March 2, 2018
Marco

Pattern Recognition methods applied to the MNIST database

Abstract

In this manuscript we study identification of handwritten digits by use of four different methods. A first attempt will be made by use of multinomial logistic regression. Then, two machine learning methods will be considered, support vector machines and neural networks, which typically provide the best results published in the literature, see (LeCun *et al.* 1998) and here for a review of the techniques, their performance and an introduction to the MNIST database. The main reference for support vector machines and neural network used is (Bishop 2006). Finally, a warping based method similar to the one proposed in (Glasbey & Mardia 2001) will be considered. Even though a comparison will be made between these methods, we would like to stress that not all parameters have been optimized and so no conclusion about which classifier performs best should be drawn.

MNIST

In this report we will analyse the MNIST database, which is a large database of handwritten digits, containing 60000 training images and 10000 test images. The digits have been preprocessed, in particular they have been normalized in size and centered in 28×28 pixel images. The original digits were black and white, but during the preprocessing step they have been converted to gray images. In Figure 1, the first 25 digits of the database are shown together with their correct label.

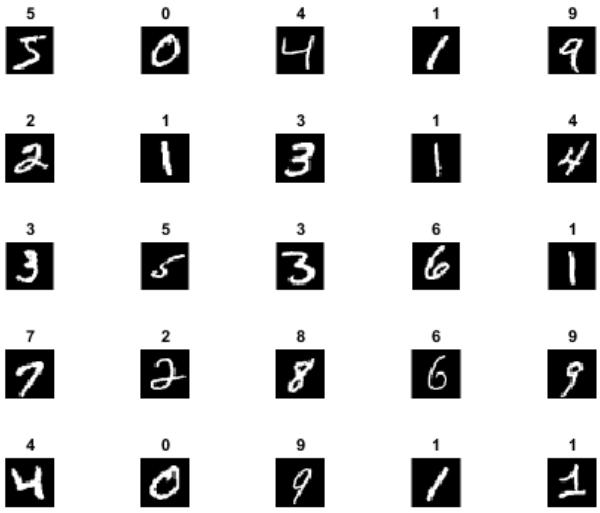

Figure 1: Example of digits included in the MNIST database with the corresponding label on top of them.

1 t-SNE

We describe t-distributed Stochastic Neighbor Embedding as a method to visualize high dimensional data. The method was introduced in (van der Maaten & Hinton 2008) and tested on different datasets including the MNIST one. In t-SNE we suppose to have data $x_1, \dots, x_n \in \mathbb{R}^{d_1}$ and we want to map them to $y_1, \dots, y_n \in \mathbb{R}^{d_2}$, where usually $d_1 >> 3$ and $d_2 = 2, 3$. On the space where the data live, we define a similarity map between points as

$$p_{ji} = \frac{p_{j|i} + p_{i|j}}{2n}, \tag{1}$$

where

$$p_{j|i} = \frac{\exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}\right)}. \tag{2}$$

and σ_i is chosen by fixing a parameter ρ_i called perplexity and imposing that

$$2^{-\sum_j p_{j|i} \log_2 p_{j|i}} = \rho_i. \tag{3}$$

Typical values of the perplexity are between 5 and 50 and are provided by the user as one of the input of t-SNE, see (van der Maaten & Hinton 2008). Similarly, we define the

similarity for pairs of mapped points y_i, y_j as

$$q_{ji} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq i}(1 + \|y_k - y_i\|^2)^{-1}}, \tag{4}$$

where the similarity of points is obtained as a function of their distance by using a t-distribution with one degree of freedom. The points y's are the found by minimizing the Kullback-Leibler divergence

$$\sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{5}$$

T-SNE will be used to visualize the results of the classification of the digits in the later sections, see for example Figure 4. As the digits live in a 28×28 dimensional space, it would be impossible to visualize them to see if the same digits form clusters, thus we need to map them to $\mathbb{R}^2$ in a reasonable way.

2 Multinomial logistic regression

We start by considering recognition of the digits through multinomial logistic regression. Here, as predictors we use all the pixel values in an image and as response the value of the digit represented in the image. Consider x to be a digit image, $l \in \{0, ..., 9\}$ its label, and define $\pi_i = \mathbb{P}(l = i),\ i = 0, ..., 9$. In a multinomial logistic regression the relationship between the response variables and the predictor is defined by

$$\text{score}(i, x) = ln\left(\frac{\pi_i}{\pi_k}\right) = \beta_i \cdot x, \tag{6}$$

where k is a reference class and usually is the last one. The scores can be converted into probabilities using the following

$$\pi_i = \frac{e^{\text{score}(i,x)}}{1 + \sum_{j=0}^{8} e^{\text{score}(j,x)}} = \frac{e^{\beta_i \cdot x}}{1 + \sum_{j=0}^{8} e^{\beta_j \cdot x}} \text{ for } i = 0, ..., 8 \tag{7}$$

and

$$\pi_9 = \frac{1}{1 + \sum_{j=0}^{8} e^{\beta_j \cdot x}}. \tag{8}$$

The estimated coefficients β based on 1000 images are plotted in Figure 2. In some of them, we can actually see an "underlying" number while in other, for example the 6, is less clear. The classification then goes as follow: we take the inner product of the image to be classified and each of the coefficients' matrices shown in Figure 2. This gives us the vector of scores in (6) and we assign the image to the class with the highest score. We know that to maximize an inner product the two elements (vectors,matrix) should be aligned. Intuitively, the coefficients of the logistic regression act as templates for each digit and we classify an image based on which of these templates is most "aligned" to it.

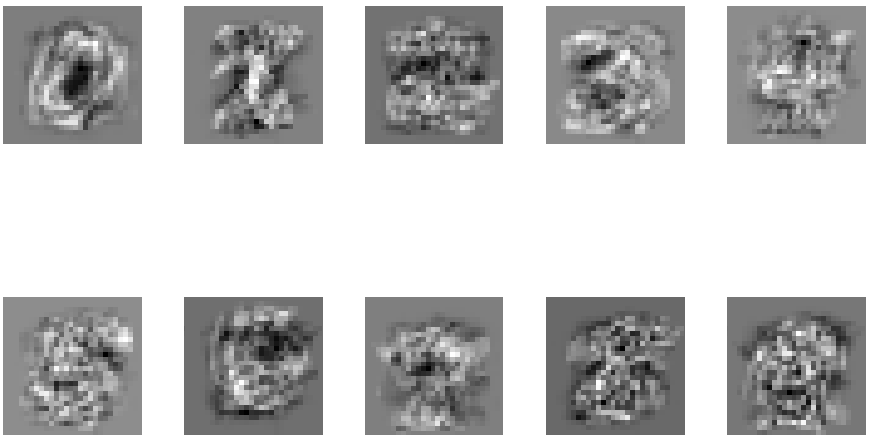

Figure 2: Coefficients $\beta_i,\ i = 0, ..., 9$ of the multinomial logistic regression.

In Figure 3 the confusion matrix for this method is shown, with an overall (resubstitution) error rate of 7.8%. In particular, we can see that we are able to easily distinguish the digits 0 (labelled as "10" in the matrix) and 1 with a low error rate, while the digits 2,5, and 8 have a larger error rate compared to the others. It seems reasonable as 1's and 0's have rather simple shapes.

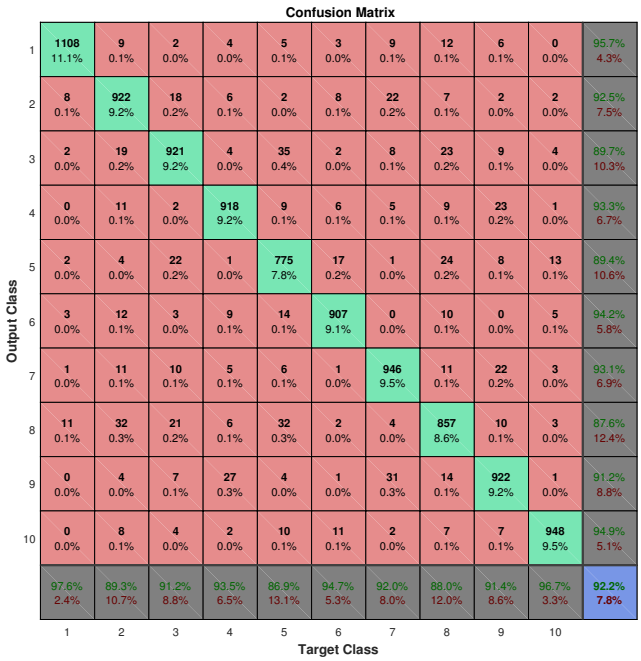

Figure 3: Confusion matrix for multinomial logistic regression.

In Figure 4 we show 400 digits using the t-Distributed stochastic neighbor embedding to visualize the result of the logistic regression classification when we used only 1000 images to estimate the model. The classification rate is 84.25%.

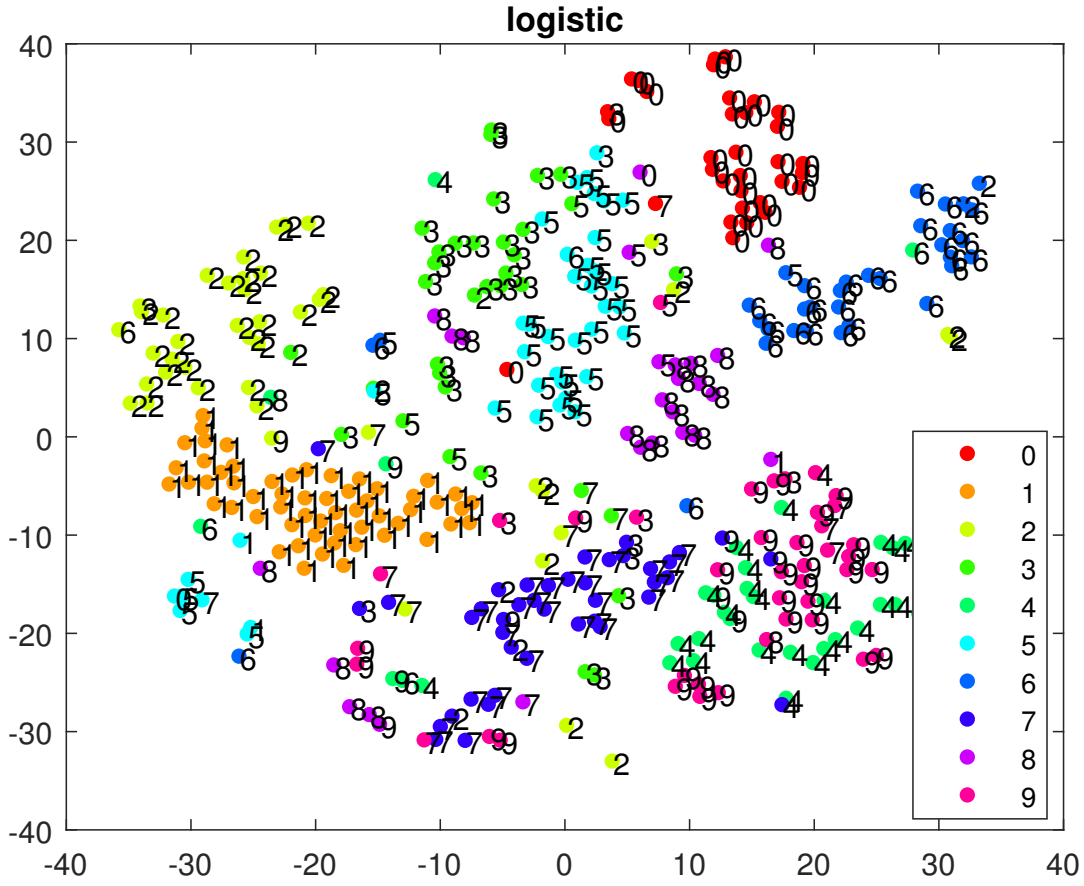

Figure 4: First 400 digits of the test set in the MNIST database. The color of each point represent the correct digit as summarized in the legend. The number close to each point is the predicted label using logistic regression

Support vector machine

In this section we consider support vector machines, which are nonlinear classifier, to classify the digits. In Figure 5 the confusion matrix is shown. We start by considering a binary problem, where the label for the two classes are $l \in \{1, -1\}$. Moreover consider again x to be the input vector. In the classic Fisher linear classifier we find the decision boundary of the form $y(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x} + b = 0$ (an hyperplane, which in 2 dimension is a straight line), and classify the data depending on which side of the plane they are with respect to the line defined by $y(\boldsymbol{x}) = 0$, i.e. we said that they are from class 1 if $y(\boldsymbol{x}) > 0$

and class -1 if $y(\boldsymbol{x}) < 0$. In SVM we do essentially the same, we define a function

$$y(\boldsymbol{x}) = \boldsymbol{w} \cdot f(\boldsymbol{x}) + b, \tag{9}$$

estimate $\boldsymbol{w}, b$ from the training set and classify new data by looking at the sign of $y(\boldsymbol{x})$. Again, the decision boundary will be defined by $y(\boldsymbol{x}) = 0$, and as $f : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ can be any nonlinear function, the decision boundary could be anything. On the other hand, in the feature space where $z = f(x)$ lives, $y(x)$ is a linear function of z and so it is a hyperplane in $\mathbb{R}^{d_2}$.

To estimate $\boldsymbol{w}$ and b we maximize the margin, i.e. the distance between any data in the training set and the decision boundary. Let r denote the index of the closest point in the training set to the decision boundary. Then we obtain

$$\begin{aligned} w &= \textstyle\sum_{n=1}^{N} \lambda_n l_n f(\boldsymbol{x}_n) \\ \lambda_r \left(\textstyle\sum_{n=1}^{N} \lambda_n l_n f(\boldsymbol{x_n}) f(\boldsymbol{x_r} + b \right) &= 1, \end{aligned} \tag{10}$$

where the λ's are the Lagrange multipliers which maximize

$$L(\boldsymbol{\lambda}) = \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m l_n l_m f(\boldsymbol{x}_n) f(\boldsymbol{x}_m) \tag{11}$$

subject to the constraints

$$\begin{aligned} \lambda_n &\geq 0, \ n = 1, ..., N \\ \textstyle\sum_{n=1}^{N} \lambda_n l_n &= 0. \end{aligned} \tag{12}$$

The function $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = f(\boldsymbol{x}_i) f(\boldsymbol{x}_j)$ is called a kernel function and is what we usually define instead of choosing the function f. Some examples of kernel functions are:

$$\begin{aligned} k(\boldsymbol{x}_i, \boldsymbol{x}_j) &= e^{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2} \text{ Gaussian} \\ k(\boldsymbol{x}_i, \boldsymbol{x}_j) &= \boldsymbol{x_i} \cdot \boldsymbol{x}_j \text{ Linear} \\ k(\boldsymbol{x}_i, \boldsymbol{x}_j) &= (1 + \boldsymbol{x_i} \cdot \boldsymbol{x}_j)^p \text{ Polynomial of order } p. \end{aligned} \tag{13}$$

SVM is a binary classifier which can be extended to handle multiclass data in many ways (one vs one, one vs all, and many more). Here we decided to use the Gaussian kernel function and a one vs all coding, where we consider 10 different binary classifiers. In each of these binary classifiers, one digits is labelled as a class (typically 1) while all the other digits fall into a different class, namely -1. The 10 separate binary SVM are trained on the training set, and then each digit in the test set is labelled as the class $\hat{k}$ given by

$$\hat{k} = arg \min_{k} \sum_{j=1}^{10} |L_{kj}| g(L_{kj}, y_j(x)),$$

where L_{kj} is the label (1 or -1) of class k in the classifier j, x is the image to classify, $y_j(x) = w \cdot \phi(x) + b$ is the score for the image x in the j-th classifier, and g is a loss function. In a simple binary SVM classifier we would classify a new image x depending on the sign of $y(x)$. The loss function $g(l, y) = max(0, 1 - ly)$ is such that if l, y have the same sign (i.e. the image is correctly classified) and $|y| > 1$ the loss is 0, otherwise the loss is a linear function of y. Note that g is not the function which is used to train the model. The overall error of this method is 6.4%

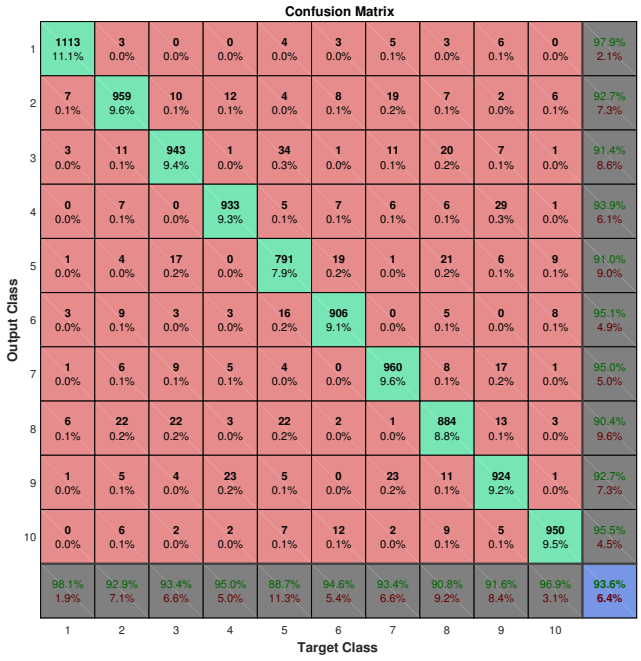

Figure 5: Confusion matrix for SVM.

In Figure 6 we show 400 digits using the t-Distributed stochastic neighbor embedding to visualize the result of the SVM classification when we used only 1000 images to estimate the model. The classification rate is 83%.

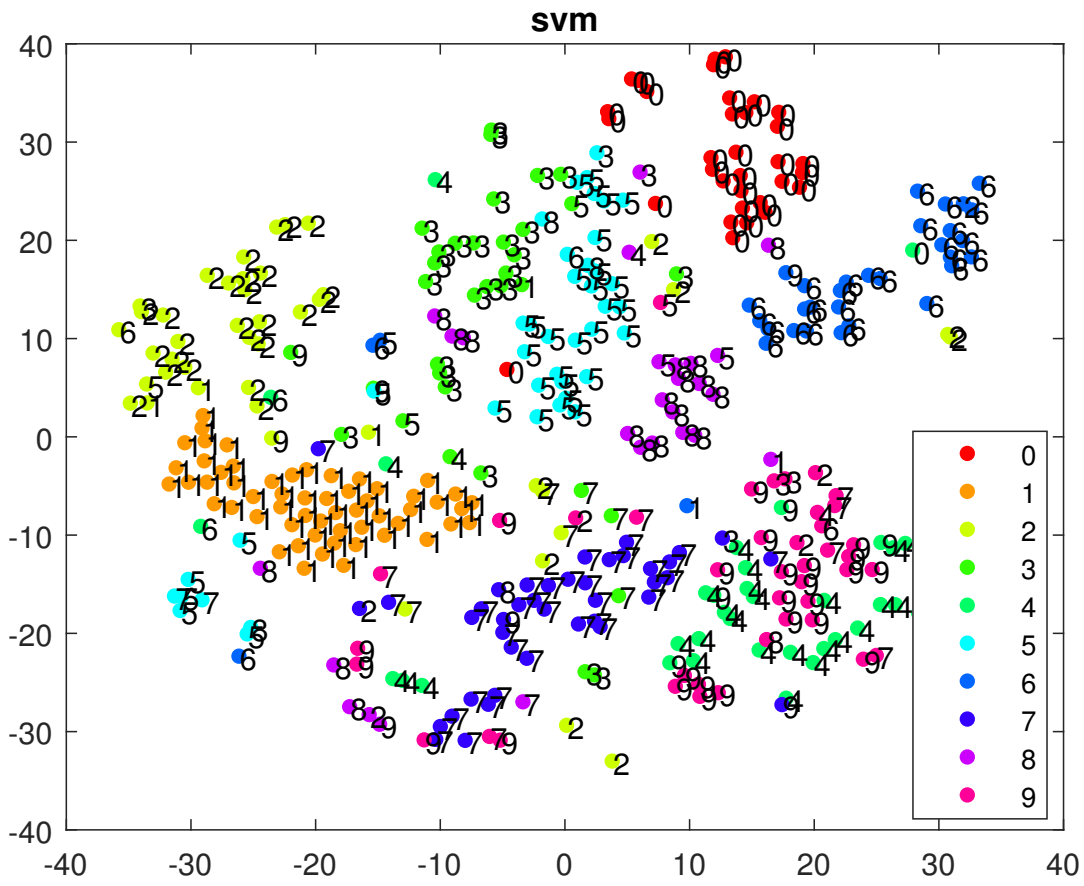

Figure 6: First 400 digits of the test set in the MNIST database. The color of each point represent the correct digit as summarized in the legend. The number close to each point is the predicted label using SVM

Neural networks

The simplest neural network consists of only one neuron. Typically, neural networks are made of different layers of neurons, where different neurons carry out different tasks. Here we will focus on a particular type of neuron, called the sigmoid neuron. A neuron works by taking an input x, for example a vector, and returning as output $\sigma(w \cdot x + b)$, where w is a vector of weights and b is sometimes referred to as the bias. σ is the sigmoid function defined by

$$\sigma_1(y) = \frac{1}{1 + e^{-y}}.$$

Other choices for σ are possible, and usually combining different such function in multi-layers provide a mixture of affine and non-linear transformations. The neural networks considered here consists of two layers, an autoencoder or a convolutional layer and a softmax layer, see Equation 14. The autoencoder tries to find a low dimensional representation of the data x in terms of some features that should be able to discriminate the input. The autoencoder acts in an unsupervised fashion and tries to approximate the identity function, in the sense that we want a function $h(w \cdot x + b) \approx x$ and $h(w \cdot x + b)$ has a smaller dimension than x. It is similar to principal component analysis (PCA), but an autoencoder is not limited to linear transformations. Here we used an autoencoder with 28×28 pixels images as input and output of dimension 100, corresponding to 100 different features used to represent the data. In Figure 7 we show the 100 features used by the autoencoder based on the MNIST training set.

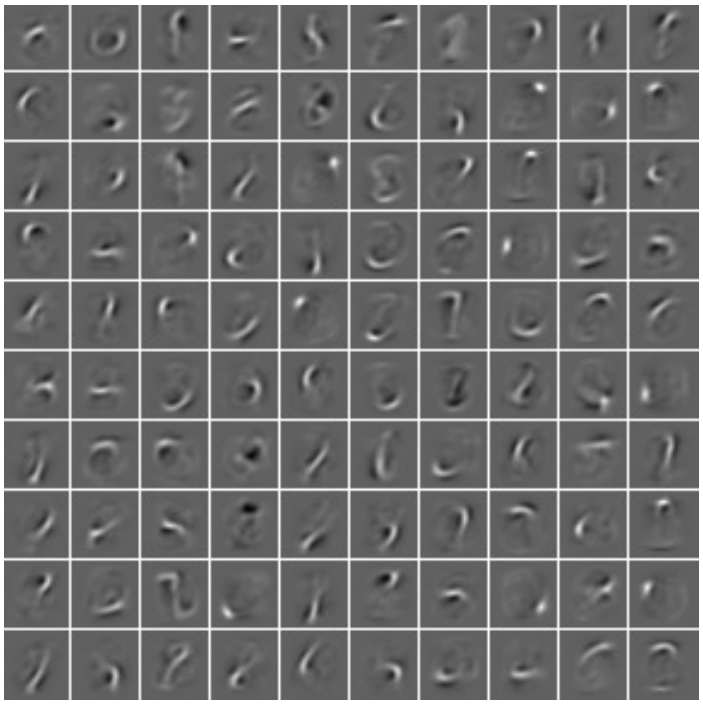

Figure 7: Plot of the 100 features used to represent the data by the autoencoder. Each image has size 28×28 pixels.

The convolutional layer applies a filter to the original input images. It convolves the image with a mask matrix to create a new map, i.e. we move the mask over the image

and sum the results of the elementwise multiplication of the two. Thus, a convolutional layer extract features of the numbers, as edges or small "curls" that could be useful to classify the different digits. A convolutional neural network is intuitively the network that resemble the most how a human would tackle the same problem. If I would try to describe how I distinguish a nine, I would say that two features/objects are important: a circle in the top, followed by a stroke of the pen going downwards. In the same way, a convolutional neural network tries to find local properties that identify the different digits. The difference between autoencoder and convolutional layer lies mainly in the connectedness of the network. The convolutional one applies a filter, which usually has a size which is smaller than the input image, so each neuron (or equivalently each pixel in the output feature map) depends only on a subset of the image. For example, if we would apply an averaging filter with size 5×5 pixels, then each neuron in this network will be connected to the 25 corresponding pixels in the input image. Instead, in an autoencoder we look for a global representation, i.e. every node in the hidden layer will be connected to every node (i.e. pixel) in the input image.

The softmax layer instead is the one that actually solve the classification problem. The name softmax comes from the softmax function that is used to produce the output, and is defined by

$$\sigma_2((y_1, \cdots, y_k) = \left(\frac{e^{y_1}}{\sum_{j=1}^{k} e^{y_j}}, \cdots, \frac{e^{y_k}}{\sum_{j=1}^{k} e^{y_j}} \right). \tag{14}$$

There exists a link between the softmax and sigmoid functions and logistic regression, which allows us to interpret the output of the neural network as the posterior probabilities of the different digits. To see this, consider a typical Bayesian framework where we model the conditional probabilities $p(x|k)$ (the probability distribution of x given that it comes from class k) and the prior probabilities of each class $p(k)$. Consider in the following the case of only two classes, and similar arguments hold for the more general case. Using Bayes' theorem we have

$$p(k|x) = \frac{p(x|1)p(1)}{p(x|1)p(1) + p(x|2)p(2)} = \sigma_1(a), \tag{15}$$

where $a = ln\frac{p(x|1)p(1)}{p(x|2)p(2)}$ and again σ_1 is the sigmoid function. Let's assume that the density of $p(x|k)$ are Gaussian with same covariance matrix in each class, i.e.

$$p(x|k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \exp\{-\frac{1}{2}(x - \mu_k)^T \Sigma^{-1}(x - \mu_k)\}.$$

Then we obtain from the definition of a that

$$a = (\mu_1 - \mu_2)\Sigma^{-1}x - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + ln\frac{p(1)}{p(2)}, \tag{16}$$

which is a linear function of x, and would have been quadratic if we had chosen different covariance matrices for different classes. Now if we replace a in Equation 15 we obtain

that the posterior probability $p(1|x)$ can be written as the sigmoid function applied to a linear function of x. Thus, we have obtained the link to the logistic regression. In this case, the boundaries of the decision regions will be linear as they are defined as those regions where the posterior probabilities are equal, and those are given by linear functions of x.

An illustration of the network used is shown in Figure 8: we give in input an image of $28 \times 28 = 784$ pixels which will be then represented in the first layer in terms of the 100 features selected. Then, this 100 features representation is used in the softmax layer, which gives us as output the vector of the posterior probabilities for each of the 10 digits for the input image.

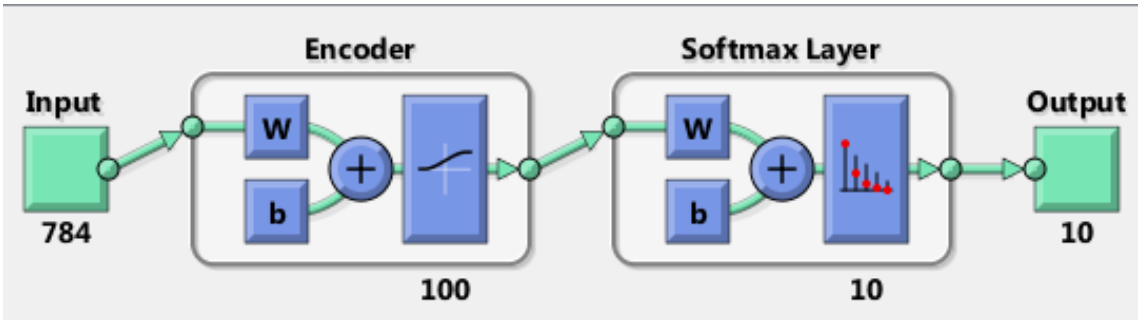

Figure 8: Scheme of the network used in this part of the project.

We trained the network on the 50000 images in the training set and tested it on the 10000 images of the test set in the MNIST database. In Figure 9 we report the confusion matrix relative to the test set. We see in the right bottom corner that the overall error rate is 2.3%, corresponding to 231 misclassified digits out of the 10000 tested.

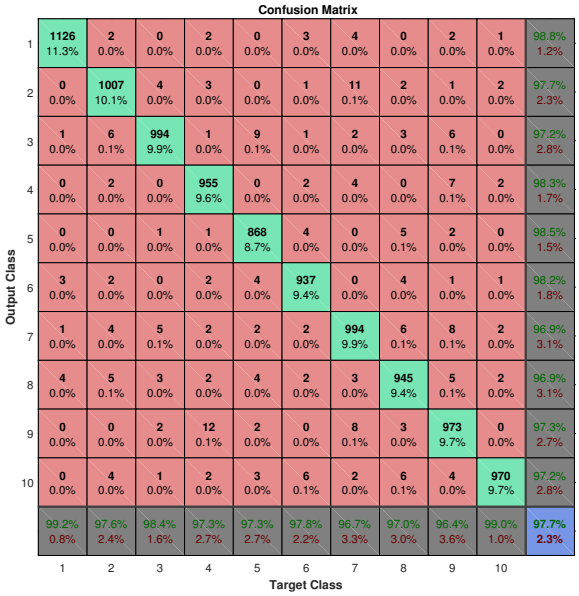

Figure 9: Confusion matrix for the neural network with an autoencoder.

In Figure 10 we plot the confusion matrix for the convolutional neural network. Even though this network is still simple, it proves to be very good at recognizing digits with an overall error rate of 1.5%.

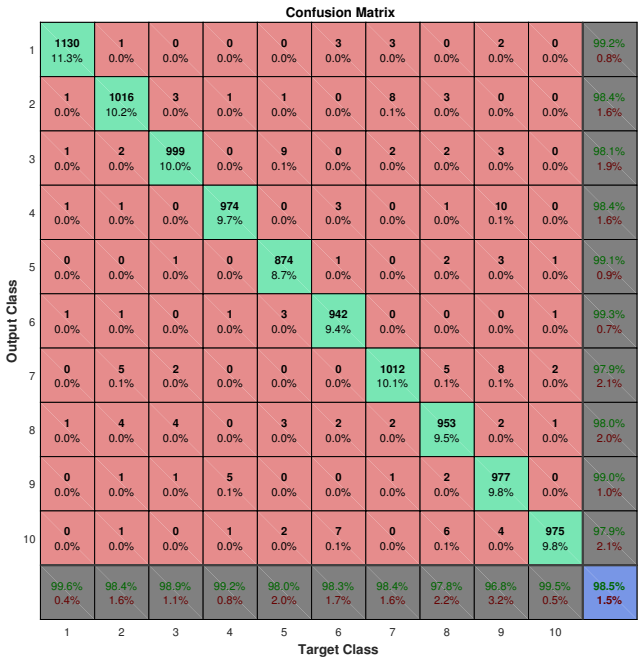

Figure 10: Confusion matrix for the convolutional neural network.

In Figure 11 we show 400 digits using the t-Distributed stochastic neighbor embedding to visualize the result of the convolutional neural network classification when we used only 1000 images to estimate the model. The classification rate is 85.5%. We can see that different methods perform differently when we change the size of the training set. As expected, the accuracy of SVM, CNN, and logistic regression increases as the sample size increases.

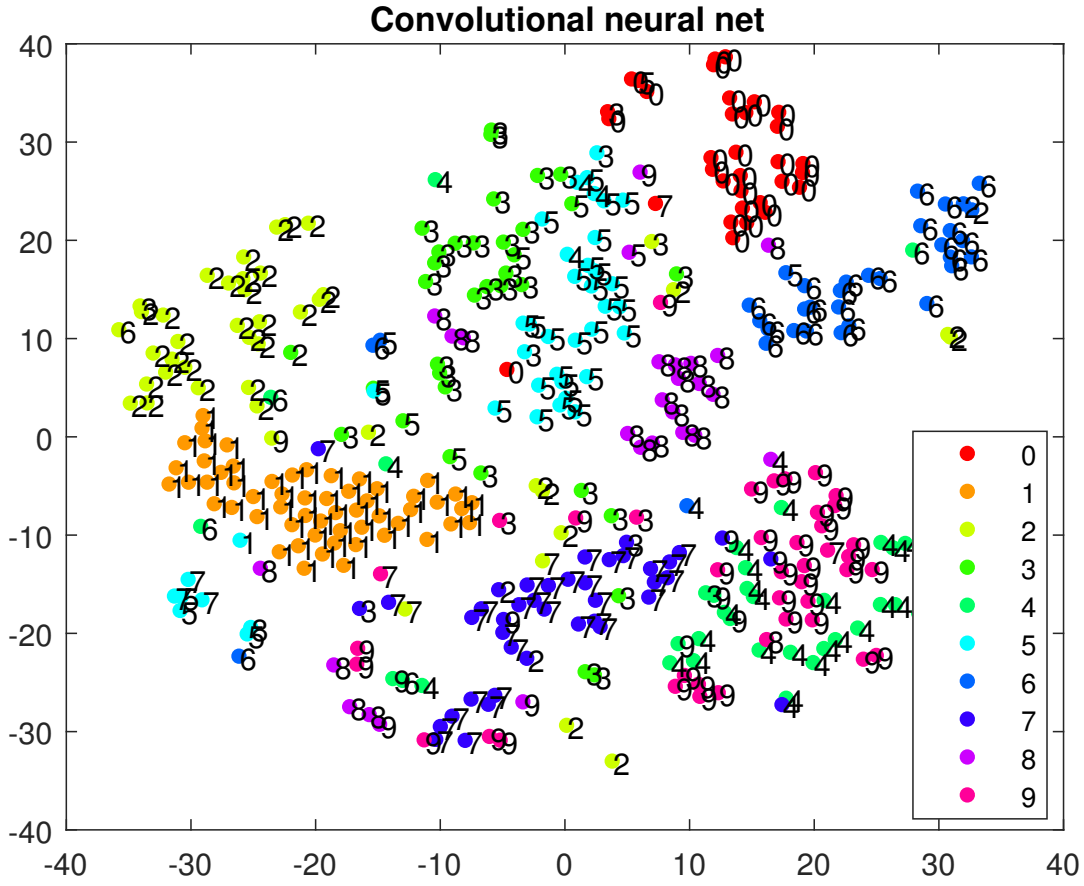

Figure 11: First 400 digits of the test set in the Mnist database. The color of each point represent the correct digit as summarized in the legend. The number close to each point is the predicted label using logistic regression

3 Warping and finding average digits

Consider a set of observations $Y_1, ..., Y_n$ corresponding to n 28×28 images of the same digit, for example the digit 1. The goal of this section is to find an "average" digit μ. Suppose at first that we know the image μ. We warp an image under a transformation $f(x) = (f_1(x), f_2(x))$ by defining how f acts on the mesh of $q+1$ points given by

$$p_{ij} = \left(\frac{28i}{q}, \frac{28j}{q} \right), \quad i, j = 0, ..., q. \tag{17}$$

The warping f is then specified by the set of $(q+1) \times (q+1)$ values

$$f(p_{ij}) = (f_1(p_{ij}), f_2(p_{ij})), \quad i, j = 0, ..., q. \tag{18}$$

Since in general $f(p_{ij})$ will not be integers, we will use bilinear interpolation to obtain the pixel values of the warped image. The transformation f is found by minimizing the squared pixelwise differences between the warped image, denoted by Y_f, and the template image μ.

$$f = \text{argmin}_g \sum_{\text{all pixels } x} (Y_g(x) - \mu(x))^2 \tag{19}$$

which is equivalent to maximizing a Gaussian likelihood. However we choose to add a penalization term to avoid "large" transformations

$$f = \text{argmin}_g \sum_{\text{all pixels } x} (Y_g(x) - \mu(x))^2 + \lambda \sum_{i=1}^{2} \sum_{j=1}^{2} \sum_{k=1}^{2} \int \left(\frac{\partial g_i}{\partial x_j \partial x_k} \right)^2 dx \tag{20}$$

where the integral on the last term is approximated by using finite differences to approximate the partial derivatives of the transformation over the image and λ governs how strong the penalization will be. This penalization term corresponds to the bending energy of a thin plate spline. This has the effect to favor the choice of f "close" to an affine transformation, as for this type of transformations the penalization will be 0 (otherwise is always positive). In Figures 12-13 we present an example of warping an handwritten digit 2 to another handwritten digit 2 and the corresponding grid of deformation with $q = 7$ points.

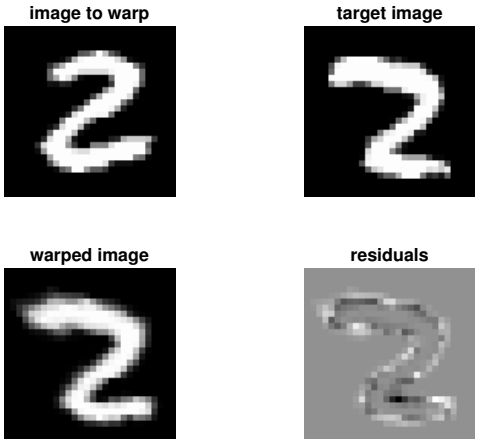

Figure 12: Two original digits "2" on the top row. The goal of this example is to see how well we can "align" the digit on the top left to the one on the top right figure. Warped digit on the second row together with the difference between the warped and target image.

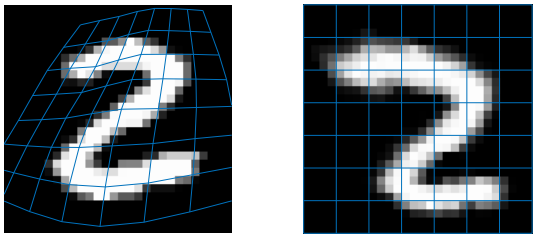

Figure 13: Original and warped digits with corresponding grids with $q + 1 = 8$ points per side. The target image used to find the warp is the one shown in the top right image of Figure 12

Since the template μ is not know, we will start by considering μ^0 as the pixelwise mean of $Y_1, ..., Y_n$. Then, we warp each of the Y_i to μ_0 with transformation f_i. We

define μ^1 to be the mean of $Y_{f_1}, ...Y_{f_n}$. We repeat this process until either a maximum number of iteration has been reached or the change in the template image (compared to the previous iteration) is smaller than a threshold value. A maximum number of 100 iterations for each image has been chosen, however most of the time few as 3-4 iterations were enough to converge. The limit of 100 iterations is low due to the fact that the number of images considered here is quite large. The penalization parameter λ and the number of points in the grid q have been first chosen to be fixed, $q = 7$ and $\lambda = 1$, and then iteratively increasing $q = 7, 14, 28$ and decreasing $\lambda = 100, 10, 1$ to allow more distorted transformations. However, the iterative approach did not seem to improve the results significantly, probably due to the simplicity of the images taken in consideration.

We present now the results based on the training set of the MNIST data, where we have about 1000 observations of each digits. For the sake of visualization, we present only the first 100 digits in the figures, but the average is the result based on all the available observations. In Figure 14 we report the original handwritten digits corresponding to the digit 5.

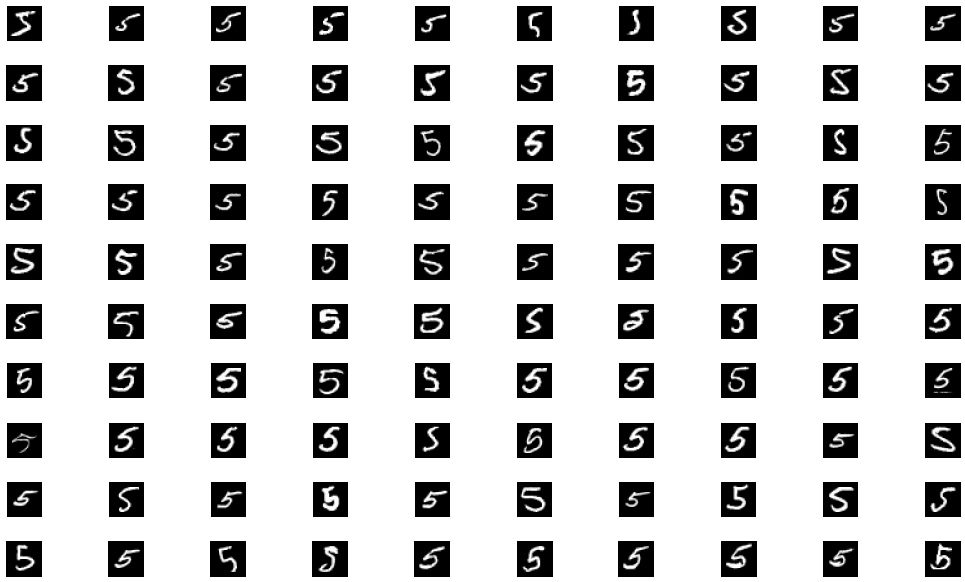

Figure 14: First 100 original handwritten digits "5" as presented in the MNIST data. This is just subset of the data used to estimate the average digit, but for visualization we consider only these.

In Figure 15 we summarize the warped digits shown in Figure 14 to the average digit

5, shown in Figure 16 below.

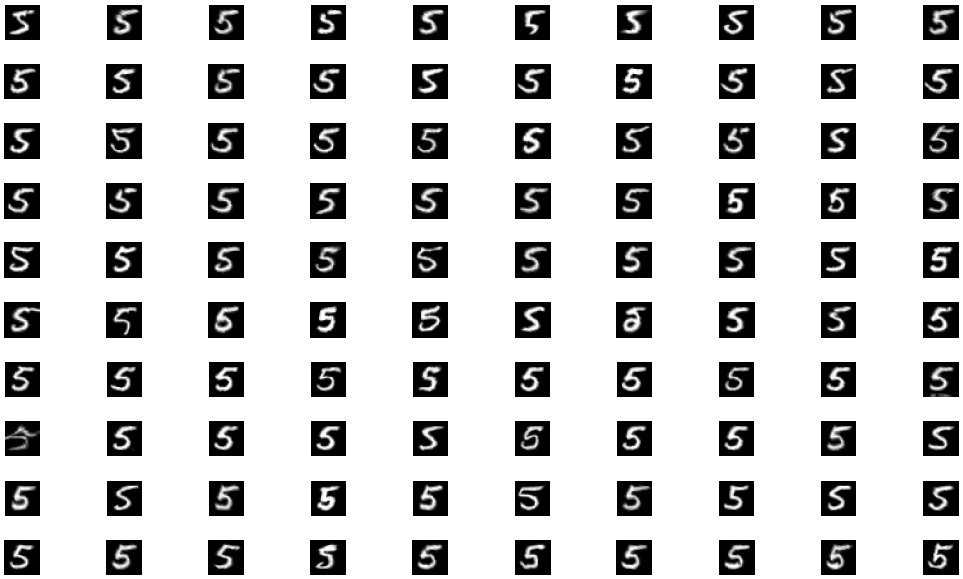

Figure 15: Warped digits corresponding to the original digits pictured in Figure 14.

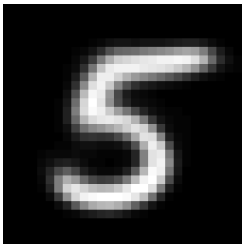

Figure 16: Mean digit 5 as obtained from the warping and averaging of the warped digits.

Below instead we look at the digit 4:

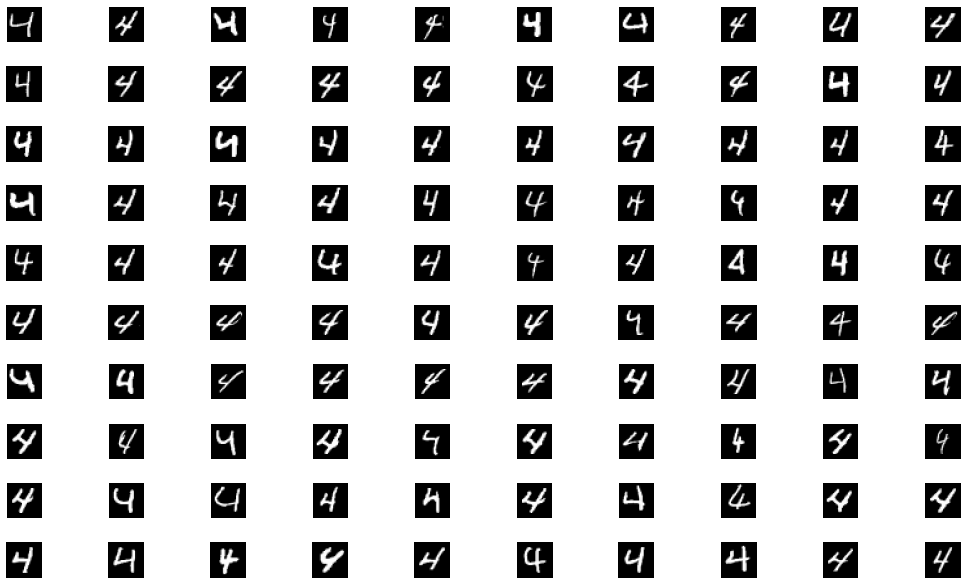

Figure 17: First 100 original handwritten digits "4" as presented in the MNIST data. This is just subset of the data used to estimate the average digit, but for visualization we consider only these.

In Figure 18 we summarize the warped digits shown in Figure 17 to the average digit 4, shown in Figure 19 below.

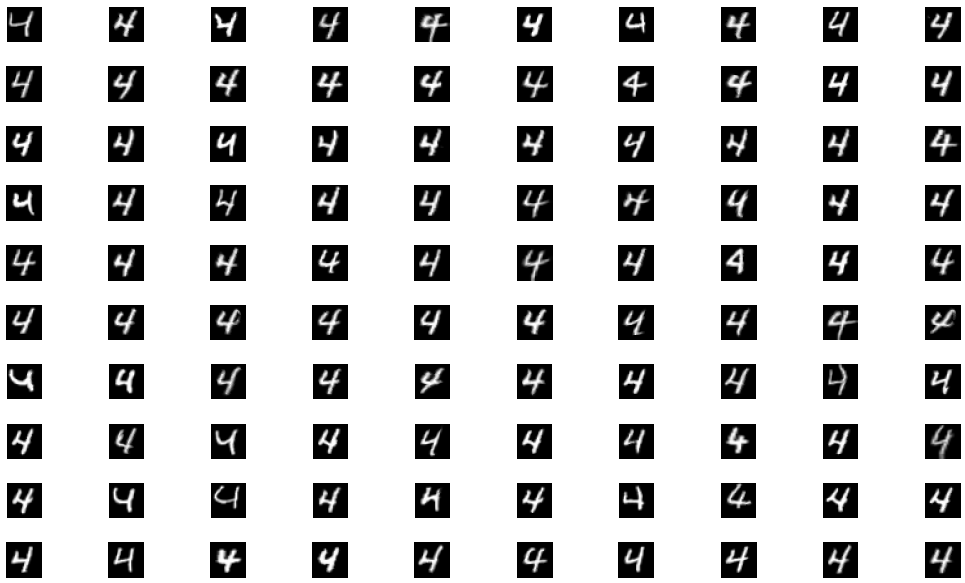

Figure 18: Warped digits corresponding to the original digits pictured in Figure 17.

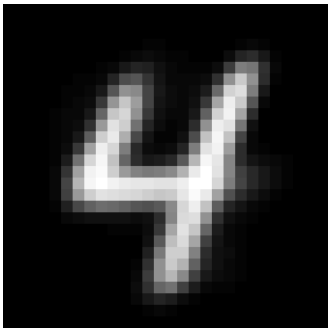

Figure 19: Mean digit 4 as obtained from the warping and averaging of the warped digits.

The average digits can be used to classify new images. Consider for a new image $Y(x)$ the transformations to the average digit $j = 0, \ldots, 9$ called in the following $\mu_j(x)$:

$$f_j = \text{argmin}_g \sum_{\text{all pixels } x} (Y_g(x) - \mu_j(x))^2 + \lambda \sum_{i=1}^{2} \sum_{j=1}^{2} \sum_{k=1}^{2} \int \left(\frac{\partial g_i}{\partial x_j \partial x_k} \right)^2 dx \qquad (21)$$

The penalized log-likelihood corresponding to the transformation f_j can be used as a measure of dissimilarity between the image $Y(x)$ and the average digit j. Thus, we can classify the image $Y(x)$ as the digit corresponding to the average digit which has the maximum penalized log likelihood. In Figure 20 we applied this method to 197 fours and fives (110 fours and 87 fives) with the digit averages presented before in Figures 16-19. Only two fours and 4 fives are missclassified, however this approach seems more costly compared to support vector machines and neural networks. In fact, all of them needs some type of training (either one trains the machine/network or estimates the average digits), but in the classification step warping is more computationally extensive than the other, as we need to further minimize a complex function of many parameters for each new image.

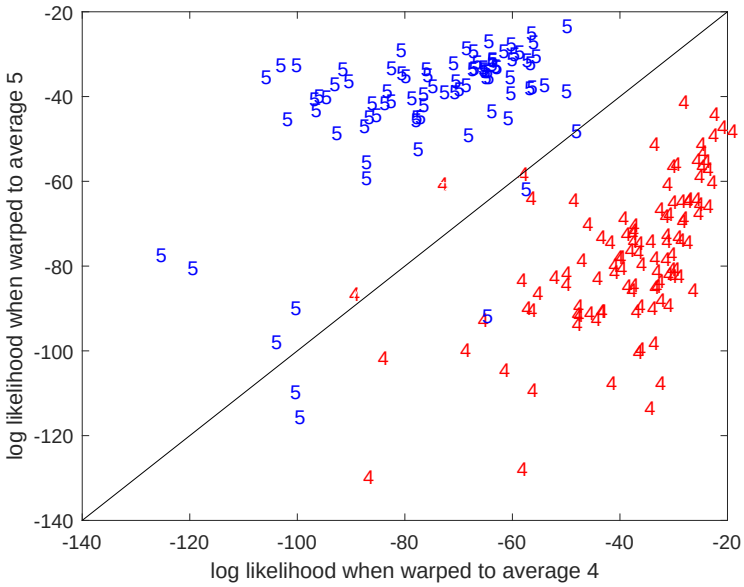

Figure 20: Log likelihood of digits "four" and "five" when warped to the estimated average digits shown in Figures 16-19. 110 fours have been considered and 87 fives.

References

Bishop, C. M. (2006) *Pattern Recognition and Machine Learning*. Springer, London.

Glasbey, C. A. & Mardia, K. V. (2001) A penalized likelihood approach to image warping. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **63**, 465–492.

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998) Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**, 2278–2324.

van der Maaten, L. & Hinton, G. (2008) Visualizing data using t-SNE. *Journal of Machine Learning Research*, **9**, 2579–2605.