IN4085 Pattern Recognition Final Assignment: Classification of handwritten digits

Vasileios Serentellos
TU Delft
Student number: 4843789
V.Serentellos@student.tudelft.nl

Rafail Skoulos
TU Delft
Student number: 4847482
R.Skoulos@student.tudelft.nl

Achilleas Vlogiaris
TU Delft
Student number: 4875974
A.Vlogiaris@student.tudelft.nl

1 INTRODUCTION

The automatic recognition of handwritten digits constitutes a major research field in pattern recognition due to its several application areas ranging from line handwriting recognition on computer tablets and zip codes identification on mail for postal mail sorting to processing bank check amounts and in general numeric entries in forms filled up by hand. There is a plethora of challenges need to be faced when someone attempts to solve this problem. Some of them are the different size, thickness, orientation and position of the handwritten digits, while the wide variety of ways that a digit can be written depending on the handwriting of each person should also be taken into account. In addition, the similarity of some digits like 1 and 7 or 5 and 6, has a negative effect on the automated classification.

An important application area is the automatic bank cheque where pattern recognition techniques are utilized to classify individual digits in bank account numbers and the monetary amount. The chance of a noteworthy reduction of the manual effort needed for this task and the relatively inexpensive CPU power nowadays, have rendered this area so popular. Nevertheless, there are many difficulties which should be confronted in order to implement such a system. The main one is the high accuracy rate that should be achieved. Given the significant negative impact that the misclassification of a monetary amount implies, the reliability of such a system becomes of high importance. This means that the classification error in every single digit should be extremely low, even lower than 1%.

The objective of the current assignment was the implementation and evaluation of several different pattern recognition techniques for the automatic classification of handwritten digits. We have created the optimal system for this purpose for two scenarios. In the first of them, the amount of training data was large (at least 200 and at most 1000 samples per class) while in the second one it was much smaller (10 samples per class). Also we were advised to choose among three different types of representations of the input data for each of the two scenarios, meaning features, pixels and dissimilarities. For that reason, we explored a plethora of different classifiers used for multiclass classification problems and experimented with several optimization techniques, in order to find the optimal solution for each case.

In particular, for each of the two scenarios a different process was followed to find the combination of features and classifiers which could provide us the best classification performance. We were asked to achieve less than 5% test error for the first scenario, while for the second scenario the test error had to be lower than 25%. A plethora of experiments were conducted, in which we evaluated a great number of classifiers on each one of the representation categories. Furthermore, we tried some dimensionality reduction techniques, such as Principal Component Analysis (PCA), Feature Selection and others, so as to reduce both the classification error of the system and its time complexity. In addition, several graphs, like learning and feature curves, were plotted in order to gain a better insight of the parameters that play an important role in the classification. Subsequently, the trained classifiers that achieved the best results were applied on a self-designed live set of actual handwritten digits.

As far as the implementation of the system is concerned, *Matlab* was used and especially the *PRTools* toolbox. This toolbox is used for fast prototyping and offers many routines which represent a basic set covering largely the area of statistical pattern recognition [1]. It contains many simple or advanced classifiers used in pattern recognition, as well as files and routines for the easy evaluation and comparison of classification algorithms. Regarding the dataset, a standard dataset of handwritten digits provided by the US National Institute of Standards & Technology(NIST)[1] was used. This dataset consists of images of handwritten digits between 0 and 9 (10 classes) represented as black-and-white. Every class contains 2800 images.

The structure of this report is organized as follows: In the 'Preprocessing' section, each of the three representations of the input data is described, along with the way the features are extracted from the images of handwritten digits. Apart from these, in the 'Dimensionality Reduction' subsection all the dimensionality reduction techniques used in our system are analysed. In the 'Proposed Classifiers and Results' section , the classifiers, that we experimented with, are presented along with their classification results both in the experiments we conducted and in the final test for each of the scenarios. The conclusion which can be drawn from the results that we achieved is summarised in the 'Conclusion' section. In the 'Live Test' section, the process followed for the test of our system in actual handwritten digits and its results are illustrated. Finally, in the 'Recommendations' section, steps that could possibly improve the performance of our system are proposed.

2 PREPROCESSING

2.1 Data Representations

The primary representations used in pattern recognition tasks on real-world objects, like the images of handwritten digits with which we had to cope in the premises of the current assignment, consist of the following 3 types:

- feature representation
- pixel representation and
- dissimilarity representation

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

The feature-based representation constitutes the "traditional" way of representing the objects to be classified in the premises of a pattern recognition problem based on the extraction of individual measurable properties or characteristics of these objects capable of not only characterizing in a similar way objects from the same class, but also discriminating them from objects originating from different classes. A drawback related to the feature-based representation derives from the fact that different objects may possess a similar representation as they could possibly differ by properties not captured by the produced dataset, resulting to a class overlap problem since in some areas of the feature space these objects would be represented by the same feature vectors. A more complete representation could be produced by sampling the objects, which in the case of images would result in the pixel representation. In that representation all the objects could be resized to contain the same number of aligned pixels so that the same pixel in different images would in theory describe objects on the same position. It can be easily seen that pixels are in general less informative comparing to features, yet they may be significantly useful in the absence of sufficiently discriminative features. Finally, the dissimilarity representation constitutes an alternative to both the use of features and pixels and is based on direct pairwise object comparisons. In a more simplistic way, if the entire objects are considered in the comparison, then solely identical objects will produce a zero (or close to zero) dissimilarity measure.

2.1.1 *Feature-based Representation*.

In this representation, several characteristics of the image are extracted and used as features for the training and the testing of the classifiers. Contrary to the other two representations, the features extracted for this representation are only 71. For the extraction of these features we used the *Matlab* image toolbox.

The first 22 features of this feature set consist of properties of image regions which derive from shape measurements[2], and were extracted through the usage of *im_features* function. These are the following:

- **Area**: Number of pixels in the region
- **BoundingBox**: Smallest rectangle containing the region
- **Centroid**: Center of mass of the region
- **ConvexArea**: Number of pixels in 'ConvexImage'
- **ConvexHull**: Smallest convex polygon that can contain the region
- **ConvexImage**: Image that specifies the convex hull, with all pixels within the hull filled in
- **Eccentricity**: Eccentricity of the ellipse that has the same second-moments as the region
- **EquivDiameter**: Diameter of a circle with the same area as the region
- **EulerNumber**: Number of objects in the region minus the number of holes in those objects
- **Extent**: Ratio of pixels in the region to pixels in the total bounding box
- **Extrema**: Extrema points in the region
- **FilledArea**: Number of pixels in FilledImage
- **FilledImage**: Image the same size as the bounding box of the region
- **Image**: Image the same size as the bounding box of the region
- **MajorAxisLength**: Length (in pixels) of the major axis of the ellipse that has the same normalized second central moments as the region
- **MinorAxisLength**: Length (in pixels) of the minor axis of the ellipse that has the same normalized second central moments as the region
- **Orientation**: Angle between the x-axis and the major axis of the ellipse that has the same second-moments as the region
- **Perimeter**: Distance around the boundary of the region
- **PixelIdxList**: Linear indices of the pixels in the region
- **PixelList**: Locations of pixels in the region
- **Solidity**: Proportion of the pixels in the convex hull that are also in the region
- **SubarrayIdx**: Elements of label matrix inside the object bounding box

We also used *im_features* function to extract another five features which are properties of image regions that are generated from pixel value measurements:

- **MaxIntensity**: Value of the pixel with the greatest intensity in the region
- **MeanIntensity**: Mean of all the intensity values in the region
- **MinIntensity**: Value of the pixel with the lowest intensity in the region
- **PixelValues**: Number of pixels in the region
- **WeightedCentroid**: Center of the region based on location and intensity value

The rest features are made up by the central moments, the horizontal and vertical profile (i.e. normalized image projections), the centers of gravity, some statistics (mean, var, min, max, size) and some skeleton-based features (branch, end, link, single) of the images. For the extraction of these features we used the *im_features*, *im_moments*, *im_profile*, *im_mean*, *im_stat* and *im_skel_meas* functions.

2.1.2 *Pixel Representation*.

In this representation the whole pixel domain of the image is used as features, meaning that the feature space is too big and probably the computational and memory cost is too large. To minimize this time consuming classification process and to achieve lower classification error, some image processing techniques were implemented.

Images are often degraded by noises, that can occur during image capture, transmission, etc. Noise removal is an important task in image processing, with its results having in general a strong influence on the quality of the image processing techniques. Several techniques for noise removal are well established in gray scale image processing. The nature of the noise removal problem depends on the type of the noise corrupting the image. In specific, in this assignment we used Morphological image processing, Orientation and Slant Correction image processing techniques.

In the field of image processing several linear and nonlinear methods have been proposed. In the premises of this assignment,

[2]https://nl.mathworks.com/help/images/ref/regionprops.html

we used various *Matlab* built-in image processing functions, which are presented below.

- *imclose*[3]
- *imerode*[4]
- *imdilate*[5]
- *im_box*[6]
- *im_moments*[7]
- *affine2d*[8]
- *imwarp*[9]
- *im_resize*(scaling)[10]

Morphological image processing
Morphological image processing pursues the goals of removing these imperfections by accounting the form and structure of the image. The main morphological image processing techniques that are implemented on this assignment is the "closing". The "closing" technique is a dilation followed by an erosion. In more details, dilation usually uses a structuring element for probing and expanding the shapes contained in the input image, while erosion produces the opposite results. Thus, the combination of these two operations fill the gaps at each number.

Orientation and Slant Correction image processing
The orientation definition of an image and its slant correction can be characterized as one of the most important image preprocessing techniques. The definition of the orientation and the re-calibration of it can provide the handwritten number recognition system with sufficient features. We utilized two different built-in *Matlab* functions (*affine2d*, *imwrap*) and one *prtools* function (*im_moments*), with the last one of them used to obtain the orientation of each image.

Firstly, *im_moments* captures the centroid of the image aiming to centralize the number in the image. In this case, the simple mean of the x and y axes is needed:

$$E(x) = \frac{(x_1 + x_2 + x_3 + \ldots + x_n)}{n_x} \tag{1}$$

$$E(y) = \frac{(y_1 + y_2 + y_3 + \ldots + y_n)}{n_y} \tag{2}$$

The next step is to calculate the central moments of each image by using the centroids, which is conducted through the equation presented below.

$$centr\_moments_{a,b} = \sum_x \sum_y (x - E(x))^a \cdot (y - E(y))^b \cdot f(x, y) \tag{3}$$

Subsequently, the orientation is calculated as it is demonstrated below.

$$orient = \frac{1}{2} arctan \frac{2 \times centr\_moments_{1,1}}{centr\_moments_{2,0} - centr\_moments_{0,2}} \tag{4}$$

This orientation is used as an input argument to the *affine2d* function, defining the affine transformation of the image. The affine transformation can be approximately characterized as the fusion of all 2d geometric transformations that an image can suffer (translation, scaling, shearing, rotation), with the matrices that represent these transformations presented in equation 6). The affine transformation can be represented with a 3×3 matrix and can be seen in equation5. The only limitation of this transformation is that this cannot be characterized as elastic, while all the pixels of the image change accordingly with the same amount distortion. Unfortunately the digits that are written in real conditions suffer from elastic transformations, meaning that each pixel's distortion cannot be linearly correlated with the distortion of the pixel next to it.

Lastly, we used the *imwrap* to apply the affine transformation on the image.

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} s_x & 0 \\ 0 & x_y \end{bmatrix} \begin{bmatrix} 1 & h_x \\ h_y & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{6}$$

After the image processing methods that were explained above, each image was resized with the built-in *im_resize* function rendering it ready to be utilized in the dimensionality reduction and the classification stages.

Concluding, the above image processing operations will provide the handwritten number recognition system with more efficient features (since the pixel representation use the pixels of each image as features for the classifier), images with less noise and with more clear numbers since the gaps are filled, rendering both the principal component analysis and the feature selection method more robust (faster training and better classification performance).

2.1.3 *Dissimilarity Representation*.

As it was mentioned before, the dissimilarity representation constitutes an alternative of the usage of features and pixels in the recognition of real world objects like images. The main idea behind the usage of such a representation lays on the assumption that objects with sufficiently small difference are assigned with the same class label. In specific, a measure that estimates the dissimilarity between pairs of objects (images) is defined, meaning that objects are represented by their pairwise dissimilarities to other objects, either directly computed on the raw data observations or from another representation, resulting to a square matrix with the dissimilarities between all pairs of objects of the examined problem. Any feature-based representation can be converted into a dissimilarity one by using a proximity mapping. In our implementation the Euclidean distance was utilized to produce the dissimilarity matrices from the raw data observations that we were provided through the NIST dataset.

From a more technical point of view the exact preprocessing procedure followed in the case of the dissimilarity representation of the data consisted of the following steps. Firstly, some processing on the images contained in the dataset is applied. In specific, all the empty rows and columns are initially removed from them, while some additional rows and columns are added so that these images are shaped into a square. Finally, the images are resized to a $30x30$ data representation, while some rows and columns are again

[3] https://nl.mathworks.com/help/images/ref/imclose.html
[4] https://nl.mathworks.com/help/images/ref/imerode.html
[5] https://nl.mathworks.com/help/images/ref/imdilate.html
[6] http://www.37steps.com/prhtml/prtools/im_box.html
[7] http://www.37steps.com/prhtml/prtools/im_moments.html
[8] https://nl.mathworks.com/help/images/ref/affine2d.html
[9] https://nl.mathworks.com/help/images/ref/imwarp.html
[10] https://nl.mathworks.com/help/images/ref/imresize.html

added to the outer part of them so that no digit would touch the border of the image. After that, a Euclidean distance based proximity mapping is applied on the processed images through the usage of the *proxm* function so that the dissimilarity matrix representation of the image dataset can be produced. Finally, in order to properly split the dissimilarity-based dataset into a training and a test set the *genddat* function is utilized, since it constitutes a dataset splitting function specially deployed for the case of dissimilarity matrices when the splitting of the dataset should affect the number of its rows as well as its columns.

2.2 Dimensionality Reduction

In both scenarios considered in the premises of the pattern recognition task with which we were asked to get involved, the aforementioned representations of the image data were used as a basis for the classifiers to be trained. As in most of the pattern recognition problems the produced data representations may contain redundant measurements, the inclusion of which in the final dataset used as the input in the classification procedure would result to the generation of both underperforming and computationally costly classification schemes. In order to deal with those two issues, meaning the generation of datasets with high information packing properties and a computationally feasible number of features, dimensionality reductions techniques were employed. There are two primary categories of techniques used in that direction, namely feature extraction and feature selection techniques.

The basic approach followed in the techniques that are associated with the first category consists of the transformation of the given set of measurements to a new smaller set of features possessing a greater amount of classification-related information. The basic reasoning behind transformation-based features is that an appropriately chosen transformation can in parallel exploit and remove information redundancies and lead to the reduction of the necessary feature space dimension. The major difference between the techniques of the first category and those comprising the second one is concentrated on the fact that the feature selection techniques mainly focus on the selection of a subset of the best according to some user-defined criterion features rather than the generation of a new one. The main motivation behind the usage of such techniques, apart from the obvious reduction of the computational complexity, lays also on the fact that although two features may carry a great amount of classification information when treated separately, there might be considerably little gain derived from their combination into a feature vector due to a high mutual correlation, meaning that the considerable increase of complexity does not equally leads to a considerable increase of the informational gain. In a more quantitative description, the aim in feature selection techniques is the selection of a subset of the initial features set leading to large between-class distance and small within-class variance in the feature vector space, meaning that that features should exhibit distant values between the different classes and closely located ones in the same class.

In the current work a feature extraction technique was initially applied on the datasets of each representation followed by the application of a feature selection one to further reduce the dimensions of the data representations and increase their discriminatory capability. In specific, Principal Component Analysis (PCA) was utilized as a feature extraction method in the cases of the features and the pixel representations, while the method of Pseudo-Euclidean Linear Embedding was preferred for the last representation. Finally, a forward feature selection approach was adopted for the final datasets to be generated. More extensive presentation on the dimensionality reductions methods used is provided in the following three subsections.

2.2.1 *Principal Component Analysis*.

Principal Component Analysis is one of the most popular methods for feature generation and dimensionality reduction in pattern recognition. Briefly, this method is designed to perform an orthogonal transformation on a set of observations of possibly correlated variables in order to convert it into a set of values of linearly uncorrelated variables, called principal components, aiming in maintaining as much variance as possible between these variables.

As it was mentioned before, PCA was employed in the cases of the feature-based and pixel representations to both reduce the dimensionality of the datasets (especially in the case of the pixel representation, since that representation consisted of images of size $30x30$, resulting in datasets with 900 features for each object) and possibly improve the performance of the classification schemes to be evaluated since redundant dimensions will be removed after the application of the pca mapping. One of the main difficulties in this approach is to determine the number of principal components to be retained after the application of the mapping. In order to decide on that choice we utilized the *prtools* function *feateval* in order to evaluate the feature set produced by pca mappings created for different numbers of dimensions. In particular, we iterated over the number of dimensions of the pca-derived feature set ranging from a small number of dimensions (50 in our implementation) to the initial dimensionality of the feature set with a step of 50 and applied the *pcam* function with the changing number of dimensions as the input argument. After the evaluation of each pca mapping produced through the forementioned process the number of dimensions that produced the best results were chosen for the final pca mapping of the initial dataset. As it will be shown in the following sections, when the error of each classifier for a different number of features will be presented, most of the classifiers tend to exhibit a stabilizing low error value around 50 dimensions, which is also the case for both of the representations discussed in this section in both scenarios.

2.2.2 *Pseudo-Euclidean Linear Embedding*.

As it was stated above, the dimensionality reduction method used in the dissimilarity representation consisted of the application of a linear mapping from the symmetric, square dissimilarity matrix representing the dataset to be used onto a pseudo-Euclidean subspace such that the measured dissimilarities are preserved. Such a mapping was produced through the usage of the *distools* function *psem*, which takes as an input the square dissimilarity matrix and the dimension of the pseudo-Euclidean subspace on which the given dissimilarity representation should be projected and outputs the linear embedding into that particular space. It can be easily seen that we had to deal with the same problem as in the case of PCA,

since the number of dimensions of the space in which the data representation should be projected has to be decided. In order to deal with that issue we followed the same procedure as that in the PCA case by utlizing the *featureval* function to evaluate the classification performance of each projection of the initial dissimilarity representation to subspaces with a variant number of dimensions. Again, as it will be shown in the following sections, most of the classifiers tend to exhibit a stabilizing low error value for embeddings into 50-dimensional pseudo-Euclidean subspaces, which is the number of dimensions that was selected for the dissimilarity representation in both scenarios.

2.2.3 *Feature Selection*.

After the initial reduction of the dimensions of each data representation through the utilization of the forementioned methods a forward feature selection algorithm was applied on the newly created datasets. As it was suggested above, the main motivation behind the adoption of such a technique lays both on the proper dealing with the curse of dimensionality and the possible improvement that might be produced by the removal of redundant features from the given datasets. Another major reason, that was taken into consideration in our choice to apply further reduction techniques, such as a feature selection algorithm, on our already processed datasets, was that imposed by the required generalization properties that our final classification scheme should possess. In particular, a large number of features is directly translated into a large number of classifier parameters, meaning that for a finite and usually limited number of training patterns, keeping the number of features as small as possible, without a significant information loss, serves the purpose of designing classifiers with good generalization capabilities.

In order to decide the number of features to be selected by the feature selection process we employed again a similar procedure as before by applying the *featureval* function on the datasets generated after the application of the feature extraction technique used on each representation. In more details, we initially iterated over the possible number of features ranging from 10 to 50 features (since that was the dimensionality of the (2 with *pcam* and 1 with *psem* processed datasets) with a step equal to 10 in order to find the appropriate number of features to be used in the feature selection algorithm. After the derivation of that value the *featself* function with the minimum of squared Euclidean distances as the evaluation criterion was utilized to produce the final feature selection mapping to be applied on the given datasets. It should be stressed out that the election procedure of the number of features to be used resulted to the number of 50 features in all representations, which as a matter of fact led to a simple rearrangement of the features in each dataset by the selection algorithm.

3 PROPOSED CLASSIFIERS AND RESULTS

3.1 Evaluated Classifiers

In this assignment we evaluated a variety of classifiers with different characteristics, a short description of which and an explanation of the reason why we chose to examine them being demonstrated below. The performance and the most efficient classifiers of the above list in respect to our task, can be seen at the last subsection, "Test on Benchmark Dataset".

Nearest Mean Classifier

Minimum distance classifiers using Euclidean distance are also called nearest mean classifiers (NMC). In the implementation of this particular classifier we assume one Gaussian distribution for each class and the challenging part is the derivation of the nearest mean between the two distributions, which is the point where the decision boundary of the classifier can be found.

$$g(x) = \begin{bmatrix} w^T & \omega \end{bmatrix} \cdot \begin{bmatrix} x \\ 1 \end{bmatrix} = \hat{w}^T \hat{x} \tag{7}$$

Parzen classifier

The Parzen classifier, as well as the Nearest Neighbor classifiers, is an non-parametric classifier, meaning that the knowledge of the data distribution is not necessary. These type of classifiers aim to process the smoothness of the samples distribution. The main advantages and disadvantages of such a classifier are presented as follows:

Advantages:

- Simple and flexible classifier
- Often a very good classification performance
- It is simple to adapt the complexity of the classifier

Disadvantages:

- Relatively large training sets are needed
- The complete training set has to be stored
- Distances to all training objects have to be computed
- The features have to be scaled sensibly
- The value for h has to be optimized

The parzen classifier estimates the probability distribution by calculating the following equation.

$$\hat{p}(\mathbf{x} \mid \omega_i) = \frac{1}{n_i} \sum_{j=1}^{n_i} N(\mathbf{x} \mid \mathbf{x}_j^{(i)}, hI) \tag{8}$$

The main challenge of the Parzen classifier is the optimization of h.

Linear Bayes Normal Classifier

The Linear Bayes Normal classifier (ldc), like the Parzen and the Nearest Neighbor classifier, calculates a linear discriminant function w for the dataset. LDC is a linear function of features that depends on the weighted sum w of the features and given its value the Linear Discriminant classifier decides on the class of each object.

Quadratic Bayes Normal Classifier

Quadratic discriminant analysis (QDA) is closely related to linear discriminant analysis (LDA), where it is assumed that the measurements from each class are normally distributed. Unlike LDA however, in QDA there is no assumption that the covariance of each of the classes is identical. A quadratic model can be seen as a generalization of the linear model, and its use is justified by the desire to extend the classifier's ability to represent more complex separating surfaces.

In a more practical view, qdc is a Bayes plug-in classifier which makes the assumption of different means and covariance matrices for each of the classes. In order for the unknown parameters to be found, a significant amount of data is needed.

K-Nearest Neighbor Classifier
The Nearest Neighbor classifier, as the Parzen classifier, is a non-parametric classifier. Nearest Neighbor classifier estimates the probability distribution by calculating the following equation.

$$\hat{p}(x \mid \omega_m) = \frac{k_m}{n_m V_k}. \tag{9}$$

k is the number of the nearest neighbours and V_k is the volume of the sphere centered at x with radius r, which is equal to the distance to the k_{th} nearest neighbor. The main advantages and disadvantages of the nearest neighbor classifier are demonstrated below.
Advantages:

- Training is very fast
- Simple and easy to learn
- Robust to noisy training data
- Effective if training data is large

Disadvantages:

- Biased by value of k
- Computation Complexity
- Memory limitation
- Easily fooled by irrelevant attributes

One more characteristic of the Nearest Neighbor classifier that cannot be categorized neither as an advantage nor as a disadvantage is the ability to gain better performance by increasing the number of data samples available.
Logistic Multi-Class Linear Classifier (Multinomial Logit) Logistic regression classifier is a generalized linear model which derives form the fact that the model depends on linear components, despite the fact that the model itself is considered a nonlinear one as the sigmoid function used $\phi(z) = \frac{1}{1+e^{-z}}$ is not a linear one. The decision boundary is estimated through the optimization of the below function.

$$ln(L) = \sum_{i=1}^{N_1} lnp(x_i^{(1)} \mid \omega_1) + \sum_{i=2}^{N_2} lnp(x_i^{(2)} \mid \omega_2) \tag{10}$$

The gradient descent algorithm is used for the optimization of the $ln(L)$ function.
Fisher's Least Square Linear Discriminant
Fisher Linear Discriminant is again a linear classifier which aims to estimate the linear border $g(x) = w^T x + \omega_0 = 0$. For the optimization of that linear border, the weight (w) parameter has to be optimized. For this optimization the Fisher's criterion is used and the below equation is transformed.

$$J_F = \frac{w^T \sum_{between} w}{w^T \sum_{within} w} \tag{11}$$

Neural Network classifier This category mainly contains non-linear classifiers inspired by biological neural networks (human brain). In specific, these networks consist from a set of layers of nodes (neurons), which are categorized in three categories, the input, the hidden and the output neurons. The functionality of each type of neurons consists of feeding the data to the system, optimizing the weights of the model and combining the outputs of the hidden neurons to the template needed for the given classification problem respectively.

The networks examined in the current work are the Random Neural Networks RNN, which briefly constitute recurrent models, i.e. neural networks that are allowed to possess complex feedback loops.
Linear Perceptron classifier
The Percepton algorithm, first proposed by Frank Rosenblatt [2], is a linear classification algorithm which aims to classify different classes by estimation the cost function

$$J(w) = \sum_{missclassified} -y_i w^T x_i \tag{12}$$

The optimization of the parameter w to maximize the J function is performed with the gradient descent algorithm. Besides the linear nature of the algorithm, Perceptron is one of the first classifiers which approximated the nature of neural networks. The main advantages and disadvantages of the algorithm are presented below:
Advantages:

- Trained incrementally, or in batches
- When the data is separable, it will find the solution
- Applicable to very large datasets

Disadvantages:

- When the data is not separable, it will update for ever.

One variant of such classifiers that we also tested in the premises of the current assignment is the Voted Perceptron, which is a more complex version of the Percepton classifiers and it is presented in more details below.
Voted Perceptron classifier
The Voted Perceptron, proposed by Freund and Schapire in 1999 [3], is a variant using multiples weighted perceptrons. The algorithm starts a new perceptron every time an example is wrongly classified, initializing the weights vector with the final weights of the last perceptron. Each perceptron will also be given another weight corresponding to the number of the correctly classified examples reported before the first appearance of the wrongly classified one, and at the end the output will be a weighted vote on all those perceptrons.
Support Vector Machine classifier
The SVM classifier constitutes another linear classifier which in short tries to find a classification boundary which maximizes the margin between the support vectors. Its main advantages and disadvantages are:
Advantages:

- Accuracy
- Works well on smaller cleaner datasets
- It can be more efficient because it uses a subset of training points
- The classifier tends to perform very well in high dimensional feature spaces.

Disadvantages:

- Not suited to larger datasets as the training time with SVMs can be high
- The data should be separable
- The decision boundary is linear

SVM is extensively used for text classification tasks such as category assignment, detecting spam and sentiment analysis. It is

also commonly used for image recognition challenges, performing particularly well in aspect-based recognition and color-based classification. SVM, finally, plays a vital role in many areas of handwritten digit recognition, such as postal automation services.

3.2 Scenario 1

In scenario 1, as it is mentioned in the introduction, we tested the proposed classifiers with a large amount of training data. The number of samples combined with the number of features that we chose, were proved to be sufficient since some particular classifiers, the results of which are demonstrated later in the current report, achieved high accuracy (classification error less than 6%).

3.2.1 *Experimental Setup*.

We used 200 samples per class, resulting to a training set consisting of 2000 samples. The optimal number of features is 50, which it can be seen in the learning curves of each classifier for each different representation in figures 1, 3 and 5. That number is the result of the two feature reduction techniques that have been analysed in previous sections, namely the PCA (or Pseudo-Euclidean Linear Embedding for dissimilarity representation) and the Feature Selection technique.

3.2.2 *Results*.

The results for the classification error for each representation and each classifier that we experimented with, after applying PCA (or Pseudo-Euclidean Linear Embedding for dissimilarity representation) and feature selection techniques as described in the previous section are listed in tables 1, 4 and 7. In specific, both the cross-validation and the hold-out results for each classifier are presented in these tables. The cross-validation procedure was followed since it is alleged to provide us with as less biased results as possible.

It can be easily seen that the best performing classifier in pixel representation (Table 4) is qdc with 5% classification error, while vpc produces an error equal to 7% while Parzen, k-NN and SVM produce 9% each. These results were expected, since these classifiers can achieve good classification performance, in large datasets with high dimensional features (large feature space).

In the feature representation (Table 1) mostly the same pattern for the best performing classifiers is demonstrated. Again qdc, as the best performing classifier, achieved a cross-validation error equal to 11% and a hold-out error of 10%, being followed by vpc, ldc, svm, parzen and knn with cross-validation errors of 11% for the first two, and 12% 14% and 15% for the last three respectively.

Finally, in the dissimilarity representation (Table 7) qdc perfromed as well as in the pixel representation, since it only produced 5% classification error, while svm, k-NN and Parzen had 9%, 10% and 10% classification error respectively. Similarly to the previous representations, the same pattern with some small variations can be seen in the dissimilarity representation too.

Concluding, qdc seems to be the most accurate classifier in the three representations, while most of the best performing classifiers presented a lower classification error in the pixel representation, where the leading classifier qdc performed with an accuracy eqaul to 95% in both pixel and dissimilarity representation. Dissimilarity representation can be theoretically characterized as the most efficient representation in comparison with the two rest.

3.2.3 *Combining classifiers*.

The evaluation of the best classifiers for each representation led us to try a combination of them. As a result, we picked the best three classifiers for each representation with the produced results being characterized as sufficient for a good classification but not good enough to choose a combining scheme instead an individual classifier, since that would entail slightly more classification error and in parallel greater computational complexity.

In particular, the outcome of the classification results from the combination strategies for the feature representation are illustrated in the tables 2 and 3 for the parallel and the sequential combination strategy respectively. The combined classifiers were Parzen, Voted Perceptron and QDC. As it can be observed from Table 2, the best classification result for parallel combination was 10% and it was obtained by the maximum, mean and product combiners. Also from table 3, we can see that the classification result for sequential combination was the same for all the combining rules that we tested (13%). Lastly, it can be easily seen that in the feature representation the performance of the best individual classifiers was better than the one achieved by the combination of the best of them.

For the pixel representation, the classification results from the combination strategies are shown in Tables 5 and 6. The classifiers that we combined in this case were the same as in the feature representation. In Table 5, it can be observed that the best performance for the parallel combination is produced by the minimum combiner with 6%, while for the sequential combination, Table 6 demonstrates that all the combiners achieve 8% classification error. Once more, the best individual classifiers produce lower error than the combined ones.

Regarding the dissimilarity representation, the classification results from the combination strategies are demonstrated in Tables 8 and 9. The classifiers that we combined in this case were QDC, Parzen and Nearest Neighbor classifier. As it can be observed from 8, the lower classification error from their parallel combination derives from the product combiner with 7%. As far as their sequential combination is concerned, Table 9 shows that all the combiners achieve 38% classification error. As in the other representations, the best individual classifiers again perform better than the combined ones.

Overall, the combination of the best classifiers for each of the representations did not provide any significant help in building a better performing classifier compared to the individual ones.

3.3 Scenario 2

3.3.1 *Experimental Setup*.

In scenario 2, contrary to scenario 1, the amount of available training data is much smaller, namely 10 objects per class. As a result, we expected that because of the small number of samples (100 in total), the optimal number of features that is produced from PCA (or Pseudo-Euclidean Linear Embedding for dissimilarity representation) and feature selection will be lower than the one for the first scenario. However, by the process which was described in details in previous sections, we reached to the conclusion that the appropriate number of features for all the representations is the same as the one in scenario 1, namely 50. This outcome can be confirmed

Classifier	Hold-out Error	10-fold CV Error
SVM	12%	12%
Nearest Mean	25%	25%
QDC	10%	11%
LDC	12%	11%
Parzen	13%	14%
k-NN	14%	15%
Logistic	16%	14%
Fisher	15%	15%
RNN	16%	16%
Linear Perceptron	19%	16%
Voted Perceptron	12%	11%

Table 1: Scenario 1 - Errors for feature representation

Combining Rule	Hold-out Error	10-fold CV Error
Max	10%	10%
Min	13%	12%
Mean	10%	10%
Prod	11%	10%

Table 2: Scenario 1 - Parallel combiner (fisher, svm, vpc) errors for feature representation

Combining Rule	Hold-out Error	10-fold CV Error
Max	15%	13%
Min	14%	13%
Mean	13%	13%
Prod	14%	13%

Table 3: Scenario 1 - Sequential combiner (fisher, svm, vpc) errors for feature representation

Classifier	Hold-out Error	10-fold CV Error
SVM	9%	9%
Nearest Mean	18%	18%
QDC	5%	5%
LDC	16%	22%
Parzen	9%	9%
k-NN	9%	9%
Logistic	21%	33%
Fisher	19%	25%
RNN	63%	55%
Linear Perceptron	38%	30%
Voted Perceptron	8%	7%

Table 4: Scenario 1 - Errors for pixel representation

by the the feature curves of each classifier for all the 3 representations in figures 7, 9 and 11. In these figures we can observe that in the majority of the cases the classification error stabilizes on 50 features.

Combining Rule	Hold-out Error	10-fold CV Error
Max	9%	9%
Min	4%	6%
Mean	8%	9%
Prod	4%	4%

Table 5: Scenario 1 - Parallel combiner (svc, parzenc, knnc) errors for pixel representation

Combining Rule	Hold-out Error	10-fold CV Error
Max	5%	8%
Min	5%	8%
Mean	5%	8%
Prod	6%	8%

Table 6: Scenario 1 - Sequential combiner (svc, parzenc, knnc) errors for pixel representation

Classifier	Hold-out Error	10-fold CV Error
SVM	9%	9%
Nearest Mean	19%	20%
QDC	5%	5%
LDC	12%	12%
Parzen	11%	10%
k-NN	10%	10%
Logistic	16%	15%
Fisher	13%	14%
RNN	15%	13%
Linear Perceptron	18%	16%
Voted Perceptron	16%	15%

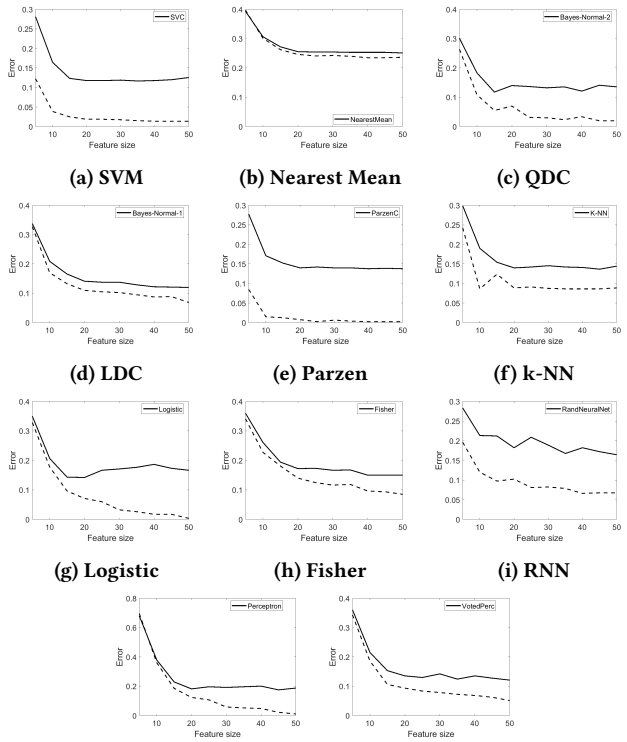
Table 7: Scenario 1 - Errors for dissimilarity representation

Combining Rule	Hold-out Error	10-fold CV Error
Max	13%	12%
Min	12%	8%
Mean	13%	12%
Prod	10%	7%

Table 8: Scenario 1 - Parallel combiner (nmc, parzen, qdc) errors for dissimilarity representation

Combining Rule	Hold-out Error	10-fold CV Error
Max	42%	38%
Min	42%	38%
Mean	42%	38%
Prod	42%	38%

Table 9: Scenario 1 - Sequential combiner (nmc, parzen, qdc) errors for dissimilarity representation



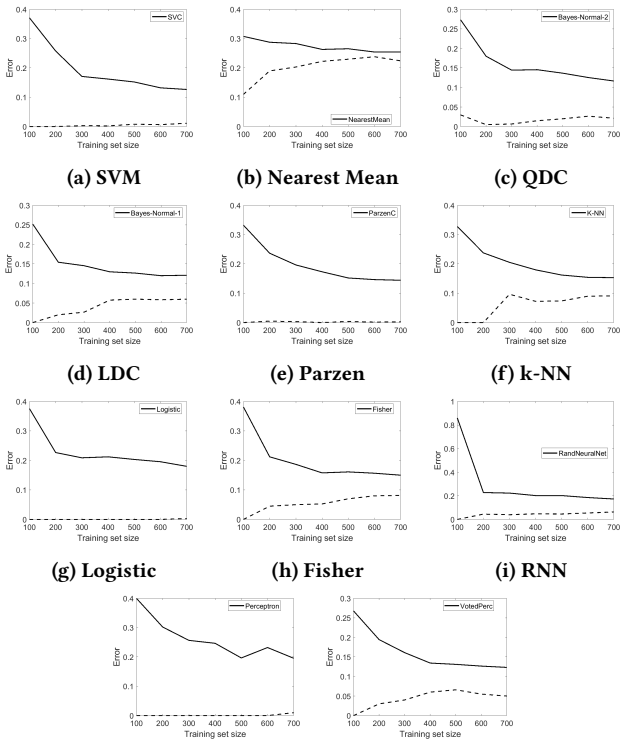
(a) SVM (b) Nearest Mean (c) QDC

(d) LDC (e) Parzen (f) k-NN

(g) Logistic (h) Fisher (i) RNN

(j) Linear Perceptron (k) Voted Perceptron

Figure 1: Scenario 1 - Feature curves for feature representation



(a) SVM (b) Nearest Mean (c) QDC

(d) LDC (e) Parzen (f) k-NN

(g) Logistic (h) Fisher (i) RNN

(j) Linear Perceptron (k) Voted Perceptron

Figure 2: Scenario 1 - Learning curves for feature representation

Furthermore, we plotted the learning curve for each classifier examined in our experiments. From these plots in figures 8, 10 and 12, we can see that the classification error stabilizes on 100 samples. Thus, we decided to use 100 samples in the experiments we conducted for scenario 2, since this choice is in accordance with the description of that scenario.

3.3.2 *Results*.

The results for the classification error for each representation and each classifier we experimented with, after applying PCA (or Pseudo-Euclidean Linear Embedding for dissimilarity representation) and feature selection techniques as described in the previous section are listed in tables 10, 13 and 16.

The best classifier in pixel representation is k-NN with 13% classification error, while Parzen and SVM also achieved relatively low error, namely 17% and 18% correspondingly. The results for SVM and k-NN was quite expected as k-NN is known that it is considered a suitable choice for small datasets and data with low dimensionality, and SVM is good at dealing with small data samples, since only support vectors and not all the samples, are used to construct the separating hyperplane. However, the relatively good performance of the Parzen classifier was not one of our expectations in that particular scenario.

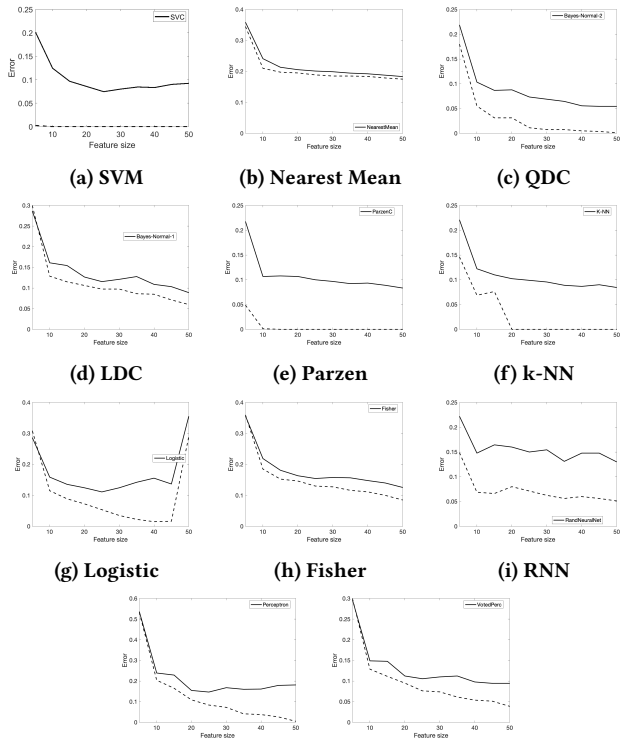
As far as the feature representation is concerned, Voted Perceptron is the optimal classifier with 25% classification error, while SVM and fisher achieved good results too with 27% and 28% accordingly. The different classifier in comparison with the pixel representation is the Voted Perceptron, whose result was expected since the number of parameters need to be estimated is not so big.

In the case of the dissimilarity representation, the best classifiers reported consist of the Quadratic Discriminant classifier, the Nearest Mean classifier and the Parzen classifier with classification errors 24%, 26% and 29% respectively. The result for Nearest Mean classifier was expected since it is a simple classifier (does not require the estimation of many parameters), while the one for Quadratic Discriminant classifier was unexpected because it usually requires a large amount of samples to provide accurate classification results.

Overall, we can conclude that the best results were achieved by using the pixel representation and the k-Nearest Neighbors classifier with 13% classification. In addition, SVM (with exponential kernel) and Parzen classifiers also produced good classification results for pixel representation, with 18% and 17% classification error correspondingly. However, the conclusion that the best representation is the one with pixels is opposed to the theory, since it is known that dissimilarity representation is the most efficient one. A possible explanation to this result is ...

3.3.3 *Combining classifiers*.

After the evaluation of the classifiers for each representation as described above, we combined the three best performing classifiers



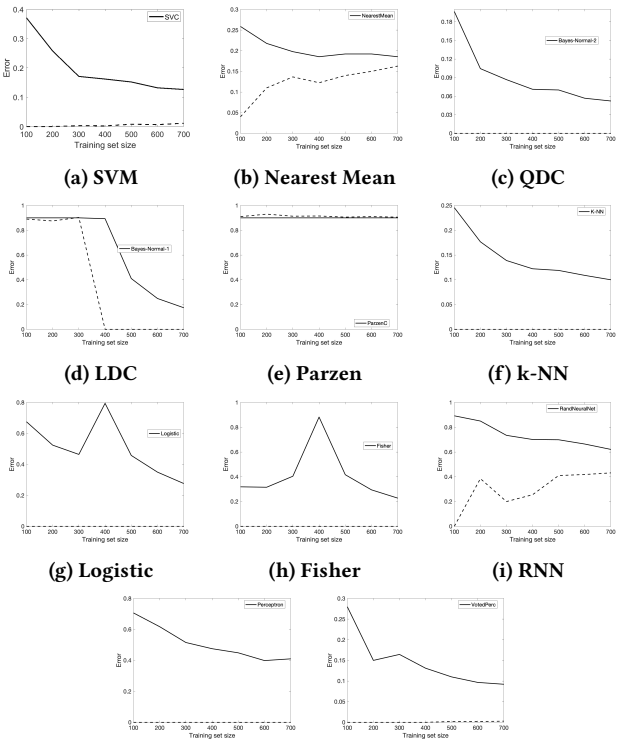
(a) SVM (b) Nearest Mean (c) QDC

(d) LDC (e) Parzen (f) k-NN

(g) Logistic (h) Fisher (i) RNN

(j) Linear Perceptron (k) Voted Perceptron

Figure 3: Scenario 1 - Feature curves for pixel representation



(a) SVM (b) Nearest Mean (c) QDC

(d) LDC (e) Parzen (f) k-NN

(g) Logistic (h) Fisher (i) RNN

(j) Linear Perceptron (k) Voted Perceptron

Figure 4: Scenario 1 - Learning curves for pixel representation

for each representation based on these results. We tried both the sequential and parallel combination of them.

The outcome of the classification results from the combination strategies for feature representation is illustrated in tables 11 and 12. The combined classifiers were Fisher, Voted Perceptron and SVM. As we can observe from table 11, the best classification result for parallel combination was 28% and it was obtained by the maximum and mean combiners. Also from table 12, we can see that the classification result for sequential combination was 35% and it was the same for all type of combiners. We can conclude that for the feature representation the performance of individual classifiers was better than the one achieved by the combination of the best of them.

For the pixel representation, the classification results from the combination strategies are shown in tables 14 and 15. The classifiers, that we combined in this case, were SVM, Parzen and k-NN. From table 14 it can be observed that the best performance from their parallel combination derives from the product combiner with 29%. As far as their sequential combination is concerned, table 15 demonstrates that all the combiners achieve 25% classification error. Once more, the individual classifiers have lower error than the combined ones.

Finally, regarding the dissimilarity representation, the classification results from the combination strategies are demonstrated in tables 17 and 18. The classifiers, that we combined in this case, were Quadratic Discriminant classifier, Parzen and Nearest Mean. As it can be observed from Table 17, the lowest classification error from

Classifier	Hold-out Error	10-fold CV Error
SVM	32%	28%
Nearest Mean	27%	33%
QDC	19%	32%
LDC	19%	29%
Parzen	28%	35%
k-NN	31%	36%
Logistic	28%	34%
Fisher	22%	27%
RNN	95%	83%
Linear Perceptron	46%	36%
Voted Perceptron	18%	25%

Table 10: Scenario 2 - Errors for feature representation

their parallel combination is produced by the minimum combiner with 33%. As far as their sequential combination is concerned, table 18 shows that all the combiners achieve 38% classification error. As in the other representations, the individual classifiers performs better than the combined ones.

The overall conclusion for the combined classifiers is that the combination of the best classifiers for each of the representations did not help us build a better classifier than the individual ones.

Combining Rule	Hold-out Error	10-fold CV Error
Max	22%	28%
Min	25%	35%
Mean	21%	28%
Prod	24%	31%

Table 11: Scenario 2 - Parallel combiner (fisher, svm, vpc) errors for feature representation

Combining Rule	Hold-out Error	10-fold CV Error
Max	23%	35%
Min	26%	35%
Mean	27%	35%
Prod	25%	35%

Table 12: Scenario 2 - Sequential combiner (fisher, svm, vpc) errors for feature representation

Classifier	Hold-out Error	10-fold CV Error
SVM	25%	18%
Nearest Mean	17%	25%
QDC	17%	21%
LDC	81%	52%
Parzen	26%	17%
k-NN	31%	13%
Logistic	27%	87%
Fisher	62%	81%
RNN	58%	79%
Linear Perceptron	44%	30%
Voted Perceptron	26%	22%

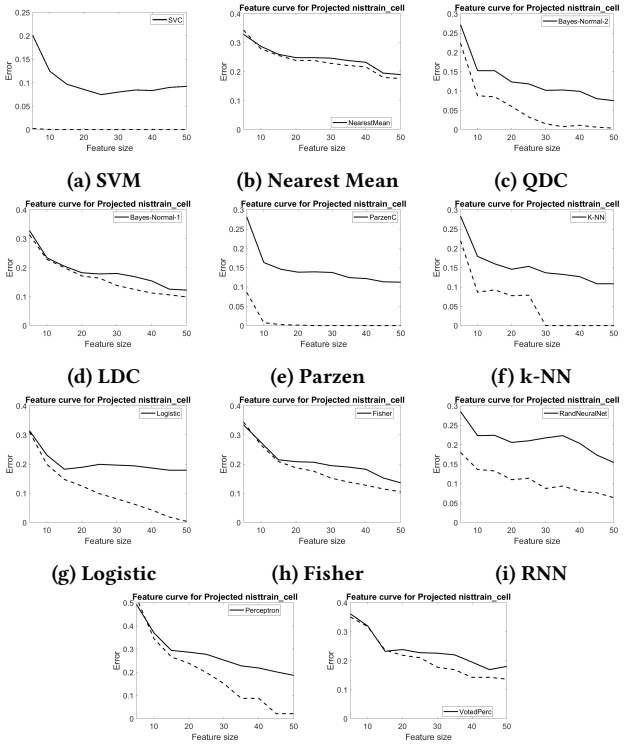
Table 13: Scenario 2 - Errors for pixel representation

Combining Rule	Hold-out Error	10-fold CV Error
Max	24%	35%
Min	22%	30%
Mean	24%	33%
Prod	19%	29%

Table 14: Scenario 2 - Parallel combiner (svc, parzenc, knnc) errors for pixel representation

Combining Rule	Hold-out Error	10-fold CV Error
Max	18%	25%
Min	18%	25%
Mean	18%	25%
Prod	18%	25%

Table 15: Scenario 2 - Sequential combiner (svc, parzenc, knnc) errors for pixel representation



(a) SVM **(b) Nearest Mean** **(c) QDC**

(d) LDC **(e) Parzen** **(f) k-NN**

(g) Logistic **(h) Fisher** **(i) RNN**

(j) Linear Perceptron **(k) Voted Perceptron**

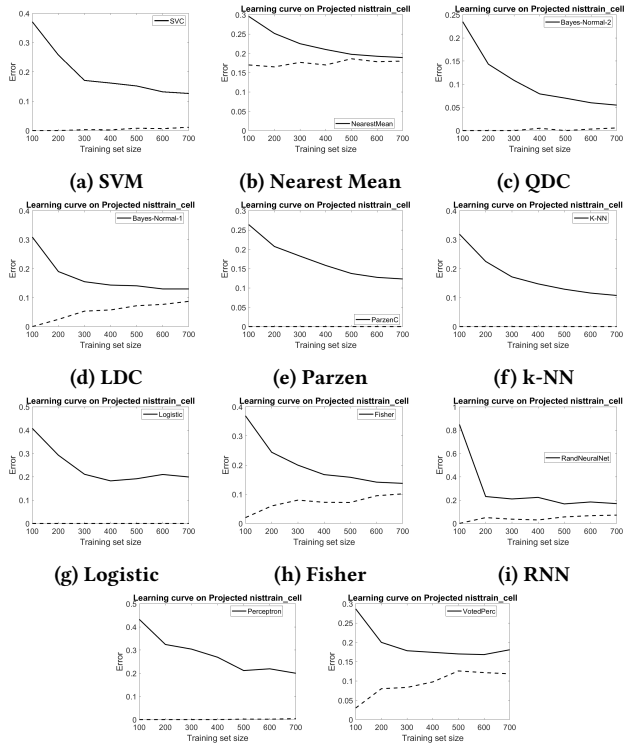
Figure 5: Scenario 1 - Feature curves for dissimilarity representation

Classifier	Hold-out Error	10-fold CV Error
SVM	90%	90%
Nearest Mean	26%	26%
QDC	21%	24%
LDC	24%	42%
Parzen	20%	29%
k-NN	18%	30%
Logistic	34%	43%
Fisher	27%	40%
RNN	86%	77%
Linear Perceptron	42%	42%
Voted Perceptron	37%	33%

Table 16: Scenario 2 - Errors for Dissimilarity representation

Combining Rule	Hold-out Error	10-fold CV Error
Max	88%	82%
Min	28%	33%
Mean	88%	82%
Prod	50%	48%

Table 17: Scenario 2 - Parallel combiner (nmc, parzen, qdc) errors for dissimilarity representation



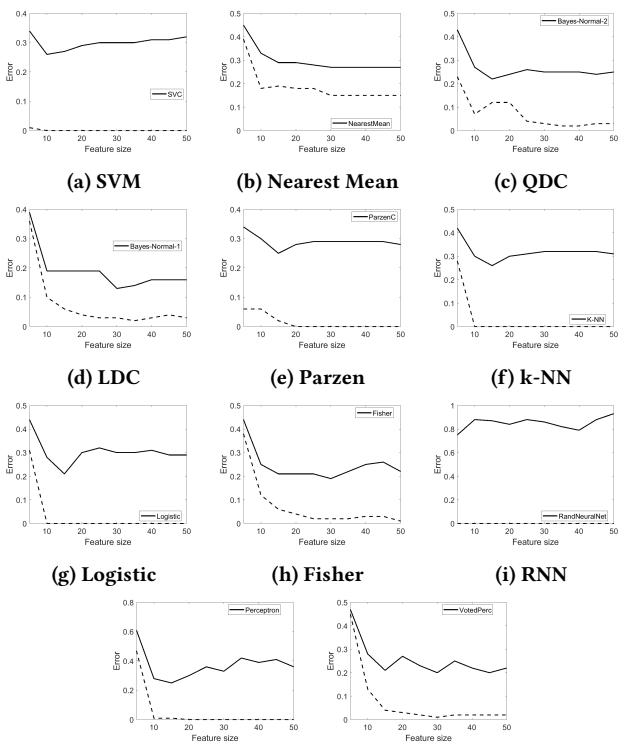
(a) SVM (b) Nearest Mean (c) QDC

(d) LDC (e) Parzen (f) k-NN

(g) Logistic (h) Fisher (i) RNN

(j) Linear Perceptron (k) Voted Perceptron

Figure 6: Scenario 1 - Learning curves for dissimilarity representation



(a) SVM (b) Nearest Mean (c) QDC

(d) LDC (e) Parzen (f) k-NN

(g) Logistic (h) Fisher (i) RNN

(j) Linear Perceptron (k) Voted Perceptron

Figure 7: Scenario 2 - Feature curves for feature representation

Combining Rule	Hold-out Error	10-fold CV Error
Max	45%	38%
Min	45%	38%
Mean	45%	38%
Prod	45%	38%

Table 18: Scenario 2 - Sequential combiner (nmc, parzen, qdc) errors for dissimilarity representation

Representation	#Features	Classifier	Error
Pixel	50	QDC	3.5%

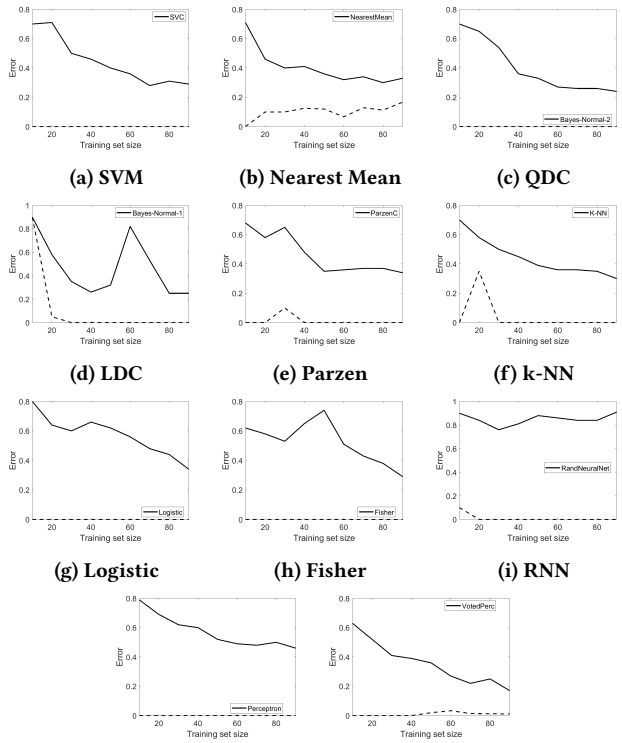
Table 19: Scenario 1 - Benchmark results

3.4 Test on Benchmark Dataset

In this section, the classifier that achieved the lowest classification error with the most efficient representation in each scenario is used to benchmark our system with the *nist_eval* function. The results for the two scenarios are illustrated in tables 19 and 20. In both of them we used the pixel representation and 50 features, as that configuration gave us the best results in our experiments. The number of objects per class that we used in the *nist_eval* function was 200 and 10 for scenario 1 and 2 accordingly. For the first scenario we used the Quadratic Discriminant Classifier and we achieved 3.5% classification error, while for the second scenario we used the k-NN classifier and we got 22.1% error. Both the results meet the requirements of 5% and 25% respectively concerning the classification error.

In scenario 1, the classification error in training set was 5% while the error in the benchmark dataset was 3.5%. Their difference is rather small (1.5%). A possible reason for this difference could be the fact that in our experiments we used 100 objects per class instead of 200. As a result, the bigger training size used in the benchmark was expected to lead to better classification result, given that in both cases the size of the test set is the same.

Regarding the second scenario, the classification error in training set was 13% while the error in the benchmark dataset was 22.1%. Their difference is higher than the one in first scenario (almost 10%). This difference can be explained by the fact that the training set used is very small (10 objects per class, 100 in total) and when a classifier is trained on a small dataset, it leads in a bias towards this particular dataset. As a result, when *nist_eval* function tests the classifier with a completely different dataset this bias leads to a high classification error, as it is very possible that the digits used for testing are different from the ones the classifier was trained on.



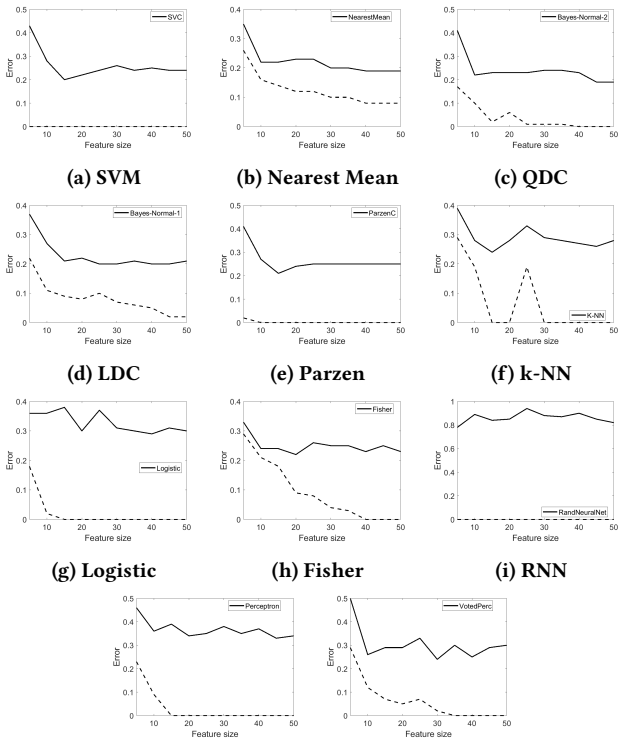
(a) SVM (b) Nearest Mean (c) QDC

(d) LDC (e) Parzen (f) k-NN

(g) Logistic (h) Fisher (i) RNN

(j) Linear Perceptron (k) Voted Perceptron

Figure 8: Scenario 2 - Learning curves for feature representation



(a) SVM (b) Nearest Mean (c) QDC

(d) LDC (e) Parzen (f) k-NN

(g) Logistic (h) Fisher (i) RNN

(j) Linear Perceptron (k) Voted Perceptron

Figure 9: Scenario 2 - Feature curves for pixel representation

Representation	#Features	Classifier	Error
Pixel	50	k-NN	22.1%

Table 20: Scenario 2 - Benchmark results

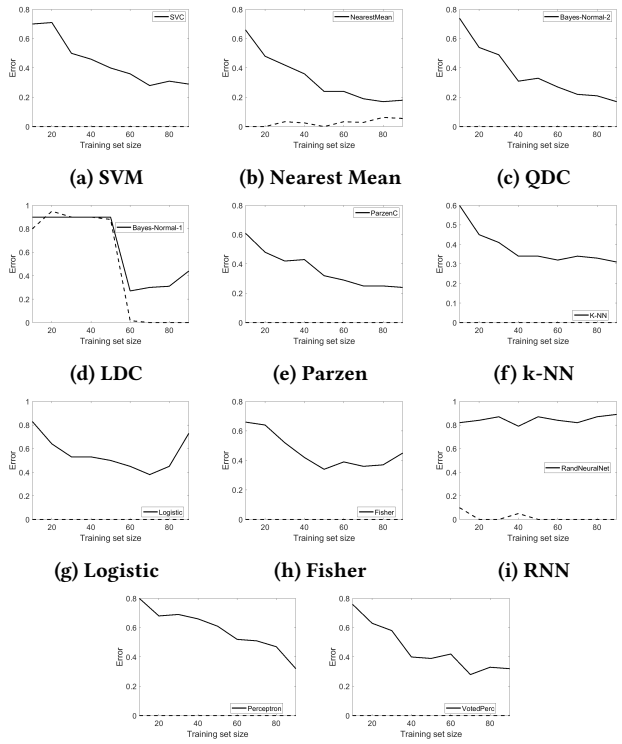
4 CONCLUSION

In this report the whole procedure followed for the implementation of a system able to automatically recognize handwritten digits is documented. The system is tested and evaluated in two different benchmarks with different size of training sets, one with 10 objects per class and one with 200 objects per class. The classification error obtained by 10-fold cross validation was 3.5% and 22.1% for the first and second scenario respectively. These results meet the requirements of the assignment which were 5% and 25% correspondingly.

The meaningful preprocessing of the handwritten digit images and the accurate dimensionality reduction, with the techniques described in the 'Preprocessing' section, constituted the determining factors for the success of our system. Therefore, a significant part of our work was devoted to the experimentation and analysis of the effect that different components of the system, like the number of features, the number of samples and the different classifiers, could have on its accuracy.

Each of the three representations of the data used (features, pixels and dissimilarities) embodies its own advantages and disadvantages, which were analyzed in the 'Preprocessing' section. The same is applicable also for the classifiers used in the system, as described in the 'Proposed Classifiers and Results' section. Therefore, the right choice of both of them is critical for the performance of the system. Still, there is also another issue which someone need to think about. Despite the fact that low classification error is definitely the main goal that someone has to consider when choosing the appropriate classifier and features for their system, the computational complexity and memory limitations are also important factors. Hence, the target should be the design of a system that is able to achieve good accuracy at low complexity, by extracting more discriminatory features and using the appropriate classifiers. In order to do so, the experimentation with different classifiers and several feature sizes, and the evaluation of their results is necessary. This was the motivation of our work and it has played a crucial role in our design choices for the system.

As far as the results achieved by our system are concerned, we can conclude that in respect to the scenario, the optimal classifier is different. In the first scenario where the training set was large, the best classifier was the Quadratic Discriminant Classifier which is a non-linear classifier that requires the estimation of a plethora of parameters. That means that large datasets are necessary in order for that exact classifier to achieve good performance. However, in the second scenario where the training set was much smaller, the best classifiers were proved to be the k-Nearest Neighbors, which indeed performs well for small training sets due to its small number



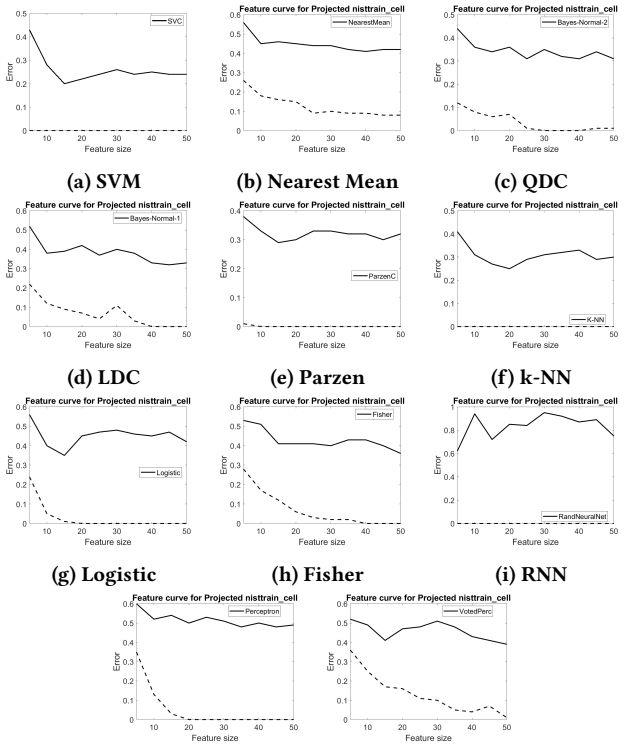
(a) SVM (b) Nearest Mean (c) QDC

(d) LDC (e) Parzen (f) k-NN

(g) Logistic (h) Fisher (i) RNN

(j) Linear Perceptron (k) Voted Perceptron

Figure 10: Scenario 2 - Learning curves for pixel representation



(a) SVM (b) Nearest Mean (c) QDC

(d) LDC (e) Parzen (f) k-NN

(g) Logistic (h) Fisher (i) RNN

(j) Linear Perceptron (k) Voted Perceptron

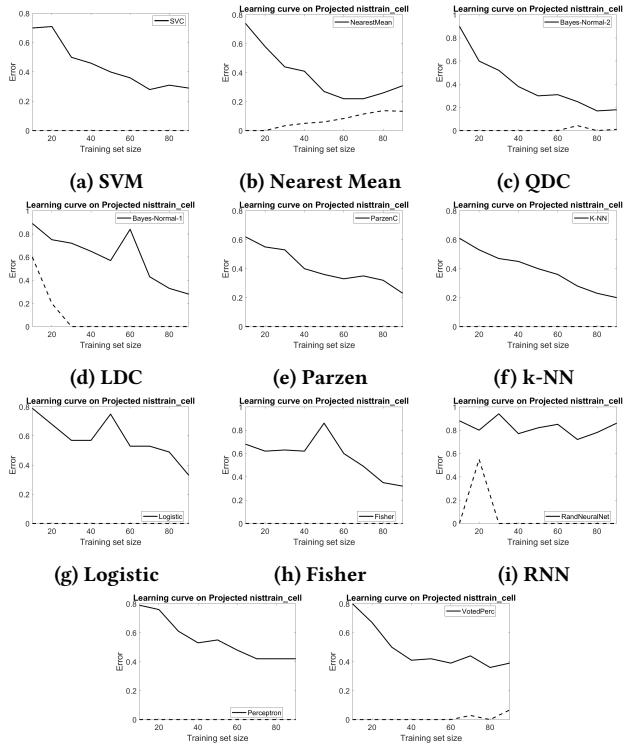
Figure 11: Scenario 2 - Feature curves for dissimilarity representation

of estimated parameters. Also, something that it was expected to happen but we were proved wrong by the evaluation of our experiments, was the discrimination of number of features used in each scenario. Especially, because of the fact that a big number of features usually leads on bad results in small datasets, we expected that the optimal number of features in the second scenario would be lower than the one for the first. Nevertheless, through our tests we found out that the optimal number was the same in both scenarios.

5 LIVE TEST

For further testing of the performance of the proposed classification scheme, a live test on actual handwritten digits was conducted. For the needs of that particular test, the scanned sheet of paper with ten handwritten digits for each class (0-9) presented in Figure 13 was used. The procedure through which the scanned image was converted into a dataset capable of being provided as input to our classification scheme is described as follows.

Firstly, the scanned image was imported into the environment of *Matlab* through the usage of the *imread* function, which reads grayscale or color images from graphics files, and was converted into a grayscale one through the usage of the *rgb2gray* function. After that, the grayscale image was transformed into a binary one through the application of the *imbinarize* function and was complemented with the *imcomplement* function so that it fits the image template that the images of the NIST dataset possess (black background and white digits). After the end of the processing phase of the whole image, each handwritten digit was segmented in order each one of them to be processed as a distinct image. In specific, the 2-D coordinates that describe the area of the image, in which a single digit is present, were calculated through the identification of pairs of rows and columns in the provided image for which the first one in the pair contains some part of a digit while the second one contains only the background. By identifying the indices of the rows and columns with such a property and combining them sequentially in a nested loop over all the values of these row and column indices the separate image of each handwritten digit could be produced through the application of the *im_resize* function on the part of the image described by the indices examined in each loop. After that, each derived image (handwritten digit) was preprocessed with the following procedure. Initially, all the empty rows and columns were removed , while some additional rows and columns were added so that these images are shaped into a square. Sequentially, the images were resized to a $30x30$ data representation, while some rows and columns were again added to the outer part of them so that no digit would touch the border of the image. Finally, the images were again resized into a $30x30$ data representation, so that all the images in the dataset would have the same size, and then the needed prdataset containing 10 images in each of the 10 classes was created. This dataset was used for testing the performance of the



(a) SVM (b) Nearest Mean (c) QDC

(d) LDC (e) Parzen (f) k-NN

(g) Logistic (h) Fisher (i) RNN

(j) Linear Perceptron (k) Voted Perceptron

Figure 12: Scenario 2 - Learning curves for dissimilarity representation

Figure 13: The handwritten digits image used for live test

best classifiers trained on the NIST dataset in both scenarios (200 objects per class and 10 object per class respectively). The results obtained are presented in the following sections.

Representation	Classifier	Error
Pixel	QDC	22%
Pixel	k-NN	22%
Pixel	Parzen	**20%**

Table 21: Scenario 1 - Live Test results

Representation	Classifier	Error
Pixel	k-NN	**22%**
Pixel	Parzen	27%
Pixel	Nearest Mean	38%

Table 22: Scenario 2 - Live Test results

5.1 Results based on scenario 1

For scenario 1, the pixel representation was chosen since that was the one that produced the best benchmark results. Apart from the best performing classifier on the benchmark dataset for that representation, the next two best performing ones according to the cross validated tests conducted on that scenario were also considered. The results provided from the live test on those three classifiers are shown in Table 21.

It is obvious that the live test results indicated lower performance comparing to the reported one on both the experimental and the benchmark results using the NIST dataset. Nevertheless, such a difference in the error rate can be easily explained by the fact that there might be great difference between the images obtained by the scanned sheet with our handwritten digits and those images originating from the NIST dataset and on which our classifiers were trained. For instance, the material used to write down the digits and the quality of the scanned picture could play an important role on the final results. Furthermore, since in this particular scenario we trained our classifiers in a considerable number of images originating from the NIST dataset (200 images per class), the possibility of over-training the classifiers on that problem is for sure existent. Thus, it would be advisable for some different approaches to be followed in order for the whole system to acquire a better generalization capability.

5.2 Results based on scenario 2

The same procedure as in scenario 1 was also followed in scenario 2. Again the pixel representation was chosen and the three best performing classifiers according to the cross validated tests conducted on that scenario were considered. The results provided from the live test on those classifiers are presented in Table 22.

It can be seen that the results produced from the classifiers that were trained in scenario 2 (10 objects per class) lay much closer to both the experimental and the benchmark results obtained before. This phenomenon can be explained by the fact that since our classifiers have been trained on such small training sets, they are considerably biased resulting to a similar error rate in the small test set used for the evaluation of the live test performance as that reported mainly on the benchmark datasets. Nevertheless, the fact, that there is such a small difference between the experimental and the live test errors, suggests that the image-related preprocessing

procedure applied on the scanned picture and the methods used to produce the pixel representation of the available data generate "informationally rich" datasets with considerable discriminating power.

6 RECOMMENDATIONS

In this assignment we achieved through numerous preprocessing techniques and parameter estimations on various classifiers to reduce the classification error to a reliable percentage for digit recognition and to achieve robust classification of the digits. The complexity of the preprocessing techniques, namely feature extraction (image processing techniques, dissimilarities), feature reduction (PCA, Pseudo-Euclidean Linear Embedding) and feature selection, help us to understand that the estimation of the parameters of a classifier and the quantity and quality of the features to be utilized can be characterized as the most important procedure in a classification problem. Through the tests conducted on the three different representations (feature, pixel, dissimilarities) and the results obtained from them it can be easily seen that all the representations have some advantages and disadvantages and that different classifiers for each representation produce different results. There is a great need for a more pellucid unified and robust classification system so that these differences demonstrated in the previous sections could be minimized. For example, it is undeniable that more samples (more than 200) could be used in the first scenario, since the training of a system with more samples could lead to better results, but again that will be a trivial solution to this problem.

In addition, we chose to combine classifiers that achieved high accuracy when we test them separately, with the motivation that the combination of two classifiers which achieved individually high accuracy, as M. Breukelen et al. [4] proposed, could lead to even better results. Apart from that, the combination of classifiers does not only attempts to answer the question "When does combining classifiers result in a reduction of classification errors and why" but also to investigate how a classifier can overcome the problems of other classifiers through the combination of their outputs. Potential improvement could also be produced through a more sophisticated selection procedure of the dimentionality of PCA or the estimation of the k parameter in the Nearest Neighbour classifier and the tuning of the SVM hyperparameters. Nevertheless, this tuning procedure can be accused as biased by the human intervention. To avoid these classification inconveniences, Dan Ciresan et al. [5] proposed the utilization of a biologically plausible deep artificial neural network. The test error rate of the proposed method of Dan Ciresan et al.[5] is only 0.23% and can be seen in the table [11] with the test error rate results.

Deep Neural Networks and more specifically Convolutional Neural Networks (CNN) are most commonly applied to analyzing visual imagery. Convolutional networks were inspired by biological processes in a way that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap covering in that way the entire visual field. In specific, CNNs constitute an end to end classification tool, which manages to embody the feature extraction, selection and reduction procedures, given that it is trained with sufficient data. Thus, it can be easily assumed that it is also more unbiased in respect to the human factor. Solely by understanding the way CNNs were inspired, it can be assumed that CNNs are one of the best possible techniques for handwritten digit recognition unlike the RNNs that we chose which can produce better results to sequential data like video or sound and not to spatial data as images.

[11] http://yann.lecun.com/exdb/mnist/?fbclid=IwAR2v_1U_ZeHlcgqREmAc0Btj7UVr4CpDTWsOje0jpwWKkpRb2SBM2u2Mj0Y

REFERENCES

[1] RPW Duin, P Juszczak, P Paclik, E Pekalska, D De Ridder, DMJ Tax, and S Verzakov. A matlab toolbox for pattern recognition. *PRTools version*, 3:109–111, 2000.

[2] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[3] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, Dec 1999.

[4] Martijn van Breukelen, Robert PW Duin, David MJ Tax, and JE Den Hartog. Handwritten digit recognition by combined classifiers. *Kybernetika*, 34(4):381–386, 1998.

[5] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745, 2012.