.309	0.415
SVM	0.048	0.13	0.210
Parzen	0.191	0.231	0.050
Fisher	0.144	0.140	0.186
LDA	0.175	0.188	0.183
QDA	0.191	0.239	0.275
Stacked NN+SVM	0.044	$-^1$	0.066

Note 1: Due to time constraints these values were not calculated.

Neural networks yield the lowest error for all three representations. For dissimilarity representation we get the lowest possible error as 3.8%.

3.5 Scenario 2: Small data

In this scenario only 10 samples were used to train each classifier. The table below illustrates the errors obtained for different classifiers trained over different representations.

Table 2: Scenario 2 Results

Classifiers	Representations		
	Raw Pixels	Features	Dissimilarity
Neural Net (NN)	0.194	0.225	0.197
Decision Tree	0.591	0.689	0.783
SVM	0.265	0.229	0.223
Parzen	0.263	0.358	0.251
Fisher	0.231	0.221	0.189
LDA	0.250	0.277	0.219
QDA	0.262	0.446	0.327
Stacked NN+SVM	0.228	0.215	0.216

As expected scenario 2, where only training data from one batch of cheques is available, performs much worse. Still the combined classifier performs best showing performance of under 25% error rate for all three representations.

3.6 Live Test

In this scenario the performance is listed for recognizing our own hand written digits. See appendix for handwritten digit images. The classifiers were again trained on 400 samples each. The table below illustrates the errors obtained for different classifiers trained over different representations.

Table 3: Live Test Results

Classifiers	Representations		
	Raw Pixels	Features	Dissimilarity
Neural Net (NN)	0.414	0.392	0.423
Decision Tree	0.678	0.578	0.713
SVM	0.441	0.364	$-^1$
Parzen	0.482	0.517	0.380
Fisher	0.498	0.494	0.503
LDA	0.565	0.533	0.517
QDA	0.621	0.475	0.593
Stacked NN+SVM	$-^2$	$-^2$	$-^1$

Note 1: Due to older code this value was not recorded.
Note 2: Due to time constraints these values were not calculated.

The table 3 showing the live tests results shows performance is much worse on our hand written digits. It is clear that either the generated live data did not match the NIST dataset well, or that the classifiers did not generalize well. Indeed when evaluating some of the training errors on the best classifiers errors below 0.01 are seen, suggesting the classifiers might be overfitting to the NIST

data. By overfitting less the live-test could perform better at the cost of making the NIST test worse.

Overall, Neural Networks seems to dominate in every representation, and works especially well in the dissimilarity space. It performs well in both scenario 1 and 2 and trains relatively fast. With further optimizations to the neural network size and structure this could be a good classifier to use.

4 Recommendations

The feature "learning" curves reveal there is probably not a lot to gain when using more training data. Most curves are already quit flat at 3-400 samples per class. Only marginal gains would be achieved by increasing the number of samples per class. Instead the classifier hyper parameters could be further optimized. Additionally other feature representations better optimized for the NIST data could be considered. The best performing representation, dissimilarities, only considers simple Euclidian distances currently. Other distance measures might improve the performance.

Given the application domain, it is important to consider reject options. Even tiny error rates when recognizing bank cheques could prove hugely problematic. Luckily, the recommended neural network classifier also outputs posterior probabilities, and so reject options based on some probability threshold could be implemented. Further work would need to be done to determine good reject parameters.

Additionally, the results show combined classifiers performing well in the direct pixel representation. Considering more combination options, and combining methods might further increase performance. It might also be the case that using different classifiers for different digit types (eg, if classifier A. says it is digit 8, we ask classifier B what it thinks) might further improve these results.

Lastly, so far when evaluating classifiers only the classifier performance was used. There are other metrics however, such as computational expense, that could be necessary to evaluate before deploying a system like this in practice. It is clear that SVM while performing well is computationally very intensive. In this report training SVM on 400 samples could take over 8 hours on a desktop grade CPU. If computational resources for training are a constraint, SVM is not likely to be a good option. The neural network based solutions were less computationally expensive, though they could still take some time to train. The parzen classifier in dissimilarity space performed relatively well for it's computational load.

5 Conclusion

This report shows that it is possible to successfully recognize hand written digits for processing bank cheques. Different representations and different classifiers were considered, with neural networks in a dissimilarity representation

ultimately performing the best at about 3% error on the NIST dataset. Additionally the system was live tested, which has revealed potential problems in the training data, method and classifier. This report further recommends some additional routes to explore to increase classifier performance, such as non-euclidian dissimilarities, and further hyper parameter optimization. Lastly the report has considered some other recommendations to implement this system in the real world such as reject options and performance considerations.

6 Discussion

Each classifier type was only tested for one set of fixed parameters. It is possible classifiers would show better results in some scenarios or even all round if there parameters were chosen explicitly for each test.

It's additionally worth noting how much worse the live test worked. This indicates that either the data processing step was not performed successfully, or the classifiers are not generalizing well and possibly overfitting. One factor influencing this result was the way our 1-digit was written. When printing out per-digit errors, it became clear almost all 1's are misclassified. This is likely because we wrote 1 with a cap, whereas in the NIST dataset 1 digits are just straight bars.

A linear dimensionality reduction technique has been used, however it should be noted that other non-linear methods could result in better performance. Lastly, hyper parameters were chosen mostly only based on small trials and not on a more rigorous process to determine suitable values. Optimizing this further could increase the performance of classifiers currently that currently do not seem to work well.

A Appendix

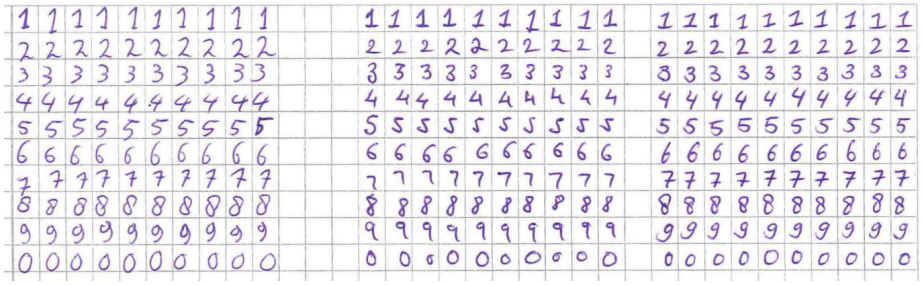

Figure 7: Handwritten digits used in the live test.

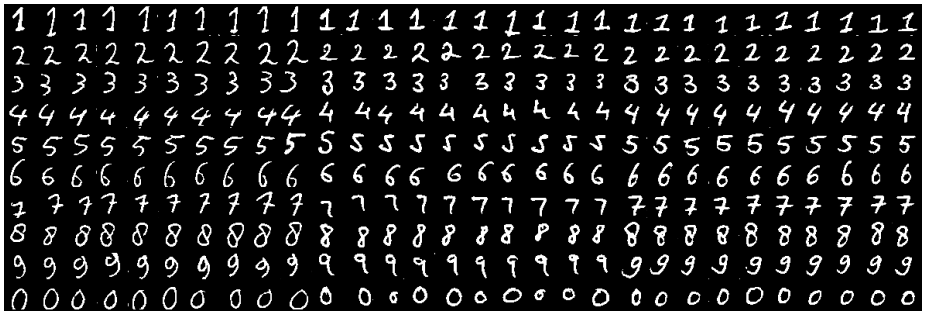

Figure 8: Handwritten digits after thresholding

Area	EulerNumber	Orientation
BoundingBox	Extent	Perimeter
Centroid	Extrema	PixelIdxList
ConvexArea	FilledArea	PixelList
ConvexHull	FilledImage	Solidity
ConvexImage	Image	SubarrayIdx
Eccentricity	MajorAxisLength	EquivDiameter
MinorAxisLength		

Table 4: List of generated features by `im_features` from PRTOOLS

References

[1] Robert PW Duin and E Pekalska. The dissimilarity representation for pattern recognition: A tutorial. Technical report, Technical Report, 2009.

[2] Dick de Ridder David M.J. Tax Ferdi van der Heijden, Robert P.W. Duin. Pattern recognition tools (prtools guide) version 5.3.3 08-mar-2017. http://www.37steps.com/prhtml/prtools.html, 2004.

[3] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition*. Academic Press, 4th edition, 2008.