NIST Handwritten Digits Recognition

A report on the classification of handwritten digits using pattern recognition techniques

Arvid Mildner
TU Delft
Delft, Netherlands
A.P.A.Mildner@student.tudelft.nl

Jakob Stenersen Kok
TU Delft
Delft, Netherlands
J.S.Kok@student.tudelft.nl

Renske de Jong
TU Delft
Delft, Netherlands
R.M.deJong@student.tudelft.nl

ABSTRACT

In this paper, a pattern recognition system is presented and the design process of it. The system is made to be able to correctly classify individual digits of bank cheques when trained once on a large amount of training data (scenario 1) and when trained on smaller batches of cheques to be processed (scenario 2). In the used method design, first preprocessing is performed, then promising image representations and feature reduction are chosen and parameters of classifiers are optimized. Next, the performance results of various classifiers were presented. The most promising results were combined into sets of classifiers with a high performance expectancy. On these combinations, the final error rate of the system was calculated for both scenarios by a benchmark dataset. The classification error for scenario 1 is 4.1% and for scenario 2 is 16.2%, which is within the set limits. Finally, we argue that the proposed system is valid, but could also be improved in different aspects.

KEYWORDS

Machine learning, Pattern recognition, NIST-digits, Digit classification

1 INTRODUCTION

When automatically processing bank cheques, it is important to be able to classify individual digits of the bank account numbers and monetary amount as correctly as possible. In this paper, a pattern recognition (PR) system is proposed to perform well in the following two scenarios:

- **Scenario 1** The PR system is trained once on a large amount of training data (250 objects in each class) and then applied in the field.
- **Scenario 2** The PR system is trained for each batch of cheques to be processed, which means the training set is much smaller and consists of only 10 objects per class.

To be able to design a reliable system, a standard dataset of handwritten digits is used [1]. This digit dataset is put together by the US National Institute of Standards & Technology (NIST)[1] and consists of 2800 images for each of the 10 digits, or classes. These digits are written by volunteers and made ready for use by thresholding the images in black-and-white and automatically segmenting them in images of 128x128 pixels.

The goal of the proposed system is to reach the targeted error or even a lower test error in classifying the digits. For scenario 1, the test error target is 5% and for scenario 2, this is 25%. After designing the PR system, it will be tested by a benchmark dataset that is not seen or used before. For that reason, it can be well used to verify if the predicted performance of the proposed system is actually valid.

This paper is structured as follows. In section 2, the design and methodology of the experiment are elaborated upon which includes a short review on each classifier. Section 3 entails finding the optimal features for each image representation, while section 4 describes how the parameters were optimized for our experiment. Section 5 contains a summary of the results and is followed up by a discussion in section 6. In 7, we perform a live test on own handwritten digits and in section 8 we point out a handful of recommendations on how to improve our system. Finally, an appendix containing the results of our classifier evaluations is found at the end of this paper.

2 DESIGN AND METHODOLOGY

To build a reliable and well-performing classifier, it is key to first review the theory and concepts of the classifiers we are considering for this system. Furthermore, the system design for both scenarios and the methodology is elaborated upon in this section.

2.1 Classifiers

To identify to what class every handwritten digit belongs, we used various classifiers. Classifiers can be sorted by their type: parametric, non-parametric or advanced. From the first type, parametric classifiers, we used the nearest mean classifier, the linear and the quadratic Bayes normal classifier, the Fisher's linear discriminant and logistic linear classifier. The non-parametric classifiers we used are k-nearest neighbor and Parzen classifier and the advanced type consists of e.g. neural networks, support vector classifiers and combinations of different classifiers. All classifiers above have elaborately been explained and tested in various literature [2]. In this section we will clarify them based on working principle and characteristics.

The nearest mean classifier (nmc) assumes a Gaussian distribution for all classes and determines their means. The resulting boundary between the means separates each class from the other with use of the Euclidean distance.

Assuming normal densities, both types of Bayes normal classifiers (or discriminant analyses) define a likelihood ratio based on the mean and covariance of the classes. Unlike the quadratic Bayes normal classifier (qdc), the linear Bayes normal classifier (ldc) assumes the covariance of each of the classes to be the same. The qdc first has to find the covariance matrix and results in a separating surface that is quadratic and is more flexible than ldc. Because of the higher variance and flexibility, qdc is preferred over ldc for larger

[1] http://www.nist.gov/

Delft '19, Jan. 2019, TU Delft, NL
2019.

training sets, since in that case, variance is not a great concern and an assumption on covariance would be reasonless.

Fisher's linear discriminant (fisherc) is closely related to ldc and qdc, however it does not make the same assumptions on the distribution, mean and covariance of the classes. Fisher's discriminant tries to optimize the Fisher criterion by finding the weights such that separability is maximized.

The logistic linear classifier (loglc) measures the relation between the categorical dependent variable of a dataset and the independent variables. It uses a logistic (sigmoid) function to do this which can estimate the likelihood probabilities of x belonging to ω_i.

The benefits of parametric classifiers are that they are rather simple, fast and they need relatively few training data. They can still work well on training data that is limited or not perfect. However, they are constrained. Since they need to assume a functional form, their complexity is limited and their match to the actual unknown underlying function is more likely to be poor. In contrast, non-parametric classifiers can be more flexible and complex, because they do not need (weak) assumptions about the underlying function. As a result, they can reach a higher performance. On the downside, they need more data and parameters to train, which makes them slower. Additionally, they have a higher risk to overfit on the training data.

The k-nearest neighbor classifier (knnc) is a non-parametric and non linear classifier that assigns x to the class ω_i based on the most common label within the k nearest neighbors. Its strength is that it has a very intuitive way of classifying. However, it is not straightforward in the way to choose a value for the parameter k.

The last classifier we individually used, is the Parzen classifier (parzenc). Its working principle is a lot related to knnc, except it uses a fixed volume instead of fixed mass when determining the most common label in its surrounding area. Like knnc, the choice of the value for the parameter - in this case h - is important, but not straightforward.

We will not use the advanced classifiers support vector machine and neural networks in this experiment, since they are too computationally expensive to train whithin our time constraints. Hence, we will not go into details on their working principle and characteristics.

2.2 Preprocessing

For any digital image, preprocessing algorithms are needed to convert the raw data image into proper representations. The aim of preprocessing is to improve the image by mitigating distortions and possibly enhancing features that are relevant for further processing. After studying and experimenting mostly by trial-and-error, the following preprocessing was settled upon.

- im_box([],2,2)
- im_resize([],[20 20])

Im_box is used to put the objects in a square box, enclosing them in a rectangle. Next, im_resize is intuitively just resizing the images according to how many pixels you desire. For this experiment, 20x20 representations was used. Again, multiple other preprocessing methods were considered, for instance rotation, but after inspecting preliminary error-rates we concluded to only include the former.

2.3 Image representations

There exist multiple methods for representing an image such that each image is portrayed in a data-format that is suitable for training classifiers. Firstly, we are considering raw-pixel representation and a variation thereof, which is two types of pixel based convolutions. This was done drawing ideas from convolutional neural networks, where convolutions of images have proven to work well as features (although the operation of a CNN is quite different). The convolution representations are implemented by ourselves, and work by sliding a window row-wise from left to right and taking the mean of the pixels within the sliding window. In this report, we use sliding windows of 2x2 and 4x4 pixels, which avoids the problem of padding since we are working with a 20x20 pixel image.

In this experiment we are also making use of standard PRTools toolboxes in MatLab called *im_features* and *im_profile* for feature representation, which are computing a handful of properties of an image such as its eccentricity, dimension, standard deviation and gravity. See the PRTools documentation for all the possible features [2] [3]. At preliminary stages of this experiment, the package *im_moments* was also used for image representation - which is another PRTools toolbox used for computing different moments of an image. However, as we struggled to get the error rate below 80% by applying this image representation, it was not included in any of our classifiers anymore.

2.4 Methodology

Before testing classifiers for both scenarios, the preprocessing described in section 2.2 was applied and dimensionality was reduced for all the image representations (both for performance and computational purposes). In other words, only the optimal subset of features was selected before running tests. It was not obvious whether feature selection algorithms or PCA feature extraction would yield the best results. Consequently, both methods were tested, and a combination thereof. As for feature selection, the algorithms are computational heavy, especially for the high dimensionality image representations. Feature selection was therefore only considered for *im_profile* and *im_features*.

Furthermore, the error rate and standard deviation for each classifier was then derived for all the image representations after dimensionality reduction. For the knnc and Parzen classifier, only those parameters that we - at preliminary stages of the experiment - found to be providing the best result, were tested. The error and standard deviation estimates are derived from a 10-fold crossvalidation. After conducting tests for all the simple classifiers on all the image representations, the top performing classifiers for each representation were chosen for further combining the classifiers. Considering that it is a combinatorially infeasible task to show all possible combinations of classifiers and combining rules, we settled for only presenting a handful of the results in the final report. Subsequently, some decisions are based on preliminary tests and intuition. The classifiers were combined according to different combining rules/strategies and the best result is ultimately tested against the NIST-eval benchmark.

[2] http://37steps.com/prhtml/prtools/im_features.html
[3] http://www.37steps.com/prhtml/prtools/im_profile.html

3 FINDING OPTIMAL FEATURE SUBSET

In this section, feature selection and extraction are discussed. The goal is to find the optimal reduction solution for each image representation. After exploring both approaches, a brief conclusion is provided with the final solution.

3.1 Feature selection

We choose to use ldc when training on objects with subsets of the features as this classifier is a simple and fast linear classifier and should give a good enough estimate. As aforementioned, it is not feasible to do a feature selection for the raw pixel data (400 features) as this would require very heavy computation. The same goes for the 2x2 convolution (100 features). We could do feature selection for the 4x4 convolution as this gives us 25 features from a 20x20 pixel image. However, we do not think it makes sense to use feature selection on these pixel based approaches, because that will effectively make us use only certain parts of the image. We already consider our preprocessing boxing function to define what part of an image we see as interesting. These are the two reasons why we do not use feature selection on the representations based on raw pixel, 4x4 convolution and 2x2 convolution.

The results from different feature selection algorithms on the *im_profile* and *im_features* representations are shown in figures 1 and 2. In both figures, solid lines correspond to the true error, while dashed lines correspond to the apparent error. The red colored lines correspond to the errors when using backwards selection, the black lines correspond to a random permutation on all the features, the blue lines correspond to a forward selection scheme and the pink lines correspond to an individual feature selection.

Feature selection graph of *im_profile* representation

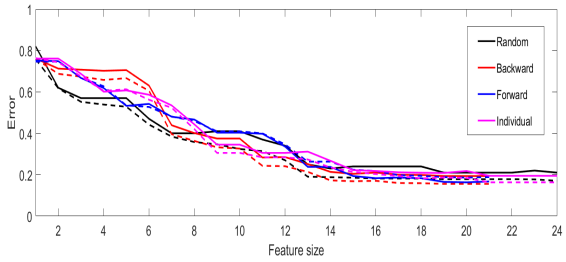

Figure 1: This graph shows the error rate of ldc for each number of features using different types of feature selection on the *im_profile* representation. Since the true error was lowest for the blue line, we chose to select the features from the forward selection scheme. We can observe that after 15 features, the classifier does not improve its error rate enough to have benefit from the other features, thus we omit the last 17 features.

3.2 PCA feature extraction

In this report, we ended up using PCA (Principal Component Analysis) extraction on all the image representations used. PCA is a method to reduce the feature dimensionality while maintaining as much of the variances as possible in the features. To put it briefly, PCA analysis works by learning a linear mapping from the features

Feature selection graph of *im_features* representation

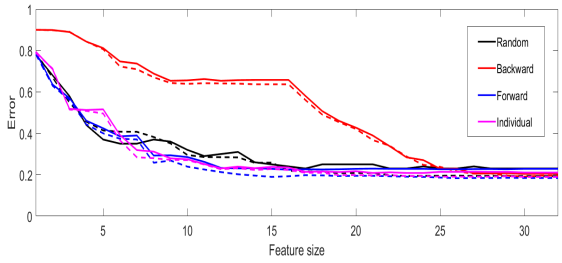

Figure 2: This graph shows the error rate of ldc for each number of features using different types of feature selection on the *im_features* representation. Drawing from these results, we see that the best performing selection scheme is individual selection, because it shows to have the lowest true error. It seems to be good enough to only use the first 14 features, since the individual selection scheme does not improve a lot by adding more features.

of the training set that maps the features to a lower dimensional subspace where we hope we can still make good classifications. This mapping is then used on the test set features before sending them to the trained classifier. When doing PCA analysis, we get the eigenvectors of the feature subspace that point in the best directions, and we can choose as many of these as we want. Oftentimes, eigenvectors are chosen such that 90% of the original variance is preserved. We tested this approach, but it did not render good results which is why we resorted to explicitly stating how many eigenvectors we wanted to use. Another point worth mentioning is that PCA is sensitive to scaling, which is why we also did some tests (not shown in this report) with this. However we could not get any improvements by doing this, in fact it made the classification worse. This led us to not using scaling normalization at all.

It is infeasible for us to optimize how many eigenvectors to keep for each image representation and show all the preliminary tests performed for this, but in the end we decided to use the PCA dimensions in table 1 since it worked well enough for the purpose of this report.

Table 1: Dimension of subspace after PCA reduction

Image presentation	PCA dimension kept
im_features	12
im_profile	12
raw pixels	50
2x2 conv	20
4x4 conv	10

3.3 Conclusion on feature reduction

In the previous sections, we described two ways to reduce the feature dimensionality and discussed what we saw. We performed some preliminary testing using different combinations of feature selection and feature extraction using PCA. We also tried doing feature selection followed by PCA analysis. In the end, we settled

for only using the PCA extraction with the values mentioned in table 1 as it rendered good enough results to reach the designated bench mark target. We got the impression that feature selection never outperformed PCA and that combining them did not give better results than only using PCA.

Furthermore, the same parameters (i.e. dimensionality after PCA extraction) were used for both scenarios. The rationale of this decision was that there exists in general less ground to make proper valuations when having less training data, and since we are operating with the same features, it seemed sensible to generalize on the bigger training set. Conclusively, the parameters that were attained for the scenario with more training data were applied to the other scenario as well.

4 OPTIMIZING PARAMETERS

Some classifiers have parameters that might influence the performance on a certain dataset. Hence, the classifiers that contained these parameters were identified and an effort of optimizing the parameters was done. In k-nearest neighbours (knnc) you have to specify how many of the adjacent feature points you want to look at when deciding the class label.

Regarding parzenc, you want to adjust the window size. The window size parameters of parzenc were optimized for every image representation according to the log-likelihood criterion. Figure 3 is showing the likelihood when applied to the Parzen density for raw pixels, which aims to minimize the error by optimizing the window size. From the figure we can see that the log-likelihood is highest for a window size of 0.25. The same procedure was conducted for the other image representations, where the results can be found in table 4 in appendix A.

Furthermore, the knnc parameter was optimized from solely inspecting the results from a 10-fold crossvalidation for each image representation, where the results can also be found in table 4. The resulting parameters we got from training and testing on the 250 object per class case were also used when training and testing on the second scenario with 10 objects per class, with the rationale being that trying to adjust these parameters to maximize performance while building a classifiers on 10 objects per class would simply overfit the model and provide a biased answer.

Log likelihood plot by different Parzen window sizes for raw image representation

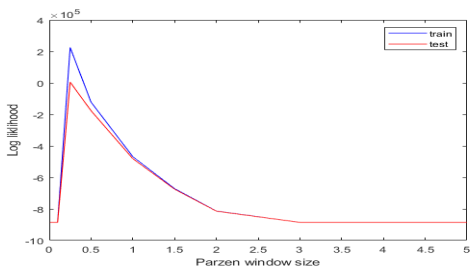

Figure 3: Optimizing Parzen window size.

5 RESULTS

The results obtained in this experiment are shown in the appendix. Appendix A contains the table for optimized values for knnc and Parzen classifiers. Appendix B shows the resulting classification errors from the crossvalidation runs. Whenever we are referring to a result or a table, the appendix should be looked up.

5.1 Elaboration on results

Table 5 shows the results from a 10-fold crossvalidation iterated twice for scenario 1 for every classifier. The qdc has the lowest error rate for all image representations. In table 6, different classifier combinations are displayed - as mentioned in section 2.4 - starting with a combination of only qdc in every representation. Finally, in table 7, the results of a 10-fold crossvalidation with 10 iterations for all proposed classifier combinations are shown.

Tables 8, 9 and 10 are arranged in the same way, but contain results for scenario 2. The best performing classifier for every representation in scenario 2 is not qdc, but ldc.

5.2 Final classifier choice and *nist_eval* testing

After the results were obtained, we chose the classifiers α_6 and β_1 for scenario 1 and scenario 2, respectively. The following results were obtained on *nist_eval*.

Classifier	Error	Training size
α_6	4.1%	2500 objects
β_1	16.2%	100 objects

Table 2: Table showing the classification error (ϵ) when our chosen classifiers for both scenarios are tested using *nist_eval*.

6 DISCUSSION ON RESULTS

To obtain the results in table 6 and table 9, we performed a 10-fold crossvalidation and ran it twice. This gave us a mean and a standard deviation per combination of classifier and representation. We must emphasize that this standard deviation is not entirely reliable, since it is taken over only two values. However, it gave us more information than only one run would. For example, it could happen that the training fold accidentally contains only 9 classes and misses one of the digit classes, which would not make it able to train on that digit. Because we run the validation twice, we could recognize this by surprising high standard deviations. Performing more than two runs for every combination would take too long, which is something we could not afford at this stage of the project. The intention of this validation is only to recognize classifiers that seem promising for further exploration when combining classifiers.

From table 5, we can see that the qdc classifier has the lowest error rate for every data representation for scenario 1. This can imply that the assumption of normal distributions hold. Since qdc outperformed ldc, one can conclude that the assumption of equal covariance matrices does not hold for the data distribution - which in turn - implies that people are more likely to write specific letters more differently than other. However, simply combining the qdc classifiers from each representation with each other did not result in the lowest error overall. The reasoning is simple, each

classifier contributes differently and when combined, they might complement each other and provide better results. α_6 resulted as the best performing classifier, which comprises of both parametric and non-parametric classifiers.

A high variance can be observed in the results of each classifier for the different image representations. This is in line with our expectation. On the other side, nmc, parzenc and knnc in particular seem to exhibit very low accuracy for *im_features* compared to the other classifiers. These classifiers are prone to feature scaling, and the high variance can perhaps be explained by this. An attempt of trying to improve accuracy could be to rescale and normalize the features such that it will not be too affected by high-scaled values in some feature dimensions.

We also saw that we got better results when we omitted the feature representation given by our 4x4 convolution. This may be explained by the fact that it gives a representation that is already encompassed by the 2x2 convolution and hence it can only lead to overfitting or conflicting with the 2x2 convolution features.

It is interesting to notice the remarkable difference in performance between qdc and ldc in scenario 2, as qdc was the best performing classifier in scenario 1. In general, one should be careful when making too haste conclusions about the rationale of such observations, as it can be contingent in nature and needs to be tested more extensively. However, a plausible reason can be that for such small training sizes, simplicity might be key, hence assuming equal covariances might be more clever. When adapting too much towards a small training set, the chance of overfitting increases.

Conclusively, both parametric and non-parametric classifiers yield good results, but they are case dependent. The non-parametric classifiers in this experiment needed scaled and normalized features to exhibit good performance.

Our final results using *nist_eval* from table 2 resemble the predicted results from the 10-fold crossvalidations in tables 7 and 10 very well (for α_6, 4.1% against 4.1% and for β_1, 12.6% against 16.2%). The larger difference in β_1 can be explained by the smaller size of the training set in scenario 2, which results in a larger variance of the classifier performance. Thus the error of β_1 depends more on the test set.

7 LIVE TEST

We implemented a scanner of handwritten digits and tested our best classifier α_6 on this data to see how it would perform in a real scenario (i.e when the digit images came from another source than NIST). There are essentially two basic choices that have to be made. The first is to determine how to gather the handwritten digits, and the second is how to make a reliable scanner. This will be described in the following sections.

7.1 Creating the test set for live testing

We created 10 different image files containing 10 digits from each digit class to be able to automatically label the data in a later stage. The image files were made by creating a 400x200px black background in MsPaint and drawing by hand the 10 digits per file using the touchpad of a laptop. We used a completely white color and drew with a pixel width of 8px wide. In figure 4, we show one of the files for the number 5.

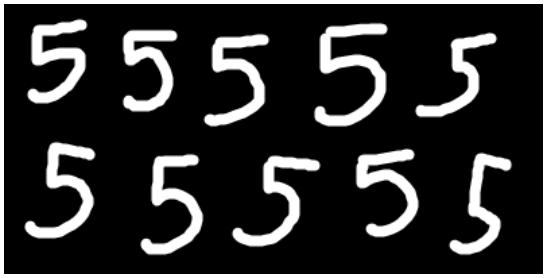

Figure 4: This is one of the test images that was given to our scanner to extract the individual digits.

7.2 The scanner

To implement the scanner, we chose to use the built-in MatLab functions *bwconncomp* and *regionprops*. Using *bwconncomp*, we can find the connected components of the png images. For every found component we write it to a new binary image to isolate it from the other blobs. Then, the function *regionprops(img,'BoundingBox')* can be called to find the bounding box to crop the image correctly. This may seem unintuitive, because *regionprops* should be able to find the boundingboxes without first using *bwconncomp* to isolate the individual blobs, however we could not get this to work rightaway. The scanner code can be seen in appendix C. Then follows simple preprocessing to get the images we need. We expected our scanner to not find all the images in the test set, but it actually turned out to have 0% error in finding digits, i.e all of the 100 digits were scanned correctly (note that this is **not** the classification error).

7.3 Labeling and testing

To test the system, the images were labeled and tested using our best classifier in scenario 1, α_6. We used the same feature representation types that were used when running *nist_eval*. The result is displayed in the following table.

Classifier	Error(ϵ)	Training size	Test size
α_6	73%	2500 objects	100 objects

Table 3: Table showing the classification error (ϵ) when we tested our classifier α_6 on our scanned dataset.

7.4 Discussion on the live test

The error we retrieved when using α_6 as the classifier, was higher than we expected. Perhaps the NIST dataset differs too much from our images, which makes our classifier that is optimized on the NIST dataset useless on our own images. For example, when looking at a sample of the NIST dataset, we can see that there are no 'hooks' and 'pedestals' used in the 1's. In our images however, we did use those characteristics. This could be a good assumption, since all the 1's were misclassified. The system classified almost all digits as a 3, 6 or 9 and all 4's, 5's and 8's are misclassified. Another contribution to the bad result is that we only used 100 test objects now, which is too little to actually state that the resulting error is resembling the true error.

In a real scenario, it would be better to scan and train digits from the cheques the bank already received. It's obvious that the optimal solution would be to have the same scanning method on our training data as we have on the test data.

8 RECOMMENDATIONS

After we finished building the system, we are convinced that there are still some ideas left that could possibly enhance the system, that we due to several limitations did not consider in our implementation. These ideas are elaborated upon in the following subsections.

8.1 More training data

It is always good to have a lot of training data, however, we think that the amount of data that already is available is enough if not more data can be gathered cheaply. However, all the training data that we already have should be used when building the real classifier for scenario 1 now when we know a good performing classifier on the data that we have.

8.2 Time constraints

We think that time constrain does not need to be considered in scenario 1 when a classifier has been chosen to be used. For the type of classifier we have proposed in this report, the classification is generally fast but the training is slow. For scenario 1 we can train it for a week if need be since it only has to be done once. It will still perform fast classifications. For scenario 2 the training time is way more relevant. Here we recommend not to use the methods with long training times (mostly SVM and various neural networks) since we need to retrain the classifier for every batch of cheques.

8.3 Other classifiers

For scenario 1, there should be no real limitation on training time for a classifier since we only would have to train it once. Its classification time will be constant even if it is trained for a long time. This means it could be useful to explore how the powerful classification coming from SVM and different kind of neural networks could improve the accuracy here. We have seen that recent research in convolutional neural networks have shown very good results. For example, in this simple example[4] using Tensorflow, which is a python library for CNNs, the authors achieved an error rate of 2.7%. There is also a technique within CNN:s called *transfer learning*. This may prove to be useful in CNN:s for scenario 2, since the technique uses an already partially trained CNN and only trains its last fine-tuning with the new data. The idea is that the first training epochs has found good ways to represent image features that is very general, and that the classification task of the network is trained in the final epochs. As mentioned earlier, training an entire CNN from scratch in scenario 2 is unfeasible given the time required for this process with commodity hardware. It might also be interesting to see if the digits can be classified into different groups in a preliminary classification and then be further classified by more special-trained classifiers. By inspection, we could imagine that digits like 1, 7 and 9 may be harder to distinguish from each other since they all comprise of a long line and something at the top. A binary classifier could then be trained saying if the digit should belong to the group of 1, 7 and 9 or the group consisting of the other digits. Then another classifier could be trained on distinguishing only 1, 7 and 9 from each other. This is just an idea and may lead to bad results if the preliminary classification is not performed with very good accuracy. Another approach could be to simply train individual classifiers on each individual digit. Then, a voting heuristic could be used that the most certain classifier about the type of digit determines the classification. This could be an interesting approach since it also makes it available to very clearly see the advantage of using the prior distributions of the digits on the bank cheques as discussed in the section on use-case error.

[4]https://www.tensorflow.org/tutorials/estimators/cnn

8.4 A word on features

It could indeed be interesting to look into other features. However, this may also lead to worse results (as we saw in our case with the 4x4 convolution). The time spent on finding more features may however not be worth it. Rather, we think that focusing on improving the classifiers is the most worthwhile approach to reducing the error rate. At some point, the curse of dimensionality comes in. This is a heuristic that says that the error rate will eventually increase when more features are added, since the multidimensional space where the features live grows so fast that distance measurements cannot really distinguish between objects anymore.

8.5 Reject options and robustness

The bank has to decide what it wants to do when the classifier is unsure of a number. The classifiers can be built in such a way that they say how sure they are of the type of digit when scanning a handwritten digit. Then we could build the algorithm such that it rejects a cheque if it's too unsure about the type of digit (by using a threshold). We recommend this approach since it is so important that the digits are correct. However, it may lead to many false negatives i.e that correctly classified digits are still rejected. To further explore the reject option, there should be a cost analysis that finds the cost of rejecting and misclassifying respectively. Using this data the optimal reject percentage that minimizes the cost can be found using the rejection curve.

8.6 Use-case error

As of now, the error definition is defined as the regular classification error when the numbers are assumed to come from the same prior distribution. However, we can assume that this is not the case for the bank cheques. Without further investigation, we can assume that the digit 0 are way more frequent then other digits in numbers representing money values. If a proper prior analysis of the frequencies of digits on the bank cheques were made, we could use this in the classification. For example, if 0's are much more common than the digit 5, an uncertainty in the classification whether it's a 5 or a 0 could be weighed by the prior probability that it is a 5 or a 0 in the first place.

REFERENCES

[1] Afshar S. Tapson J. van Schaik A. Cohen, G. 2017. EMNIST: an extension of MNIST to handwritten letters. (2017). http://arxiv.org/abs/1702.05373

[2] Sergios Theodoridis Sergios Theodoridis Konstantinos Koutroumbas. 2008. *Pattern Recognition* (4th. ed.). Academic Press.

APPENDIX

A OPTIMIZED PARAMETERS

	im_feature	im_profile	im_raw	im_conv(2x2)	im_conv(4x4)
parzenc	0.8	0.05	0.25	0.1	0.05
knnc	5	6	1	3	4

Table 4: Optimized values for the knnc and Parzen classifier.

B RESULTS FOR SIMPLE CLASSIFIERS (PCA-BASED)

B.1 Scenario 1

Classifier		im_features	im_profile	raw pixel	2x2 conv	4x4 conv
qdc	ϵ	0.184	0.167	0.0594	0.064	0.1190
	σ	0.0011	0.0008	0.0020	0.0011	0.0014
ldc	ϵ	0.2072	0.2248	0.1166	0.1340	0.1794
	σ	0.0006	0.0006	0.0020	0.0000	0.0008
loglc	ϵ	0.1956	0.2168	0.1004	0.1005	0.1540
	σ	0.0011	0.0017	0.0014	0.0000	0.0006
nmc	ϵ	0.7322	0.3346	0.1846	0.1908	0.2334
	σ	0.0031	0.0048	0.0020	0.0011	0.0003
fisherc	ϵ	0.2404	0.2702	0.1446	0.1654	0.2022
	σ	0.0006	0.0003	0.0020	0.0008	0.0003
parzenc	ϵ	0.4146	0.1898	0.0790	0.0946	0.3434
	σ	0.0008	0.0037	0.0014	0.0020	0.0003
knnc	ϵ	0.4122	0.1766	0.0792	0.0924	0.3330
	σ	0.0037	0.0014	0.0006	0.0017	0.0048

Table 5: Table showing the classification error (ϵ) and the standard deviation (σ) when 10-fold crossvalidation was run on 2500 training objects (250 objects per class) for the above classifiers with two iterations.

Classifier	im_feature	im_profile	im_raw	im_conv(2x2)	im_conv(4x4)	Combiner rule
α_1	qdc	qdc	qdc	qdc	qdc	prodc
α_2	loglc	knnc([],6)	parzenc(0,25)	qdc	qdc	prodc
α_3	ldc	knnc([],6)	qdc	parzenc(0.1)	-	prodc
α_4	ldc	parzenc(0.05)	qdc	qdc	-	prodc
α_5	loglc	knnc([],6)	qdc	parzenc(0.1)	-	prodc
α_6	qdc	knnc([],6)	qdc	parzenc(0.1)	-	prodc
α_7	qdc	qdc	qdc	qdc	-	prodc

Table 6: Description of the combined classifiers used in scenario 1.

Classifier		Results
α_1	ϵ	0.0459
	σ	0.0006
α_2	ϵ	0.0472
	σ	0.0022
α_3	ϵ	0.0434
	σ	0.0009
α_4	ϵ	0.0486
	σ	0.0019
α_5	ϵ	0.0436
	σ	0.0010
α_6	ϵ	0.0414
	σ	0.0017
α_7	ϵ	0.0465
	σ	0.0014

Table 7: Table showing the classification error (ϵ) and the standard deviation (σ) when 10-fold crossvalidation was run on 2500 training objects (250 objects per class) for the various combined classifiers described in table 6 with 10 iterations.

B.2 Scenario 2

Classifier		im_features	im_profile	raw pixel	2x2 conv	4x4 conv
qdc	ϵ	0.4950	0.5100	0.9000	0.6250	0.4850
	σ	0.0071	0.000	0.00	0.0071	0.0071
ldc	ϵ	0.1950	0.2150	0.200	0.1500	0.2050
	σ	0.0212	0.0071	0.0283	0.0141	0.0071
loglc	ϵ	0.3200	0.3700	0.3750	0.3050	0.2500
	σ	0.0141	0.0707	0.0354	0.0071	0.0141
nmc	ϵ	0.7800	0.3050	0.2400	0.2150	0.2600
	σ	0.000	0.0141	0.0071	0.0071	0.000
fisherc	ϵ	0.2100	0.2950	0.2400	0.2500	0.2450
	σ	0.0141	0.0212	0.0283	0.000	0.0071
parzenc	ϵ	0.5500	0.3200	0.2150	0.2100	0.3500
	σ	0.0071	0.0283	0.0071	0.0141	0.0424
knnc	ϵ	0.5900	0.4950	0.2100	0.3400	0.3350
	σ	0.0424	0.0212	0.0141	0.0283	0.0071

Table 8: Table showing the classification error (ϵ) and the standard deviation (σ) when 10-fold crossvalidation was run on 100 training objects (10 objects per class) for the above classifiers with two iterations.

Classifier	im_feature	im_profile	im_raw	im_conv(2x2)	im_conv(4x4)	Combiner rule
β_1	ldc	ldc	ldc	ldc	-	product
β_2	fisherc	ldc	knnc([],1)	ldc	-	product
β_3	ldc	ldc	ldc	ldc	ldc	product
β_4	fisherc	ldc	knnc([],1)	ldc	ldc	product

Table 9: Description of the combined classifiers used in scenario 2 (when few training samples were available).

Classifier		Results
β_1	ϵ	0.126
	σ	0.0135
β_2	ϵ	0.119
	σ	0.0191
β_3	ϵ	0.13
	σ	0.0221
β_4	ϵ	0.14
	σ	0.0082

Table 10: Table showing the classification error (ϵ) and the standard deviation (σ) when 10-fold crossvalidation was run on 100 training objects (10 objects per class) for the various combined classifiers described in table 9 with 10 iterations.

C THE MATLAB FUNCTION CODE FOR THE SCANNER

```
1
2  function img_objects = extract_digits_from_img(filename)
3  A = imread(filename);
4  %If input is RGB we need to convert it, but it wont work if the input is
5  %not RGB hence the try-catch.
6  try
7  A = rgb2gray(A);
8  catch
9      A=A
10 end
11
12 CC =bwconncomp(A);
13 B = zeros(size(A));
14 img_objects = [];
15 for i = [1:CC.NumObjects]
16     pixel_idx = cell2mat(CC.PixelIdxList(i));
17     B(pixel_idx) = 1;
18     rect = regionprops(B,'BoundingBox');
19     curr_img = im2bw(imcrop(A,rect.BoundingBox)*im_box([],2,2)*im_resize([],[20  20]));
20     try
21         img_objects = cat(3,img_objects,curr_img);
22     end
23     B = zeros(size(A));
24 end
```