Coding Project 4: Teaching a Computer to Recognize Written Numbers

Wietse Vaes

Abstract

Everyone knows at least one person who has unreadable handwritten notes. For most of my friends, that person is me. Instead of squinting your eyes until they hurt, one might want to turn to machines, which might be able to "translate" someones writing into a readable pdf document. In this paper we would like to be able to have a computer recognize written symbols, numbers here, and categorize them as these symbols in the computer itself. After finding the edges of the symbols in a training set, using wavelets, and processing them even more with singular value decomposition, we can find a threshold to categorize them into different groups using linear decomposition analysis. Thereafter we test our thresholds on another set of processed handwritten numbers to see how well it does.

1 Introduction

Given a data set, which might or might not be processed, we would like for a computer to tell us something about the input, while it has never seen it before. In this case, we would like a computer to recognize random numbers written by random people. We would thus like to teach the machine a possible way of recognizing new numbers. We do this by training it with data sets where we know the answers for in order for the computer to have a good way of recognizing which data set represents which number. This is pretty much machine learning, we train a machine with training data and thereafter we hope our method used to train the computer is good enough to recognize something in new data sets.

In this paper we will talk about wavelet theory, in order to find the edges of the numbers, thereafter we use n-rank approximations, to approximate these edge such that we're not to focused on that data set, to potentially have more general data set. We use these techniques for two numbers on a training set to find a projection for generally any data set of numbers and a corresponding threshold to identify certain numbers. This will be done using linear discrimination analysis. Afterwards, we show the results of these two numbers and expand it to all ten. Eventually we hope that the computer does a good job at recognizing my handwritten numbers.

2 Theoretical Background

In order to recognize written numbers, we'd like to focus more on the shape of the number and thereafter approximate this image such that it's not relying on this specific shape, but an approximation of that shape. In order to find the edges, we use wavelets, thereafter we approximate them using n-rank approximations of the data sets. Then we'd like to find a projection of this image such that the different numbers are as clearly separated as possible, we do this with linear discriminant analysis. Eventually, we find a threshold on these projections such that we can section them of and categorize them. Most of these topics will be discussed in this section.

2.1 Wavelet Theory

Before we use the linear discriminant analysis (LDA), we need to process the images to something we want to use. In this case, we would like to only have the edges of the images. These edges would give less variational images in order to let LDA do its job better. More on that later. There are a lot of ways to detect edges, such as blurring and then finding the high differences in this blurred image, Laplacian kernels and more. For this paper we will use wavelets, more specifically Haar wavelets. Before we speak of Haar wavelets in two dimensions, we'll go over Box wavelets in one dimension. These wavelets are just functions that are piecewise constant and 0 most of the time. In particular, the box wavelets are one over an interval and zero over the rest. Say we always define our domains in a d-dimensional space over $[0,1]^d$. First, define the ϕ as one over interval $[0,1]$ and zero elsewhere:

$$\phi(x) = \begin{cases} 1 & 0 \leq x \leq 1, \\ 0 & \text{elsewhere.} \end{cases}$$

We would now like to get these functions over smaller intervals of $[0,1]$, we then use the following functions:

$$\phi_i^j(x) := \phi(2^{-j}(x+i)), j \in \mathbb{N}, i = 0, \ldots, 2^j - 1.$$

These functions are better represend through figure. In figure 1, one can see the case where $j = 2$, with all i.

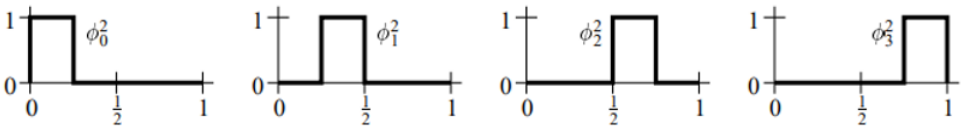

Figure 1: One dimensional Box wavelets where $[0,1]$ is split into four parts.

Note here that these are clearly orthogonal to each other and are easily made orthonormal by multiplying by 2^j. Haar wavelets are now give by a function that are 1 on one half on an interval and -1 on the other half. We define functions

$$\psi_i^j(x) := \psi(2^{-j}(x+i)),$$

with

$$\psi(x) = \begin{cases} 1 & 0 \leq x \leq \frac{1}{2}, \\ -1 & \frac{1}{2} \leq x \leq 1, \\ 0 & \text{elsewhere.} \end{cases}$$

For $j = 1$, the Haar wavelets can be seen in figure 2.

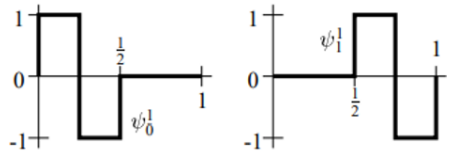

Figure 2: One dimensional Haar wavelets where $[0, 1]$ is split into two parts.

Note how these functions are also orthogonal to each other and ϕ_i^j! Thus, using these functions, we could describe a piecewise constant function using linear combinations of these functions. We just need to find these coefficients. It is easy to see however that we could describe a function over an interval as the mean over that interval and then the distance from this mean to and the left values of the interval and the right, which is the same, but different sign (exactly what the Haar wavelet has). We then get a description of the two numbers as the mean and the value which you can do plus and minus in order to get your original values. For example:

Resolution	Averages	Detail coefficients
4	[9 7 3 5]	
2	[8 4]	[1 −1]
1	[6]	[2]

Here we see that we could get 9 and 7 by adding and subtracting the detail coefficient 1 from the average value 8. We could do this multiple times to even get only one mean and the rest are differences. Thus we could describe a function with constant values, over the same length interval, $[9, 7, 3, 5]$ by $[6, 2, 1, -1]$ or $[8, 4, 1, -1]$. Thus it would look like [means,differences], these will be the coefficients of [Box wavelets, Haar wavelets]. Note that it does not reduce the amount of information stored, nor does increase it! This information tells us where big and small differences happen, which is usefull. Say one wants to compress data, then combine the intervals with low difference! Want to find edges/shocks? the points with big differences tell you where a big shock is. Of course, these differences must also be multiplied by the normalizing coefficient for the wavelets in order to judge them with each other.

One might ask: what does this have to do with recognizing numbers? But one must then remember that an image is just a two dimensional piecewise constant function (in black and white). The use of Box and Haar wavelets in two dimensions are closely related to the one dimensional method. In order to find we means and differences here, two options come to mind: first find the mean and differences over all rows

and then all columns, or do it in an alternating manner. Depending on the way you calculate these, your wavelet functions will be different. We will shortly only talk about the standard decomposition and construct using wavelets here, this uses the first way of thinking (first all rows, then columns). We will mainly do this by look at images. In figure 3, one can see how we get the coefficients of the wavelets following the standard deconstruction:

Figure 3: Standard approach to calculating the coefficients for the Box and Haar wavelets in the two dimensional case.

Where the mean value of the whole painting is found at the bottom left. Thus, here we would use a Box wavelet over the domain of the whole image. To the right of this element, we find the differences in the column manner and above it we find the differences in the row manner, diagonally we get more finer differences where both rows and columns are combined in order to get the differences in both columns/rows. We could use these as coefficients for combinations of wavelets, which we can see in figure 4, to "undo" the transformation. From there on we can do this again with this "reconstructed" image, but now we use more than 3 coefficients to get the next reconstruction, we use $3 * 4^1$ coefficients now. We can look at this as a 4-couple of wavelets to reconstruct again. Thereafter, we use $3 * 4^j$ coefficients at step j until we get our image back. This is very vague to read, but we hope figure 4 does an excellent job at portraying what is meant.

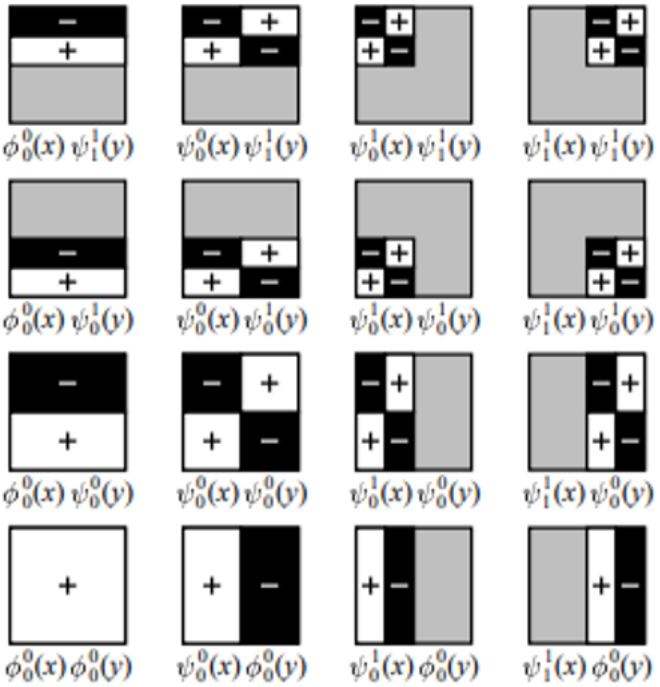

Figure 4: The basis used in the standard wavelet approach in the two dimensional case. These are multiplications between the Box and Haar wavelets in one dimension, but using x and y.

As said before, we can now use wavelet theory to find the edges and thereafter process it even more by using SVD, which is explained in my last paper (reference). Computationally this is also good to use as, using a lifting scheme, we use no extra memory, it's efficient, the algorithm is easily reversible and expansion to a higher dimensionality is rather straightforward.

2.2 Linear Discriminant Analysis

By having processed the data using the previous section, we would now like to categorize this data into groups. For this purpose, we look at the Linear Discriminant Analysis (LDA). The gist of this is finding the best projection onto a lower dimensional line such that we can split this line into clear sections where most of one group of data lives and the other in another section. Thereafter we can use the mean and variance of one group to find a threshold as to find clear sections to categorize new projected data and do it correctly. First we try to find that projection for categorizing two classes C_1 and C_2.

In order to categorize two different classes, we take the same amount data points of these classes from our training data. After this, we want to find the mean value of each picture separately. Say we have n images of each class, then $\mu_1\mathbb{R}^n$ is the mean of the n images of class one and μ_2 the mean of the other class. We can now find the variance of the difference in the means, or the scatter matrix for between-class S_B:

$$S_B = (\mu_2 - \mu_1)(\mu_2 - \mu_1)^T,$$

and the variance of the data sets, or the scatter matrix within-class S_W, as the sum of the covariances within the two classes:

$$S_W = \sum_{j=1}^{2} \sum_{\mathbf{x} \in C_j} (\mathbf{x} - \mu_j)(\mathbf{x} - \mu_j)^T.$$

With these two matrices, we would like to find a projection $\mathbf{w}$ such that the ratio of the projection of S_B and S_W reaches a maximum, thus

$$\mathbf{w} = \arg\max_{\mathbf{w}} \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w} S_W \mathbf{w}}.$$

This comes down to solving a generalized eigenvalue problem

$$S_B \mathbf{w} = \lambda S_W \mathbf{w},$$

we the eigenvalue λ indicate the importance of this projection basis, therefore we will be using the eigenvector of the largest eigenvalue as our projection $\mathbf{w}$. To have unicity, we normalize these vectors. Once we have this, we can project our data with it, by multiplying our data set by the transpose of the projection, this results in one number per image. Using our projected classes, we can find the mean of them. With this we can adjust our projection basis, as it might be the positive of what we had above or the negative, both would give the same properties. Based on how the experiment is set up and the means of the projected classes, we might want to multiply every by -1. Thereafter we find the threshold to differentiate between classes. Multiple ways of finding this thresh hold exist, but we'll use the midpoint between where most of the data is separated.

Now that we know how to differentiate between two classes, differentiating between multiple classes can follow from expanding the method above or basing it on the two class method. For example: if we want to differentiate between N classes, we can do this process N times for differentiating class C_j and everything but class C_j. Another way we could *perhaps* do it, is by splitting up the training data set into two sets, finding the thresholds for $\binom{N}{2}$ combinations of cases using different features and "testing" it on the second part of the training data such that we maximize the success rate. The difficult part would be to combine these thresholds, but we would use a lot more of the original data than the other method. This as the previous method of finding the threshold is limited by the amount of one type of number and gives a disproportionate mean for "all the rest".

3 Results

Using MATLAB, the above theory will now be applied to the famous data set of Yann LeCun. First we will find a threshold for two numbers, namely, zero and one. However, before that we process the whole data set. First we find the edges and thereafter the n-rank approximation of the edges. As to which rank we will be using is a bit of a guessing game. In order to make this decision we plot the energies up

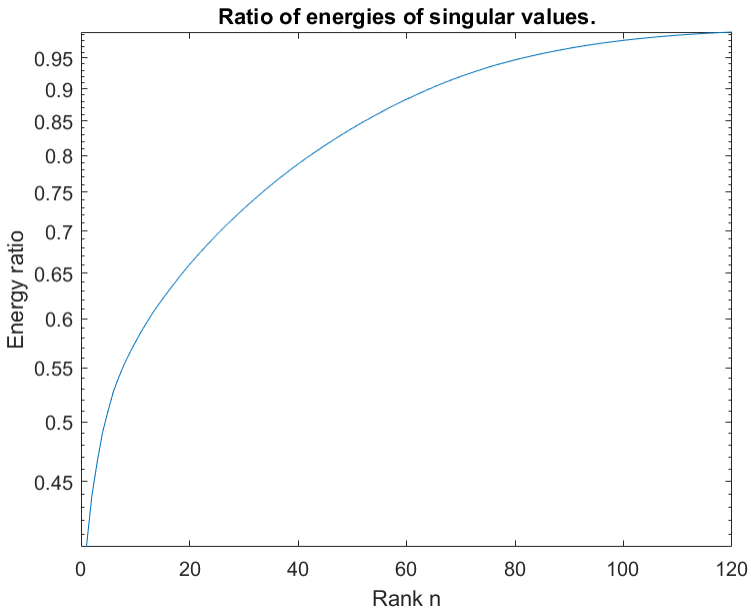

Figure 5: The ratio of total energy remaining when we use a n-rank approximation of our training data set up to rank 120. It is not a rapid curve, therefore a lot of ranks would possibly do a good job. The rank we would want depends on which numbers we want to compare, for this would could use an optimization algorithm.

to the 120^{th} singular value, which can be found in figure 5.

Here we clearly see that a lot of orthogonal basis have importance when composing the training data set. Thus, for different numbers, the rank of the matrix might need to be totally different. For the two cases where we choose one and zero, we choose a 15-rank approximation, where keep about 55% of the information. This gives us the best success rate. In general, we could take n as variable and try to maximize the success rate for the training data. One can now get the 15-rank approximation at the edges and this gives us figure 6.

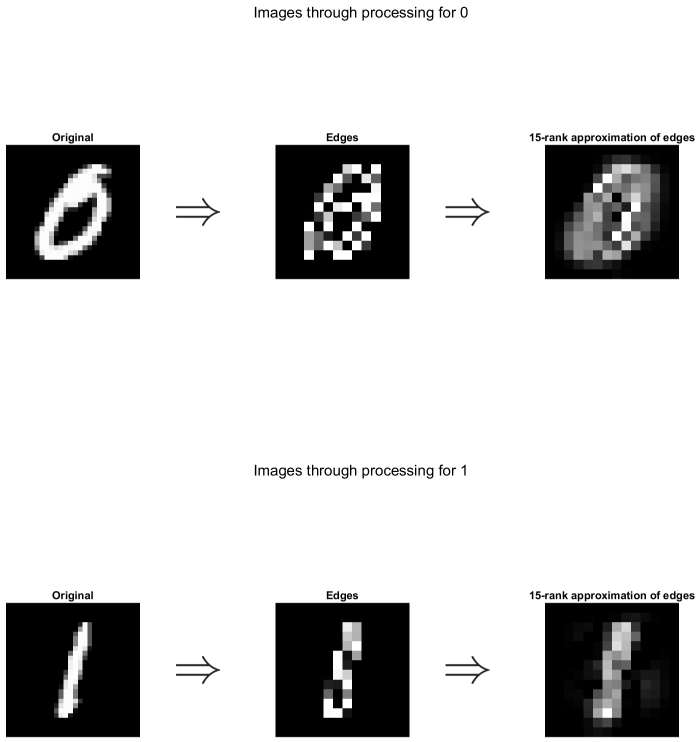

Figure 6: Processing two individual images of zero and one. Note that the edges of 1 looks like something else, but this is just because of the amount of pixel we compressed it too.

It's clear that the edge detection does something, It might not look like it's doing a good job, but that is just because of the resolution of the images and the fact that the wavelet transform in MATLAB does not keep this resolution. The n-rank approximation clearly does do what we would expect, it looks like it, but it's a bit more vague/smudged.

Using the Linear discriminant analysis we find a threshold of -1.55 for the numbers zero and one. Figure 7 represents the transformed data that we got by projecting the data set containing only the zero and one images with projection $\mathbf{w}$:

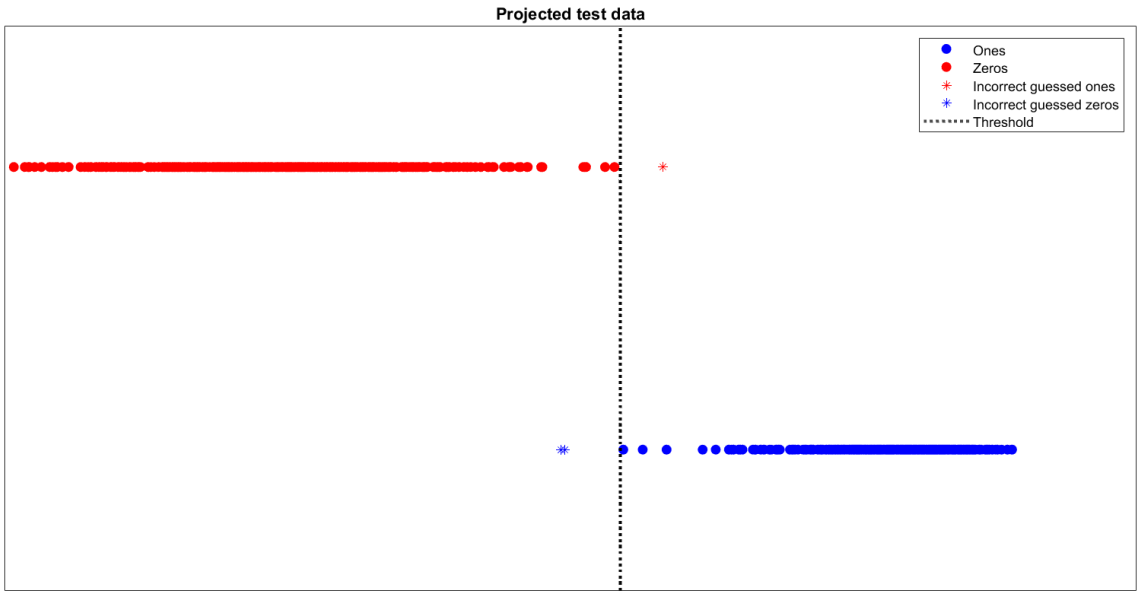

Figure 7: The projection of the test data containing only zeros (red) and ones (blue) images using a 15-rank approximation of the training data and the projection $\mathbf{w}$ gotten by using LDA on our training data. After calculating a threshold (the dotted line), we found that about 99.7% of numbers are correctly categorized. The stars will not be categorized correctly.

Here we see that, after projecting the test data, a lot of the projections are on the correct side of the threshold (the dotted line), only .3% are categorized incorrectly. Therefore this projection and threshold might do good in real life where we don't know the answer.

In order to expand this two-case method, I chose to compute the features, thresholds and projections $\mathbf{w}$ of all 45 different combinations of numbers such that we had an optimal success rate when we divide the training data into 2 separate sets. Thereafter, I go through all of the combinations again, use the feature for which we had optimal success and project the test data with the corresponding projection $\mathbf{w}$. Based on the corresponding threshold, we then chose which of the two numbers, in that combination, it resembled more. Eventually we chose the number which was chosen the most in all the combinations. Without implementing it, one problem is obvious: what if multiple numbers are chosen the same maximal amount. In order to deal with this, we could check which number wins when we compare these two

numbers themselves. When implementing this, about 46% were undecided. This is a very big number, thus the last step definitely needs to be implement. However, this method also does not work well for the other, already decided, numbers. It gives a general success rate of 10% and for the decided numbers a rate of about 18%. This method is atrocious, but I'm sure it could be made better. The method is, however, not totally worthless as it gives us an insight of which numbers are difficult to differentiate. The worst optimal success rate was about 90% for differentiating 3 and 5. Others that are difficult to differentiate are 4 and 9, and 1 and 7.

Even though the algorithm above did not give good results, we might want to use on my own handwriting. An external problem, that should also be implemented when developing a better method, is that Europeans write their numbers differently. Figure 8 shows how I write my numbers.

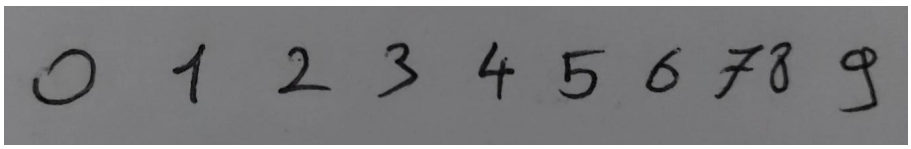

Figure 8: Handwritten numbers of a European. Note that 1, 2, 4 and 7 are very different from how an American writes it.

It is clear that they're not exactly the same as how Americans usualy write it. The 1 has an extra line, the 2 does not have a loop in the bottom left, the 4 has a short and longer diagonal line at the top and the 7 has a line in the middle as well. Surprisingly, the algorithm identified zero correctly. The rest did not work.

4 Conclusion

Using the famous data set of Yann LeCun, we were able to easily categorize zeros and ones. This was done by first processing the images by finding their edges, using wavelets, and computing an n-rank approximation. Thereafter, we used linear discriminant analysis to project onto a space where the data was separated the most based on means and variances. Eventually we found a threshold to categorize processed and projected data into zeros or ones. This method was 99.72% accurate when testing it on a different data set. Next we tried to categorize all 10 numbers by finding thresholds and projections for optimal results on training data. This method did not do well and thus needs more refining. It was undecided for about 46% of the data and of the decided numbers, only 10% was correct. One improvement is to put the undecided numbers through the projection associated with finding the difference between these two numbers. Lastly, we remarked that a good method to recognize written numbers should also keep different handwriting styles in mind such as European-American. This is easily fixed by taking a large enough sampling population.

Acknowledgment

Professor dr. Ingrid Daubechies' contributions to wavelet theory were immense and without her, I would not be able to brag about having come from so close to her hometown. Professor dr. Nick Michiels also deservers an honorable mention as he provided the course where I got the wavelet pictures from.