Treball final de grau

GRAU D'INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Wavelet pooling for convolutional neural networks

Autora: Aina Ferrà Marcús

Directora:	Dra. Petia Radeva
Co-director:	Eduardo Aguilar
Realitzat a:	Departament de Matemàtiques i Informàtica
Barcelona,	June 27, 2018

Abstract

Wavelets are mathematical functions that are currently used in many computer vision problems, such as image denoising or image compression. In this work, first we will study all the basic theory about wavelets, in order to understand them and build a basic knowledge that allows us to develop another application. For such purpose, we propose two pooling methods based on wavelets: one based on simple wavelet basis and one that combines two basis working in parallel. We will test them and show that they can be used at the same level of performance as max and average pooling.

Resum

Les wavelet són funcions matemàtiques que s'utilitzen avui en dia en molts problemes de visió per computador, com per exemple netejar o compressió d'imatges. En aquest treball, primer estudiarem tota la teoria bàsica que ens permet entendre les funcions i construir un coneixement sòlid per poder construir una aplicació nova. Per aquest propòsit, proposem dos mètodes de *pooling* basats en wavelets: un mètode utilitza una sola base de wavelets i l'altre en combina dues que funcionen en paral·lel. Provem i testegem els mètodes per demostrar que es poden utilitzar per aconseguir els mateixos resultats que el *max* i *average pooling*.

Acknowledgements

First of all, I would like to thank my professor Dr. Petia Radeva for this opportunity. I would also like to thank Eduardo Aguilar for your valuable help in the understandings of the many dimensions of TensorFlow and Keras.

I would like to thanks the author of the paper that served as inspiration for this work, Dr. Travis Williams and Dr. Robert Li. Special thanks to Dr. Williams for responding to my e-mails in such a short notice and being so specific about small details.

I would like to thanks Dr. Lluís Garrido for the corrections of this work, among many valuable advises.

Thanks to my two best friends, Jose and Ignasi, for accompanying me on this journey. You guys are the best.

And of course, thanks to my family and Nico. You have always been there for me.

Contents

1 Introduction

Wavelets are mathematical functions that have been heavily studied during the 20th century, used in many applications. The first wavelet development is attributed to Alfréd Haar in 1909. The development of the continuous wavelet transform would come much later (1975) while studying the reaction of the ear to sound; and can be attributed to George Zweig. In 1982, Pierre Goupillaud, Alex Grossmann and Jean Morlet established the formulation of that now is known as the continuous transform, thus "rediscovering" it. They were interested in analyzing data from sesmic surveys, which are used in oil and mineral explotations to get images of layering in subsurface rock [1]. Then would come Jan-Olov Strömberg's work on discrete wavelets (1983), Ingrid Daubechies' orthogonal wavelets with compact support (1988), Stéphane Georges Mallat's multiresolution framework (1989), Ali Akansu's binomial QMF (1990), Nathalie Delprat's time frequency interpretation of the continuous transform (1991), Newland's harmonic wavelet transform (1993) and many more since then.

We can see that the origin of wavelets comes from different real world applications and in the span of 20 years, many researchers have shown interest in them. Such tendency has continued: wavelets are currently used in data and image compression [2], partial differential equation solving [3], transient detection [4], pattern recognition [5, 6], texture analysis [7], noise reduction [8], financial data analysis [9] and many more fields. Perhaps an interesting and surprising example is the FBI algorithm for fingerprint compression [10].

On another front, one of the biggest breakthroughs of recent computer science has been the development of deep learning. Such technique imitates the workings of the human brain in processing data and creating patterns for use in decision making. Deep learning belongs to the field of artificial intelligence. The term "Artificial Intelligence" was first used at a workshop held on the campus of Darmouth College during the summer of 1956. The term "deep learning", however, would come much later, used for the first time by Igor Aizenberg and colleagues around 2000. One may recall the historical events of 1996 when Deep Blue won its first game against the world chess champion Garry Kasparov; or in 2016, when AlphaGo defeated the 9-dan professional Lee Sedol at Go. Those are just some examples of what artificial intelligence, specifically deep learning for the latter, can achieve.

Neural networks, the main tool of deep learning, are a *before-and-after* in the history of computer science. One special kind, convolutional neural networks, are commonly associated with computer vision. Their historical roots can be traced back to the 1980s, when Kunishiko Fukushima proposed a neural network inspired by the feline visual processing system [13] and [14]. Yann LeCun helped establish how we use convolutional neural networks nowadays: as multiple layers of neurons for processing more complex features at deeper layers of the network. Their properties differ from other traditional networks; for example, they use what best represents images: height, width and depth (number of channels). Instead of a set of neurons that accept a single numerical vector of values, convolutional neural networks work with the whole image, adding layers of information in the depth dimension. In summary, convolutional neural networks consistently classify images, objects and videos at a higher accuracy rate than vector-based deep learning techniques.

Pooling methods are designed to compact information, i.e., reduce data dimensions and parameters, thus increasing computational efficiency. Since convolutional neural networks work with the whole image, the number of neurons increases and so does the computational cost. For this reason, some kind of control over the size of our data and parameters is needed. However, this is not the only reason to use pooling methods, as they are also very important to perform a multi-level analysis. This means that rather than the exact pixel where the activation happened, we look for the region where it is located. Pooling methods vary from deterministic simple ones, such as max pooling, to probabilistic more sophisticated ones, like stochastic pooling. All of these methods have in common that they use a neighborhood approach that, although fast, introduce edge halos, blurring and aliasing.

Max pooling is a basic technique that usually works, but perhaps too simple and it tends to overfit. On the other hand, average pooling is more resistant to overfitting, but it can create blurring effects to certain datasets. Choosing the right pooling method is key to obtain good results. The main goal of this work is to prove that wavelets can be used as a valid pooling method. We claim that wavelets can be a good choice for the majority of datasets and that they can be a safe choice when in doubt. In order to do that, we will study the basic theory of wavelets as mathematical abstract functions and we will prepare and test a wavelet pooling layer against three different datasets. On the other hand, we will state the basic concepts of convolutional neural networks and we will study the common pooling techniques. Then, we explain the pooling algorithm taken from the works of Dr. Travis Williams and Dr. Robert Li [20] and test it in different contexts. Then, we will explore a how different choice of wavelet basis affect the performance of the network. Finally, we propose a new pooling method: *multiwavelet* pooling. Since wavelets work similarly to filters, we will prove that combining multiple wavelets basis in the same layer contributes to achieve a higher accuracy.

This work is structured as follows: Section 2 contains the state of the art; Section 3 contains methodology, including basic wavelet theory and basic convolutional neural network knowledge; Section 4 formalizes our proposed pooling methods; Section 5 contains the validation and results; and Section 6 contains the conclusions and future work.

2 State of the art

In this work, we will be focusing mainly on the effect and performance of different pooling techniques. The purpose of pooling layers is to achieve spatial invariance by reducing the resolution of the feature maps. Here, we will be presenting the most used pooling techniques.

Perhaps the most known and used pooling method is max pooling: it reduces data size by taking the highest value among a fixed region. Max pooling is a simple and computationally efficient technique, since it requires only comparisons and no further computation; often used by default when in doubt. However, due to its simplicity, it tends to overfit the data [24].

The second most used method is average pooling: it computes the average of all activations in a fixed region. This method is also computationally efficient but, by taking a neighborhood of values, it can produce halos and blurring [26].

To combat the previously mentioned problems, researchers focused on probabilistic methods, i.e., introducing randomness that hopefully helped performance. For that, mixed pooling was born: with a set probability λ max pooling will be performed and with probability $1 - \lambda$, average pooling will be. There is no set way to perform this method, as this can be applied for all features within a layer, mixed between features within a layer or mixed between regions for different features within a layer [25]. Additionally, gate pooling takes this idea and generalizes it a bit further: the probability based on which we choose between average and max pooling will be a trainable parameter [25]. One step further brings us to what is referred to as tree pooling, in which pooling filters are learnt and also how to responsively combine those learned filters [25].

Continuing with the same line of thought, stochastic pooling was introduced: assign a probability to each pixel corresponding to its activation value; the higher the value, the higher the probability to be chosen. This method works in the same fashion as max pooling, but introducing some randomness that can help improve accuracy [26].

Wavelet pooling is designed to resize the image without almost losing information [20]. With this property, it could be a safe choice when one is doubtful between max pooling and average pooling: wavelet pooling will not create any halos and, because of its structure, it seem it could resist better overfitting. We propose to generalize a bit further this technique by combining several wavelet pooling layers that will act like a pooling filter.

3 Methodology

In this section, we will explore mainly two subjects: wavelet theory and convolutional neural networks. For the first one, we will introduce all its basic concepts, giving formal definitions, building up to the matrix that we will use in the proposed algorithm. For the latter, we will review some concepts of *deep learning* and the what and why for this kind of network.

3.1 Fourier analysis

One may recall that in 1807, Joseph Fourier asserted that any 2π periodic function $f(x)$ could be written as

$$f(x) = a_0 + \sum_{k=1}^{\infty}(a_k \cos(kx) + b_k \sin(kx))$$

now known as *Fourier series.* The coefficients a_0, a_k, b_k can be calculated by

$$a_0 = \frac{1}{2\pi}\int_0^{2\pi} f(x)dx \qquad a_k = \frac{1}{\pi}\int_0^{2\pi} f(x)\cos(kx)dx \qquad b_k = \frac{1}{\pi}\int_0^{2\pi} f(x)\sin(kx)dx.$$

Later on, that would give rise to a new mathematical field: **frequency analysis**. Let us assume that we have a smooth signal $f(t)$. The standard Fourier transform

$$Ff(w) = \frac{1}{\sqrt{(2\pi)}}\int e^{-iwt} f(t)dt$$

gives a representation of the frequency content of f. In many applications, one is interested in its frequency content *locally* in time. But the Fourier transform fails to encode high frequency bursts. Time localization can be achieved by windowing the signal f, so as to cut off only a well localized slice of f and then taking its Fourier transform

$$(Ff)_g(w,t) = \int f(s)g(s-t)e^{-iws}ds.$$

This is called *the windowed Fourier transform*, where g represents our window. It is common in signal analysis to work with its discrete frequency version, where t and w are assigned regularly spaced values $t = nt_0$, $w = mw_0$, $n, m \in \mathbb{Z}$ and $w_0, t_0 > 0$ are fixed. Then we have

$$(Ff)_g(m,n) = \int f(s)g(s-nt_0)e^{-imw_0 s}ds.$$

Finally, to work with each frequency equally, the width of the window should be allowed to vary inversely with the magnitude of the frequency. And so arranging the formula, the first wavelet transform appears

$$(Ff)_g(w,t) = \frac{1}{\sqrt{\alpha}}\int_{-\infty}^{\infty} f(s)g\left(\frac{x-t}{\alpha}\right)e^{-2\pi iw\frac{x-t}{\alpha}}dx$$

where α is a weighting coefficient: as it increases, the frequency decreases and the window function expands. This arrangement can also be done with its discrete counterpart.

3.2 Wavelets

It is common in literature to assume that one already knows what a wavelet is and so a precise definition is hard to find. Furthermore, there are some mixed concepts in wavelet analysis, such as the words *wavelet*, *mother wavelet*, *father wavelet* and *wavelet family*. We will present a proper definition for each of those concepts, but in time, the word *wavelet* will take the lead.

Definition 3.1. A *mother wavelet* is a function $\psi : \mathbb{C} \to \mathbb{C}$ satisfying the following conditions

1. $\displaystyle\int_{-\infty}^{\infty} |\psi(t)|^2 dt < \infty$
2. $\displaystyle\int_{-\infty}^{\infty} \frac{|\Psi(w)|^2}{|w|} dw < \infty$

where Ψ is the Fourier transform of ψ.

The first thing we should notice is that the definition states that ψ is defined over $\mathbb{C}$, meaning that we should be careful performing some operations. This will not bother us later on, since our object of study (images) can be viewed as a function over $\mathbb{R}$ and everything will work just fine.

Now, about the interesting mathematical concept. The first condition implies that the function has finite energy, an important concept in *signal processing*. The second one is called *admissibility condition* and implies that if Ψ is smooth then $\Psi(0) = 0$. It can be shown that square integrable functions satisfying the admissibility condition can be used to first analyze and then reconstruct a signal without loss of information. That statement gives us a hint why researchers have decided to use wavelets as a tool in image processing: they allow us to **pool the image**, then **analyze it** and **reconstruct the same exact image**.

Further on, since $\Psi(0) = 0$, that implies that the average value of the wavelet must be zero

$$\int_{-\infty}^{\infty} \psi(t) dt = 0$$

and therefore it must be oscillatory, that is, $\psi(t)$ must be a *wave*.

Roughly speaking, a mother wavelet is a wave-like oscillation with an amplitude that begins at zero, increases and then decreases back to zero. The main difference between cosines, sines and mother wavelets is that while the former are extended to the whole time domain, the latter are only “alive” for a short period of time. Let us look at some examples of how mother wavelets look like.

Now, we present what does it mean to form a family of wavelets.

Definition 3.2. Consider ψ a mother wavelet, a *family of wavelets* is a basis defined by

$$\psi_{(s,l)}(x) = \frac{1}{\sqrt{s}} \psi\left(\frac{x-l}{s}\right)$$

where s is an integer called *scale index* and l is an integer called *location index.*

It is clear now why the phrasing becomes lazy and ends up being *wavelet* for everything. A family of wavelets is just a set of functions generated by translating and scaling the mother wavelet. So, essentially, a family of wavelets is just a bunch of modification of the same mother wavelet.

3.3 Continuous wavelet transform

It is obvious that we will not work with continuous data nor functions, but studying this case is fundamental to understand its discrete counterpart and the problems that come with it. The wavelet transform provides a time-frequency description, similar to the one in the Fourier transform. The location and scale parameters s, l vary continuously over $\mathbb{R}$. The *continuous wavelet transform* (CWT) of a smooth function $f(t)$ is given by

$$CWT_{(s,l)}f = s^{-1/2} \int f(t)\psi\left(\frac{t-l}{s}\right) dt.$$

The resemblance with the Fourier transform is striking, we are just changing the function $e^{\cdot}$ with a wavelet $\psi(\cdot)$. From Euler's formula, $e^{iy} = \cos y + i \sin y$, we are just changing sines and cosines for another type of wave.

Another important thing is that a function can be reconstructed from its wavelet transform by means of

$$f(t) = \frac{1}{C_\Psi} \int_0^\infty \int_{-\infty}^\infty CWT_{(s,l)}f\psi_{(s,l)}(t)\frac{dsdl}{l^2}$$

where c_Ψ comes from the admissibility condition:

$$C_\Psi = \int_0^\infty \frac{|\Psi(w)|^2}{w} dw < \infty.$$

The inverse formula can be viewed in two different ways: as a way of reconstructing f once its wavelet transform is known, or as a way to write f as a superposition of wavelets $\psi_{(s,l)}$; the coefficients in this superposition are exactly given by the wavelet transform.

We could stop studying the continuous transform here, since we have everything we need to understand the further process. However, there are some properties that are worth studying since they give a better vision about wavelets and help with our understanding. Let us talk about moments.

An important concept for wavelets is the notion of *vanishing moments*. Wavelets decay over time and, eventually, they die out. But how they do that and how smooth they are is coded with the vanishing moments.

Definition 3.3. The p-th *vanishing moment* of a wavelet $\psi_{(s,l)}(t)$ is defined by

$$M_p = \int_{-\infty}^\infty t^p \psi_{(s,l)}(t)dt.$$

Vanishing moments can be defined for any function and measure the "smoothness" of such: the larger they are, the smoother the function. If we expand the Taylor series of our CWT at $t = 0$, taking $l = 0$ as well for simplicity:

$$CWT_{(s,0)}f = \sum_{p=0}^{n} f^{(p)}(0) \int \frac{t^p}{p!}\psi_{(s,0)}(t)dt + O(n+1)$$

now, rewriting the expression with vanishing moments:

$$CWT_{(s,0)}f = f(0)M_0 s + f^{(1)}(0)M_1 s^2 + \frac{f^{(2)}(0)}{2!}M_2 s^3 + \ldots + \frac{f^{(n)}(0)}{n!}M_n s^{n+1} + O(s^{n+2}).$$

From the admissibility condition, we know that $M_0 = 0$. If the other moments are zero as well, then the wavelet coefficients $\psi_{(s,l)}(t)$ will decay as fast as s^{n+2} for a smooth signal $f(t)$. If a wavelet has N vanishing moments, so will the continuous wavelet transform. Vanishing moments will be further explained in section 3.9.

3.4 Discrete wavelet transform

It is time to move on to the core part of our study. In this case, the location and scale parameters l, s take only discrete values. For s we choose the integer powers of one fixed scale parameter $s_0 > 1$, that is, $s = as_0^m$ and $a \in \mathbb{R}$, $m \in \mathbb{Z}$. It follows that the discretization of the location parameter l should depend on m: narrow (high frequency) wavelets are translated by small steps in order to cover the whole time range, while wider (lower frequency) wavelets are translated by larger steps. We choose then $l = nl_0 as_0^m$, where $l_0 > 0$ is fixed and $n \in \mathbb{Z}$. The formula looks like

$$DWT^{m,n}_{(s_0,l_0)}f = s_0^{-m/2}\int f(t)\psi(s_0^{-m}t - nl_0)dt.$$

Since this is the counterpart of a continuous algorithm, some questions arise: is it possible to characterize f completely knowing its discrete wavelet transform? Is it possible to reconstruct f in a numerically stable way? Considering the dual problem that we proposed in the continuous form, one may ask the following questions: is it possible to write f as a superposition of $\psi_{(s,l)}$? Is there a numerically stable algorithm to find such coefficients?

In the discrete case, there does not exist, in general, a formula to recover f from CWT. Reconstruction, if at all possible, must therefore be done by some other means. Besides, discrete frames often offer redundant information. This redundancy can be exploited or eliminated to its bare essentials. This takes us to the next concept.

3.5 Multiresolution analysis

First of all, we will start with a general definition.

Definition 3.4. Let V_j, $j = \ldots, -2, -1, 0, 1, 2, \ldots$ be a sequence of subspaces of functions in $L^2(\mathbb{R})$. The collection of spaces $\{V_j, j \in \mathbb{Z}\}$ is called a *multiresolution analysis with scaling function* ϕ if the following conditions hold:

1. Nested: $V_j \subset V_{j+1}$
2. Density: $\overline{\bigcup V_j} = L^2(\mathbb{R})$, where $\overline{X}$ denotes the topological closure of a space X.
3. Separation: $\bigcap V_j = \{0\}$
4. Scaling: $f(x) \in V_0 \iff f(2^j x) \in V_0$
5. Orthonormal basis: $\phi \in V_0$ and the set $\{\phi(x-k), k \in \mathbb{Z}\}$ is an orthonormal basis (using the L^2 inner product) for V_0.

Let us stop for a moment. We have introduced some new notation without explaining it. What is a L^2 space?

Definition 3.5. The L^2 space is the space of square-integrable functions, i.e., those functions that

$$\int_{-\infty}^{\infty} |f(x)|^2 dx < \infty.$$

Alternatively, it can also be defined as the set of functions which square is *Lebesgue integrable.* The inner product of the space is given by

$$\langle f, g \rangle = \int_A f(x) g(x) dx.$$

This is a simplified definition, but it serves our purpose. In our field of research, all functions are square integrable, since we are working with finite set of values (images), so that is purely notation.

The V_j spaces are called *approximation spaces.* There may be several choices of ϕ corresponding to a system of approximation spaces. Different choices of ϕ may yield different multiresolution analyses. So it is not a one to one relationship.

Remark 3.6. Although the definition requires the translates of ϕ to be orthonormal, a simple basis of ϕ will suffice. We can then use ϕ to obtain a new scaling function for which $\{\phi(x-k), k \in \mathbb{Z}\}$ is orthonormal. This is a common concept for lineal algebra and it will have consequences further on.

This definition is often presented together with the central equation in multiresolution analysis, the scaling relation.

Theorem 3.7. *Suppose $\{V_j, j \in \mathbb{Z}\}$ is a multiresolution analysis with scaling function ϕ. Then, the following scaling relation holds:*

$$\phi(x) = \sum_{k \in \mathbb{Z}} h_k \phi(2x-k) \quad \textit{where} \quad h_k = 2\int_{-\infty}^{\infty} \phi(x)\phi(2x-k)dx. \tag{3.1}$$

Since this is an introductory work and proofs require extensive mathematical knowledge on the matter, we will not state them here. However, if one is interested, the proof can be found on [1]. Now, at this point, the reader may be wondering: what does any of this have to do with wavelets? Let us check out the next revealing theorem.

Theorem 3.8. *Suppose $\{V_j, j \in \mathbb{Z}\}$ is a multiresolution analysis with scaling function ϕ. Let W_j be the span of $\{\psi(2^j x - k) : k \in \mathbb{Z}\}$ where*

$$\psi(x) = \sum_{k \in \mathbb{Z}} g_k \phi(2x-k) \quad \textit{where} \quad g_k = (-1)^k g_{1-k}. \tag{3.2}$$

Then $W_j \in V_{j+1}$ is the orthogonal complement of V_j in V_{j+1}, i. e., $V_j \oplus W_j = V_{j+1}$. Furthermore, $\{\psi_{j,k}(x) := 2^{j/2}\psi(2^j x - k), k \in \mathbb{Z}\}$ is an orthonormal basis for W_j.

This may be a lot of information to process, but what is really important is that now, our space looks like that, fixed a $j_0 \in \mathbb{Z}$:

$$L^2(\mathbb{R}) = V_{j_0} \oplus W_{j_0} \oplus W_{j_0+1} \oplus W_{j_0+2} \oplus \dots$$

which means that ϕ and ψ both together are a basis of our space. Moreover, the definition of ψ given by the theorem is the definition of a family of wavelets, just changing $j = -m$ and everything checks out. This is a very strong property: we have a basis formed by our scaling function and our wavelet. This is the reason why, sometimes, the scaling function is also called *father wavelet.*

3.6 Wavelet matrix

In our case study, we are dealing with images; they have a stronger property than just being discrete: they are finite. This simplifies a lot of our work. For this purpose, let us consider $\mathbb{Z}_N$, $N \in \mathbb{Z}$, the space of finite successions (which means that we have a series of values $a_0, a_1, a_2, \dots, a_{N-1}, a_N = a_0, a_{N+1} = a_1, \dots$). This property is also written as $(R_N a)(m) = a_{m-N}$. In this scenario, data can be thought of vectors of N-length with the described property. Then, we can consider the space $l^2(\mathbb{Z}_N)$ which can be thought of the space of finite signals. This is where our wavelets lie now.

For example, consider a signal $z \in L^2(\mathbb{Z}_N)$, which is represented by a vector of values. Then, we can perform the Fourier transform by simply doing

$$\hat{z} = \sum_{n=0}^{N-1} z_n e^{-2\pi imn/N}.$$

Our first concern is to know whether we can find an orthonormal basis or not and, in case that we could, how to do it. Recall that, from theorem 3.8, our wavelets form an orthonormal basis in the discrete case, so it is only natural to think that this property would be preserved.

Lemma 3.9. *Let $w \in L^2(\mathbb{Z}_N)$. Then $\{R_k w\}_{k=0}^{N-1}$ is an orthonormal basis for $L^2(\mathbb{Z}_N)$ if and only if the discrete finite Fourier transform $\hat{w}$ follows that $|\hat{w}(n)| = 1$ for all $n \in \mathbb{Z}_N$.*

We can find a proof of this in [15]. Now we have a way to find an orthonormal basis in terms of the Fourier transform, but we would like something that requires less calculus. For this purpose, let us define now our wavelet basis.

Definition 3.10. Suppose N is an even integer, $N = 2M$ for some $M \in \mathbb{N}$. An orthonormal basis for $L^2(\mathbb{Z}_N)$ of the form

$$\{R_{2k}u\}_{k=0}^{M-1} \cup \{R_{2k}v\}_{k=0}^{M-1}$$

for some $u, v \in l^2(\mathbb{Z}_N)$ is called a *first stage wavelet basis* for $l^2(\mathbb{Z}_N)$. We call u and v the *generators* of the first stage wavelet basis. We sometimes also call u the *father* wavelet and v the *mother* wavelet.

This is basically saying that we have to find two vectors u, v that considering a system of vectors

$$\begin{array}{cccc}
(u_0 & u_1 & \dots & u_{N-1}) \\
(u_{N-2} & u_{N-1} & \dots & u_{N-3}) \\
\vdots & \vdots & \dots & \vdots \\
(u_3 & u_4 & \dots & u_2) \\
\hline
(v_0 & v_1 & \dots & v_{N-1}) \\
(v_{N-2} & v_{N-1} & \dots & v_{N-3}) \\
\vdots & \vdots & \dots & \vdots \\
(v_3 & v_4 & \dots & v_2)
\end{array}$$

such vectors are an orthonormal basis for $L^2(\mathbb{Z}_N)$. Considering that we have to apply a Fourier transform to prove that they are an orthonormal basis, we expect something simpler from the finite case, which is given by the following theorem.

Theorem 3.11. *Suppose $M \in \mathbb{N}$ and $N = 2M$. Let $u, v \in L^2(\mathbb{Z}_N)$. Then*

$$B = \{R_{2k}v\}_{k=0}^{M-1} \cup \{R_{2k}u\}_{k=0}^{M-1}$$

is an orthonormal basis for $L^2(\mathbb{Z}_N)$ if and only if the system matrix $W(n)$ of u and v is unitary for $n = 0, \dots, M-1$.

The proof of this theorem can also be found on [15]. So now we have completely defined our matrix wavelet in terms of an orthonormal basis and a unitary matrix. This relates to the discrete case, because such vectors can be found following the equations 3.1 and 3.2. Therefore, our wavelet matrix can be expressed as

$$W = \left(\begin{array}{ccccc}
h_0 & h_1 & h_2 & \dots & h_{N-1} \\
h_{N-2} & h_{N-1} & h_1 & \dots & h_{N-3} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\hline
g_0 & g_1 & g_2 & \cdots & g_{N-1} \\
g_{N-2} & g_{N-1} & g_1 & \cdots & g_{N-3} \\
\vdots & \vdots & \vdots & \ddots & \vdots
\end{array}\right).$$

This theory has been introduced to treat with one dimensional array data, but can be easily extended to two dimensional data, which is our case of interest and our next goal.

3.7 2D transform explained

Until this point, we have been working with $1D$ vector functions. The main point to focus on is that, in order to build a wavelet matrix, we need to find two vectors that form an orthonormal basis. What may seem problematic is that with the wavelet transform we go from $1D$ to $2D$. What happens with images, then? Do we need to work with $4D$ matrices? The answer is no. Let us look at the process.

First of all, some notation. Since our matrix is completely characterized with differentiated elements h_i and g_i from different subspaces, let us write

$$W = \left(\begin{array}{ccccc} h_0 & h_1 & h_2 & \dots & h_{N-1} \\ h_{N-2} & h_{N-1} & h_1 & \dots & h_{N-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline g_0 & g_1 & g_2 & \dots & g_{N-1} \\ g_{N-2} & g_{N-1} & g_1 & \dots & g_{N-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \end{array}\right) = \begin{pmatrix} H \\ \hline G \end{pmatrix}.$$

Let us assume that A is an $n \times n$ matrix to transform, say an image, for example. How do we transform A? If we compute WA we are simply applying the transform to each column. Now, for an image, this would look like that which does not seem so bad. But it is the result of transforming only the columns and building together a new image. It does not go well with the theory that we have studied.

What should we do to process the rows of A as well? We just need to compute WAW^T. The reader may take a minute to check that the result is a square matrix. Even if W was not square, we would have $(m \times n) \cdot (n \times n) \cdot (n \times m) = m \times m$, which is mathematically relevant because the result of the process is always a square matrix.

This process, when looking at it in block format, is revealing

$$WAW^T = \begin{pmatrix} H \\ \hline G \end{pmatrix} A \begin{pmatrix} H \\ \hline G \end{pmatrix}^T = \begin{pmatrix} HA \\ \hline GA \end{pmatrix} \left(H^T \mid G^T\right) = \left(\begin{array}{c|c} HAH^T & HAG^T \\ \hline GAH^T & GAG^T \end{array}\right) = \left(\begin{array}{c|c} LL & HL \\ \hline LH & HH \end{array}\right).$$

where the last matrix contains its usual notation. The original image A is transformed into 4 matrices called *subbands*. The matrices H and G are designed so that the LL subband is the low resolution residual consists of low frequency components, which means that it is an approximation of our original image; and thee subbands HL, LH and HH give horizontal, vertical and diagonal details, respectively.

3.8 Example: the Haar wavelet

The Haar wavelet is defined by its mother wavelet function ψ and its scaling function ϕ:

$$\psi(t) = \begin{cases} 1 & 0 \le t < 1/2, \\ -1 & 1/2 \le t < 1, \\ 0 & \text{otherwise} \end{cases}$$

$$\phi(t) = \begin{cases} 1 & 0 \le t < 1, \\ 0 & \text{otherwise} \end{cases}$$

Let us take a look at the graphs of those functions, see Figure 1.

Now we can compute the values h_i and g_i mentioned before:

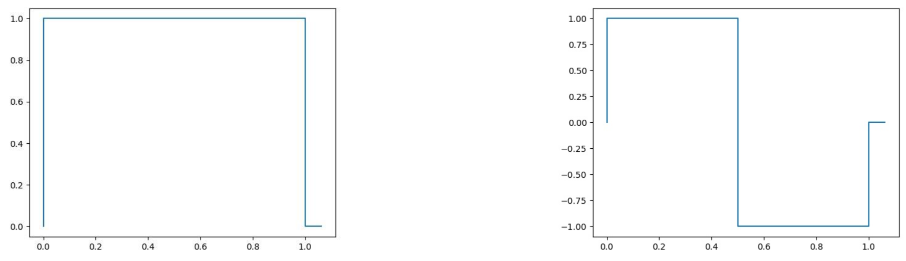

(a) Haar scaling function (b) Haar mother wavelet

Figure 1: Haar filter is defined by the scaling function and the mother wavelet.

- $h_0 = 2\int_{-\infty}^{\infty} \phi(t)\phi(2t)dt = 2\int_0^1 \phi(t)\phi(2t)dt = 2\int_0^{1/2} \phi(t)\phi(2t)dt = \int_0^{1/2} 1dt = 2t|_0^{1/2} = 1.$
- $h_1 = 2\int_{-\infty}^{\infty} \phi(t)\phi(2t-1)dt = 2\int_0^1 \phi(t)\phi(2t-1)dt = 2\int_{1/2}^1 \phi(t)\phi(2t-1)dt = \int_{1/2}^1 1dt = 2t|_{1/2}^1 = 1.$
- $h_2 = 2\int_{-\infty}^{\infty} \phi(t)\phi(2t-2)dt = 2\int_0^1 \phi(t)\phi(2t-2)dt = 2\int_0^1 1\cdot 0dt = 0.$
- It is easy to see that $h_k = 0, \forall k > 1$.

And for the other values it follows that:

1. $g_0 = (-1)^0 h_{1-0} = 1\cdot 1 = 1.$
2. $g_1 = (-1)^1 h_{1-1} = -1h_0 = -1.$
3. And $g_k = 0, \forall k > 1$.

Therefore, the final matrix is

$$H = \begin{pmatrix} 1 & 1 \end{pmatrix} \quad G = \begin{pmatrix} 1 & -1 \end{pmatrix} \quad W = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}.$$

But remember remark 3.6, what happens with this matrix? It is not orthonormal as we expected, since:

$$\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}^T = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \neq \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

That is ok, because, as said in the remark, you can always transform the basis into an orthonormal basis. In this case, you just have to multiply by $\sqrt{2}/2$ and then:

$$\begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{pmatrix} \cdot \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{pmatrix}^T = \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{pmatrix} \cdot \begin{pmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

Now, for a moment, let us look in a very practical way what this matrix is doing. Let us assume, then, that we have an $n \times n$ matrix A.

- When multiplying the H block, in every step we are computing $\sqrt{2}(\frac{x_i+x_{i+1}}{2})$. Basically, we are averaging the columns of A.
- When multiplying the G block, in every step we are computing $\sqrt{2}(\frac{x_i-x_{i+1}}{2})$. So, in this case, we are computing the differences for every column.

Remark 3.12. Notice that $\sqrt{2}(\frac{x_i+x_{i+1}}{2}) + \sqrt{2}(\frac{x_i-x_{i+1}}{2}) = \sqrt{2}x_i$, which is the reason that explains that we can invert the process. In some books and talks, one may be introduced to the Haar filter as the "averaging" filter. And all the theory may be developed focusing on this interesting property. Although we have decided to present the abstract theory here, one may find some references about it in this wavelet workshop [16].

So, for our block process, we had that:

$$WAW^T = \begin{pmatrix} H \\ \hline G \end{pmatrix} A \begin{pmatrix} H \\ \hline G \end{pmatrix}^T = \begin{pmatrix} HA \\ \hline GA \end{pmatrix} \left(H^T \mid G^T\right) = \left(\begin{array}{c|c} HAH^T & HAG^T \\ \hline GAH^T & GAG^T \end{array}\right).$$

Now, we can explicitly explain every subband:

- HAH^T averages along the columns of A and then along the rows of HA. It is clear that this will produce an approximation (blur) of A, losing the details due to the average factor.
- HAG^T averages along the columns of A and then differences along the rows of HA. This will produce vertical differences between A and the blur of A.
- GAH^T differences along the columns of A and then averages along the rows of GA. In this case, this will produce the horizontal differences between A and the blur of A.
- And finally, GAG^T differences along columns and arrows. This will produce the diagonal differences between A and its blur.

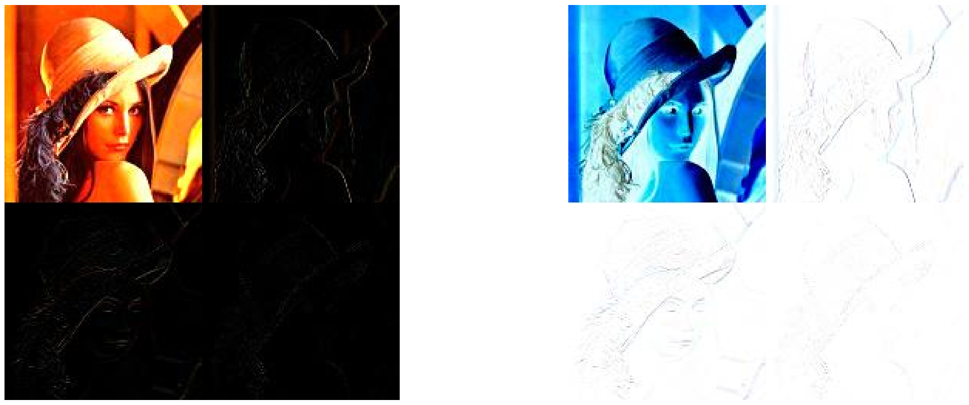
This simple example clarifies all the strong theory that is involved in the discrete wavelet transform. In the next section, we will explore further, with some visual examples, the effect of various wavelets on our field of research: images.

3.9 Visualization and effects of some wavelets

We will now prepare a bank of examples of wavelets and its different effects to images. If the reader is interested in using wavelets as a programming tool, we suggest using `MatLab`, which has them integrated, or `Python` [17], which has multiple libraries for it. In order to compare the effects of different wavelets, we will be using the same image as example: Lena, see Figure 2.

Figure 2: “Hello world” computer vision version.

In Figure 1, we have already seen the shape of the Haar functions. Let us check the impact on an image, which is what we are interested in.



(a) Haar result.

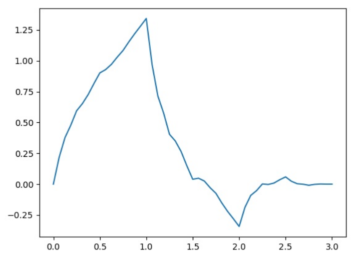
(b) Inverted Haar result.

Figure 3: The inverted result is given just for better appreciation of details.

Another known wavelet, with extended use, is the Daubechie wavelet. Although Alfreéd Haar was first in describing the Haar wavelets, this family of wavelets is a generalization of his definition. They are characterized by a maximal number of vanishing moments given some support, i.e., given some subset of the domain where all elements are not mapped to zero. There are lots of papers and articles describing and analyzing it. In [1], we can find a whole chapter of properties and in [18], we can read the discoverer herself. Again, in the workshop [16], there is a different approach to this wavelet. First, we will present the marriage of functions that gives life to this wavelet.

Remark 3.13. When reading more about wavelets, one will classically see this wavelet labeled as “Daubechie D4”. This number is referred to its *vanishing moments*, a concept explained in definition 3.3. Vanishing moments affect smoothness and wave length, therefore, they produce different effects. In our experiments, we have used Daubechie D4, which will now be presented. Another interesting fact is that Haar wavelets are also known as Daubechie D1 wavelets, thus stating the fact that Daubechie generalizes the Haar wavelets.

Let us check the effect of this vanishing moments in a more visual way.


(a) Daubechie D4 scaling function

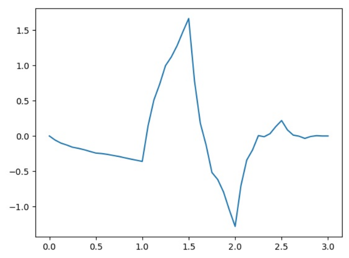
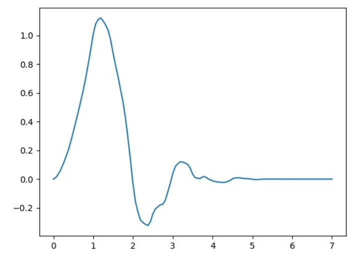

(b) Daubechie D4 mother wavelet

Figure 4: Classic Daubechie filter presented everywhere.


(a) Daubechie D6 scaling function

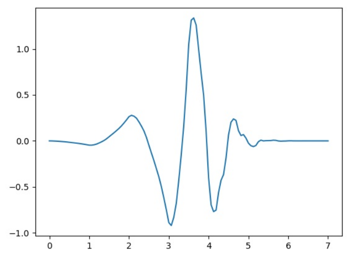
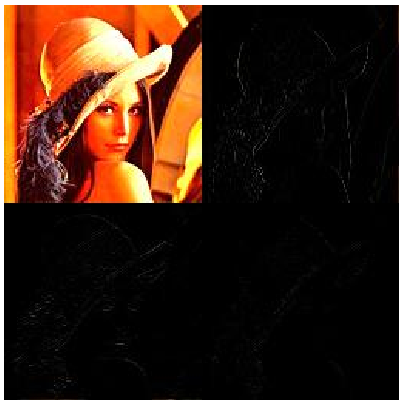

(b) Daubechie D6 mother wavelet

Figure 5: Daubechie D6 is given by these functions.

Figure 4 illustrates the common Daubechie D4 filter while Figure 5 illustrates the Daubechie D6 wavelet. Notice how, in the second image, the function appears to be much smoother. Also, notice how in the D6 wavelet, the domain for both the scaling function and the mother wavelet is larger than in the case of the D4. We will show the effect of the Daubechie D4 filter applied to Lena on Figure 8.


(a) Daubechie D4 transform result.

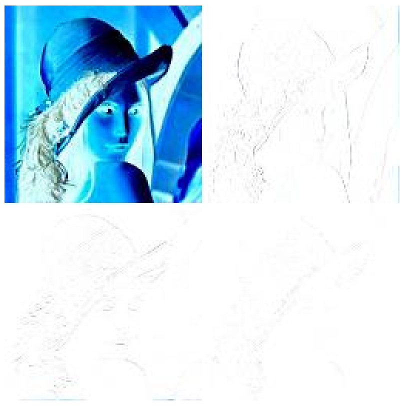
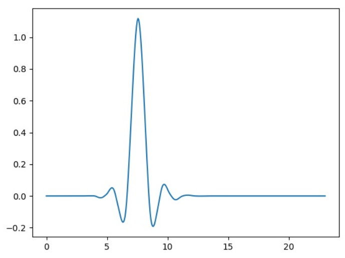

(b) Inverted Daubechie D4 transform result.

Figure 6: Applied Daubechie D4 transform result.

Our last example are Coiflet wavelets, also designed by Ingrid Daubechie upon asked for a wavelet whose scaling function also had vanishing moments. These wavelets have been used in many applications using Calderón-Zygmund operators, in a special field of harmonic analysis. More on this can be found on [18] and [19].


(a) Coiflet D3 scaling function

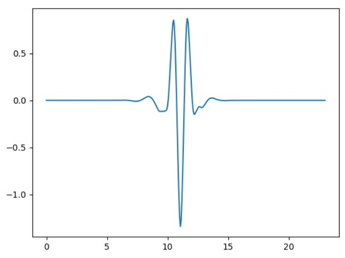
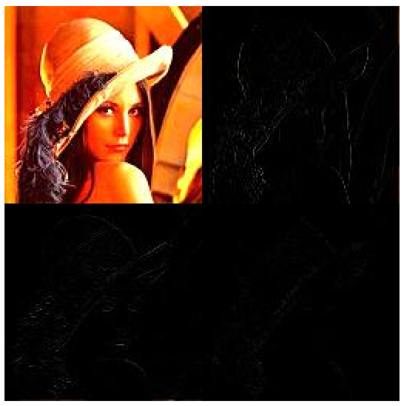

(b) Coiflet D3 mother wavelet

Figure 7: Coiflet D3 filter is defined by these two functions.

The effect on an image can be seen on the following figure.


(a) Coiflet D3 transform result.

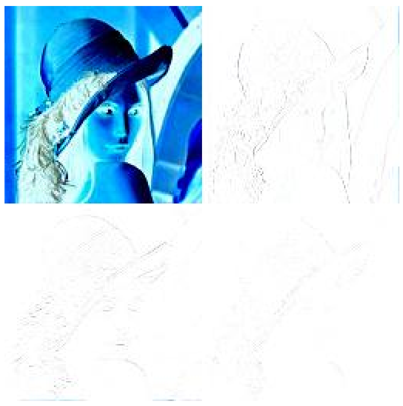

(b) Inverted Coiflet D3 transform result.

Figure 8: Applied Coiflet D3 transform result.

3.10 Introduction and basic concepts of neural networks

For now, the abstract mathematical theory is completely defined, so it is time to change subject to the computer science one. We will assume a basic knowledge of deep learning, but first, we will review briefly the main concepts.

Definition 3.14. *Machine learning* is an application of artificial intelligence that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.

Definition 3.15. *Deep Learning* is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks.

This takes to the core of our the problem.

Definition 3.16. A *neural network* is a system that accepts an input signal which is processed by a set of interconnected components – called neurons – to produce an output.

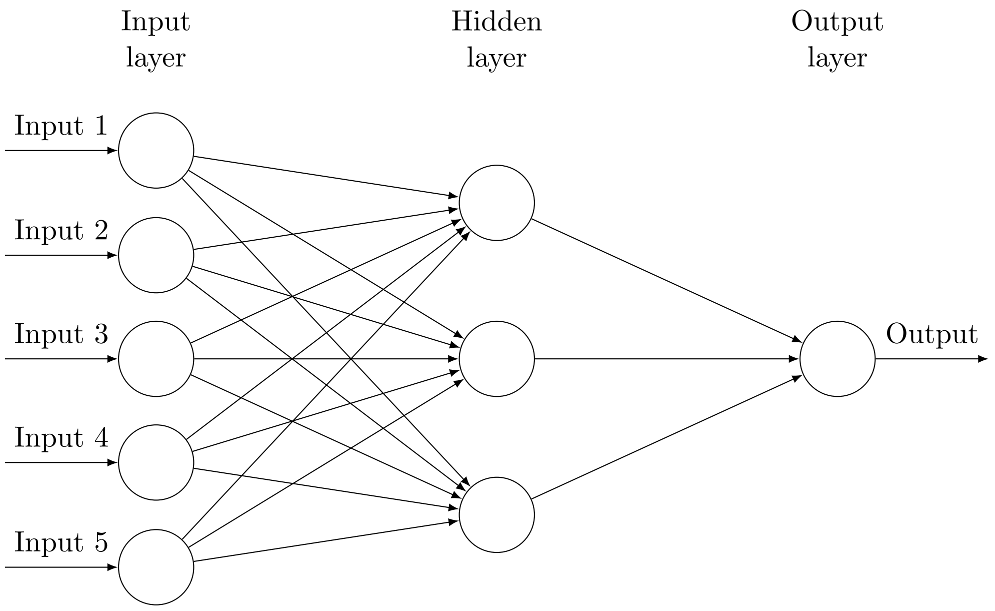


Figure 9: High-level picture of how neural networks are

Those definitions establish the base on which we will build our so called convolutional neural networks. There are plenty of papers working on this matter, a perfect introductory one could be the Standford class [21]. Figure 9 shows, in general terms, how a neural network works. The computations, even though they can be tough and obscure, come from a very easy concept: a mix of linear algebra and an optimization problem.

3.11 Convolutional neural networks

Convolutional neural networks are very similar to regular neural networks: they have an input layer, an output layer, and are made up of neurons that have learnable weights and biases. Then, why change? Neural networks have proved themselves very useful; why should we use anything different? The answer, similar to many programming problems, is because of computational cost. Convolutional neural networks are not the solution of all problems: they just happen to outperform other network architectures in artificial vision problems.

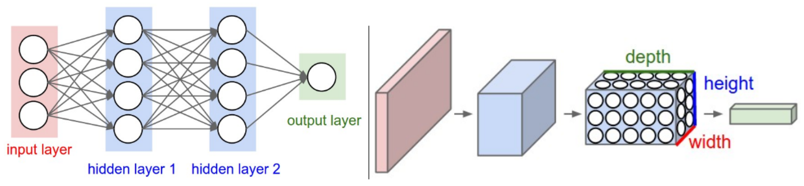


Figure 10: Comparison between neural networks architectures.

Left: common neural network. Right: convolutional neural network.

Take the MNIST dataset, its images are of size $28 \times 28 \times 1$. A single fully-connected neuron in a first hidden layer of a regular neural network would have $28 \cdot 28 \cdot 1 = 784$ weights. This does not look so bad. But MNIST is oversimplified. Tale an image of CIFAR-10, which are of size $32 \times 32 \times 3$. Then the neuron would have $32 \cdot 32 \cdot 3 = 3072$ weights. This may still seem manageable, but an image of size $200 \times 200 \times 3$ would require 120000 weights. Combining it with the other neurons produces an absurdly large amount of weights. Besides, the larger the amount of parameters, the easier it is for the neural network to overfit.

So, the key in all of this is that neurons in a convolutional neural network are arranged in 3 dimensions: **width**, **height** and **depth**. At the end of the network, the full image will be transformed into a single vector of class scores.

A typical example of a general convolutional neural network architecture would involve the following steps or layers:

- **Image pre-processing**. Convolutional neural networks are specially design to work on an specific kind of image. It is not the same problem to work with grayscale images, grayscale encoded with RGB or full RGB images. Additionally, one may want to resize all images to be the same size or even apply transformations so the objects of the study (say, for example, in a classification problem) are all image centered.
- **Convolutional layer**. The key of success. We will explain extensively this layer later on.
- **Pooling layer**. A simple concept, a layer whose function is to reduce the size of our data and so reduce computational cost. We will study this layer later, as this is the focus of our research.
- **Fully connected layer**. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. While the two previous layers are good finding low-level features, i.e., identifying edges and such, we still lack a method to use this in order to classify the image. Here is where the fully connected layer is useful, it will take this information and based on the response to the activation function, it will be able to classify the image.

3.12 Convolutional layers

The convolutional layer's parameters consist of a set of learnable filters. Each of them takes a small span along width and height and extends through the full depth. During the forward pass, this filter slides over the input, computing a dot product, which produces a $2D$ activation map. Convolutional layers have the property of **local connectivity**: a neuron is only connected to a local region of the input volume. The spatial extent of this connectivity is a hyper-parameter called the **receptive field** (or **filter size**).

There are three hyper-parameters that control the size of the output volume and hence the number of neurons. We need to know how these work together and how to control them. Although there are lots of libraries that encapsulate this part, one needs to compute the size of inputs and outputs in order to join layers and form an architecture. Such hyper-parameters are:

- **Depth**: the number of filters to apply.
- **Stride**: number of pixels with which slide the filter.
- **Size of zero padding**: sometimes it is necessary to pad the input with zeros to avoid either shrinking the input or misscalculate the filter. This happens when the input size does not work well together with the filter size and the stride.

Remark 3.17. If W is the input volume size, F is the filter size, S is the stride and P is the amount of zero-padding, the output volume size (and neurons) can be calculated with $(W-F+2P)/S+1$. This means that there are **restrictions** that must be respected. For example, if we have an input of size 7×7 and a filter of size 5×5, if we want to use 0 amount of padding, then we cannot use a stride of 3 because $(7-5)/3+1=1.66$ which is not an integer. Instead, we should use $P=1$ or $S=2$, and we can play around with this formula to obtain the desired values.

Convolutional layers use something called **parameter sharing** to control the number of parameters. For example, let us take the architecture from [23] as seen in Figure 11 it would accept images of size $227\times227\times3$. On the first convolutional layer, it used neurons with $F=11, S=4$ and $P=0$. This gives a total of $(227-11)/4+1=55$. Together with depth $K=96$, the output volume was $55\times55\times96$. Each neuron was connected to a region of size $11\times11\times3$. In total, $55\cdot55\cdot96$ neurons $\cdot11\cdot11\cdot3$ weights $=105.705.600$ weights.

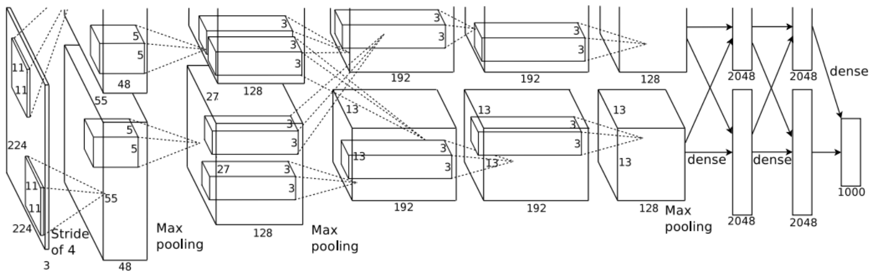


Figure 11: AlexNet neural network structure, image taken from [23].

The number of parameters can be dramatically reduced by making a reasonable assumption: that if one feature is useful to compute at some spatial position (x,y), it will also be useful to compute at a different position (x_2,y_2). This means considering together each different plane (x,y). It makes sense, considering an edge, for example, that the whole edge will be displayed over the same plane; and the same plane will have the same notorious features. Those planes are called **depth slices** and all the neurons in the same depth slice are going to use the same weights and biases. So, with that example, we would now have $96\times11\times11\times3=34848$ weights, which is much more manageable.

Remark 3.18. If all neurons in a single depth slice are using the same weight vector, then the forward pass can be computed as a **convolution** of the neuron's weights with the input volume (hence the name of *convolutional layer*). However, sometimes the parameter sharing assumption does not make sense. For example, if the images have some specific structure that requires the network to learn different features on each side of the image. For example, one may have a dataset composed by image centered faces. In this context, one could expect that different eye-specific or hair-specific features could be learned in different spatial locations. In this case, we talk about **locally connected layers** [27]

3.13 Pooling layers

Pooling layers are designed to progressively reduce the spatial size of the representation, thus reducing the amount of parameters and computation, preventing overfitting. They are also important to implement multi-scale analysis: reducing the information helps focus on the source of the activation, thus identifying the important features. Pooling methods bring tolerance to the identification of features: by working on a neighborhood, they become independent of the exact pixel where the activation happened, focusing on the area instead. It is another name for **subsampling**. Following the paper [20], we will be here reviewing some of the principal pooling methods.

3.13.1 Max pooling

Perhaps the most known pooling method, max pooling function is expressed as:

$$a_{i,j,k} = \max_{(p,q)\in R_{i,j}} (a_{p,q,k})$$

where $R_{i,j}$ is a region of our input volume and $a_{i,j,k}$ is the output activation of the k^{th} feature map at (i, j). So, basically, what we are doing here is define a window of some size and take the maximum value within this window, Figure 12 show this effect. For a max pooling with window size 2 and stride 2, we will halve the size and discard 75% of the activation map.

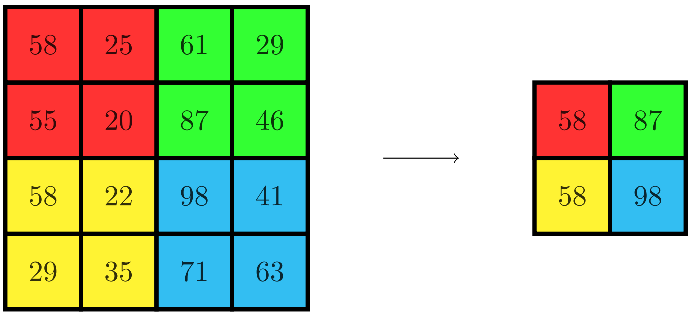


Figure 12: Example of performing max pooling with window size 2 and stride 2.

It is a simple and easy way to perform pooling, although it has its flaws. As explained in [24], if the main features have less intensity than the details, then max pooling will erase important information from an image. In the same paper, we can find some proof that this method commonly overfits training data.

3.13.2 Average pooling

The secondly most known pooling method is the average pooling, which goes by the function:

$$a_{i,j,k} = \frac{1}{|R_{i,j}|} \sum_{(p,q)\in R_{i,j}} a_{p,q,k}$$

using the same notation as described before, additionally $|R_{i,j}|$ is the size of the pooling region. Figure 13 shows the effect of average pooling.

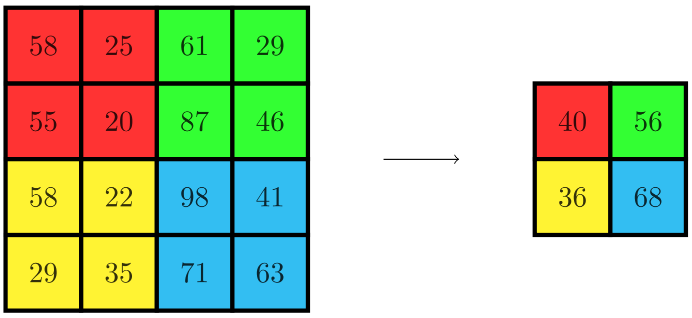


Figure 13: Example of performing average pooling with window size 2 and stride 2.

In the previously cited paper [24], it is also explained that average pooling can dilute details from an image. This happens when the insignificant details have much lower values than our main feature.

We have seen the problems with both famous pooling methods. In order to combat these issues, researchers have created probabilistic pooling methods.

3.13.3 Probabilistic pooling methods

Let us start with **mixed pooling**, which is shown in the equation:

$$a_{i,j,k} = \lambda \cdot \max_{(p,q)\in R_{i,j}} (a_{p,q,k}) + (1-\lambda)\frac{1}{|R_{i,j}|} \sum_{(p,q)\in R_{i,j}} a_{p,q,k}$$

using the same notation as described before, λ is a random value 0 or 1, indicating which pooling method should be used.

There is not an unique way to perform this pooling method. It can be applied in different ways: for all features within a layer, mixed between features within a layer or mixed between regions for different features within a layer. More on this can be found in [24] and [25].

Another probabilistic pooling method is the **stochastic pooling**, which improves max pooling by randomly sampling from neighborhood regions based on the probability values of each activation ([26]). These probabilities p for each region are calculated by normalizing the activations within the region

$$p_{p,q} = \frac{a_{p,q}}{\sum_{(p,q)\in R_{i,j}} a_{p,q}}.$$

The pooled activation is sampled from a multinomial distribution based on p to pick a location l within the region, as seen in the following equation ([26]):

$$a_{i,j,k} = a_l \quad \text{where} \quad l \sim P(p_1, \ldots, p_{|R_{i,j}|}).$$

This process is basically stating that the highest the value, the more probability that it will be the chosen value (thus approximating a max pooling), but any value can be chosen since we are working with probabilities.

Probabilistic pooling avoid the shortcomings of max and average pooling while enjoying some of the advantages of both, but they require further operations and processing.

4 Wavelet pooling

Now, we have reached the main point of this work: to explore wavelet pooling. In the previous sections, we have been studying all of the theory that we needed to argument the use of this pooling method. Max pooling as an effective pooling method, but perhaps too simple; average pooling can produce blurring; and probabilistic methods do not completely solve the two previous presented problems. Wavelets are a strong tool in lots of image applications, such as denoising and compressing. They have interesting properties, such a non-image dependent matrix which means that we can compute it offline and then apply the pooling method without performing further operations. We think that wavelet pooling can become a strong and solid choice as a pooling technique, since it focuses on the inherent image properties to produce a reduced output.

The algorithm used for wavelet pooling is:

1. Compute your favorite wavelet-based wavelet matrix W. In the paper of reference [20], they have used Haar wavelets.
2. Present the feature image F and perform the discrete wavelet transform WFW^T. Remember that this yields a matrix
$$\left(\begin{array}{c|c} LL & HL \\ \hline LH & HH \end{array}\right).$$
3. Discard HL, LH, HH, only taking into account the approximated image LL.
4. Pass on to the next layer and repeat this process when necessary.

Some heavy testing is necessary here and the advantages and disadvantages of this method will be discussed once the experiments are presented in the next sections.

4.1 Multiwavelet pooling

Upon reviewing this looking method, we thought that it was interesting to try a similar idea to convolutions. Since every wavelet basis identifies some detail in particular and perform differently, it was an interesting idea to see how we could combine different basis in the same pooling. In a sense, we were creating a "bank" of pooling techniques in the same way that we create banks of filters with different convolutions. This could help our network to identify features based on the different effects that wavelets have on them; and, at the same time, still performing the pooling operation. In this case, the algorithm is as follows.

1. Choose two different wavelet basis and compute their associated matrices W_1 and W_2.
2. Present the feature image F and perform, in parallel, the two associated discrete wavelet transforms $W_1 F W_1^T$ and $W_2 F W_2^T$.
3. Discard HL, LH, HH from every matrix, this only tacking into account the approximated image LL_1 and LL_2 by the two different basis.
4. Concatenate the two results and pass on to the next layer.

5 Validation

In this section, we discuss the 3 datasets we used for the purposes of the validation, the validation measures, tests and comparison of the different pooling layers performance.

5.1 Dataset

The first used dataset has been MNIST, as seen in Figure 14.

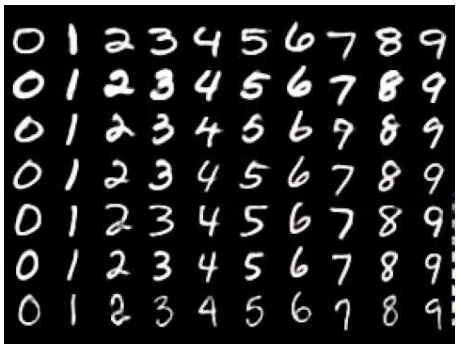


Figure 14: Example of MNIST dataset.

This dataset is composed of images of handwritten digits, all grayscale images of size 28×28. We use the full training set of 60.000 images and the full testing set of 10.000 images.

Our second used dataset has been CIFAR-10, as seen in Figure 15.

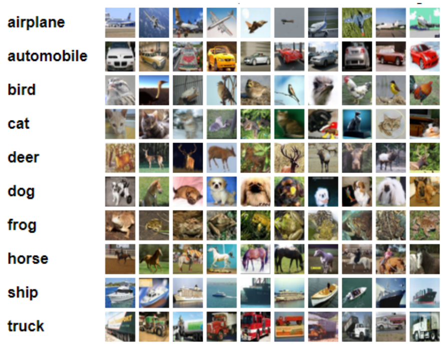


Figure 15: Example of CIFAR-10

This dataset is composed of RGB images of size 32×32 from ten different categories: airplanes, automobiles, birds, cats, deers, dogs, frogs, horses, ships and trucks. The full training set of 50,000 images is used, as well as the full testing set of 10,000 images.

Our third and last dataset is Street View House Number of SVHN for short, as seen in Figure 16.

Figure 16: Example of SVHN

For this one, we used the “cropped digits” version, all RGB images of size 32×32. We use the full training set of 55.000 images and the full testing set of 26.032 images.

5.2 Validation methodology

To evaluate the effectiveness of each pooling method, we will use the metric accuracy. To compare the convergence of each pooling method, we will use the categorical loss entropy as loss function.

5.3 Settings

We will now present the technical details about every neural network model.

5.3.1 MNIST

For the MNIST experiments, we have used a batch size of 600, we performed 20 epochs and we used a learning rate of 0.01.

5.3.2 CIFAR-10

For the CIFAR-10 experiments, we used a batch size of 500 and a dynamic learning rate.

For the case without dropout, we performed 45 epochs and the learning rate changes: 0.05 for epochs between 1 and 30; 0.005 for epochs between 30 and 40; and 0.0005 for epochs between 40 and 45.

For the case with dropout, we used two layers of dropout with rate 0.01, we performed 75 epochs and the learning rate changed as: 0.05 for epochs between 1 and 45; 0.005 for epochs between 40 and 60; 0.0005 for epochs between 60 and 70; and finally 0.00005 for epochs between epochs 70 and 75.

5.3.3 SVHN

For the SVHN experiments, we used a batch size of 700, we performed 45 epochs without dropout and we used the same dynamic changing learning rate that in the case of CIFAR-10 without dropout.

5.4 Experiments

5.4.1 MNIST

We conducted a set of experiments over MNIST, following the architecture described in [20], that we can see in Figure 17. We tested all the described pooling methods (max, average, wavelet and multiwavelet).

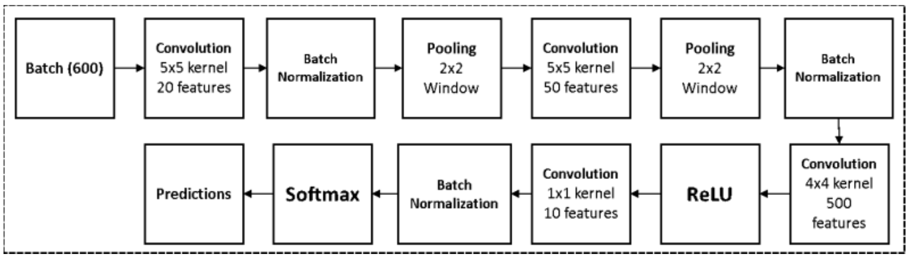

Batch (600)
Convolution 5x5 kernel 20 features
Batch Normalization
Pooling 2x2 Window
Convolution 5x5 kernel 50 features
Pooling 2x2 Window
Batch Normalization
Convolution 4x4 kernel 500 features
ReLU
Convolution 1x1 kernel 10 features
Batch Normalization
Softmax
Predictions

Figure 17: CNN block diagram structure for MNIST, image from [20].

	Haar	Average	**Max**
Accuracy (%)	98.15	98.36	**98.93**

Table 1: Comparison of Haar, Average and Max pooling methods.

	Haar	Daubechie	Coiflet	Biorthogonal
Accuracy %	98.15	98.29	98.08	97.96

Table 2: Comparison between the choice of different wavelets basis.

In this case, Tables 1 and 2 show that although our proposed method is the third with higher accuracy, it is very consistent independently of the choice of wavelet basis. What is more, the difference between the results is very small, so our method could be a safe choice for this dataset. In Figure 18, we can observe that the average and max pooling converge faster than our proposed algorithm; in this case, Haar and Daubechie are the chosen basis that converge faster. No further experiments were conducted with this dataset because of the simplicity of it. In Figure 19, we present a selection of the predictions for each pooling method. For every result, the first row represents correct predictions while the second one represents wrong predictions.

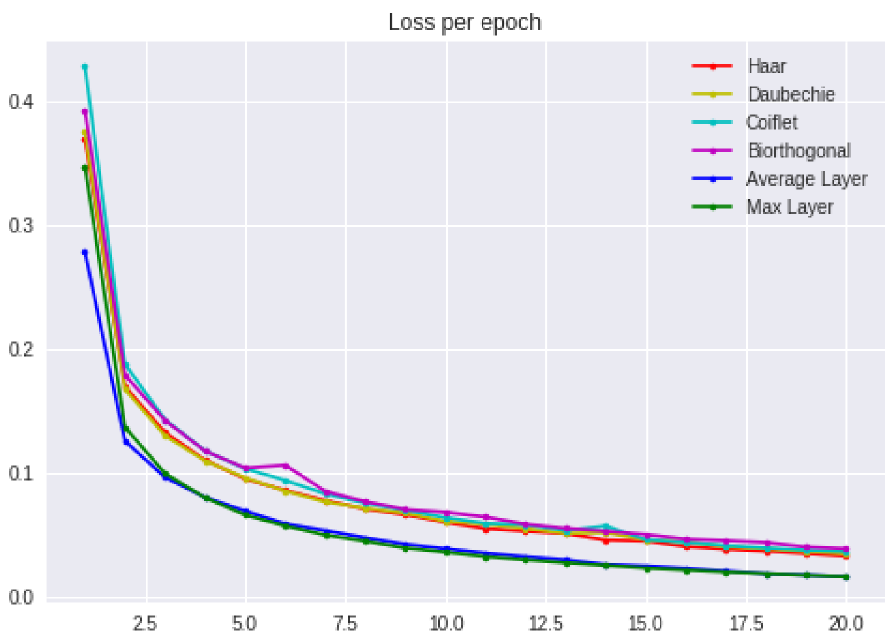
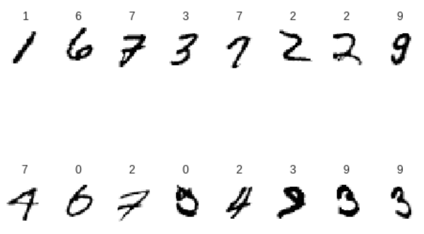
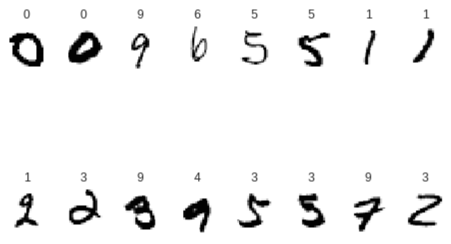


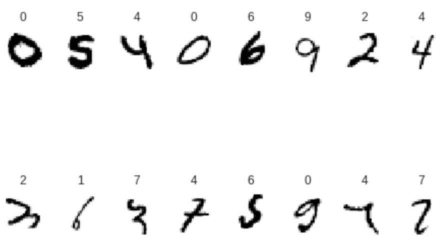
Figure 18: MNIST architecture loss function.



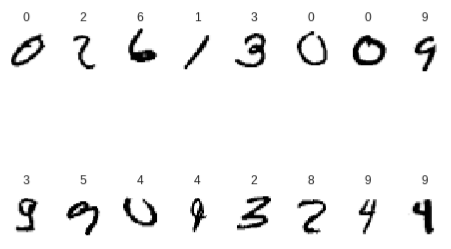
(a) Haar results.



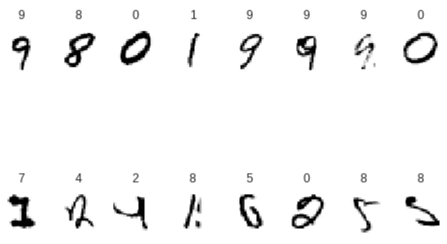
(b) Daubechie results.



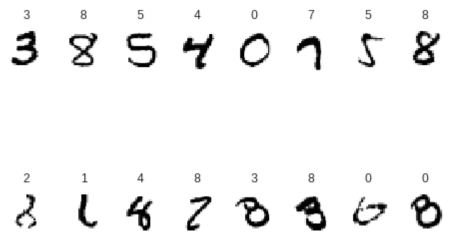
(c) Coiflet results.



(d) Biorthogonal results.



(e) Average results.



(f) Max results.

Figure 19: Predictions for each different pooling method.

5.4.2 CIFAR-10

We conducted several experiments over CIFAR-10. We tried all pooling methods (max, average, wavelet and multiwavelet). We used two different architectures: one with more epochs and with dropout; and one with less epochs and without dropout. Additionally, we used a dynamic learning rate in both cases. In Figure 20, we can see the CNN architecture.

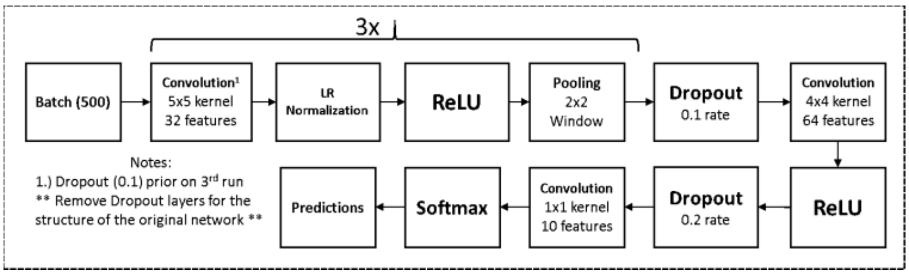

3x
Batch (500)
Convolution¹ 5x5 kernel 32 features
LR Normalization
ReLU
Pooling 2x2 Window
Dropout 0.1 rate
Convolution 4x4 kernel 64 features
Notes:
1.) Dropout (0.1) prior on 3rd run
** Remove Dropout layers for the structure of the original network **
Predictions
Softmax
Convolution 1x1 kernel 10 features
Dropout 0.2 rate
ReLU

Figure 20: CNN block diagram structure for CIFAR-10, image from [20].

For the case without dropout, Tables 3, 4 and 5 show that our proposed method outperforms max pooling and is consistently the second best method. Although the average method is the one with highest accuracy, all different choices with wavelets (different basis or combining different wavelets) come to a close second. Additionally, the Daubechie basis performs almost as well as the average method.

	Haar	**Average**	Max
Accuracy (%)	74.49	**76.40**	73.35

Table 3: Comparison for CIFAR-10 without dropout.

	Haar	Daubechie	Coiflet	Biorthogonal
Accuracy %	74.49	76.12	74.94	74.82

Table 4: Different choice of wavelets for CIFAR-10 without dropout.

	Haar + Daubechie	Haar + Coiflet	Daubechie + Coiflet
Accuracy (%)	75.39	75.43	75.13

Table 5: Combinations of multiwavelet for CIFAR-10 without dropout.

In Figure 22, we can see a selection of images of the three best performance architectures: average pooling, Daubechie wavelet pooling and multiwavelet Haar and Coiflet pooling. For every image, the first row represents correct predictions and the second row, wrong predictions.

In Figure 21, it becomes clear that every wavelet pooling method converges much faster than max pooling or average pooling. Additionally, the wavelet techniques present a smoother descend than the other two pooling methods.

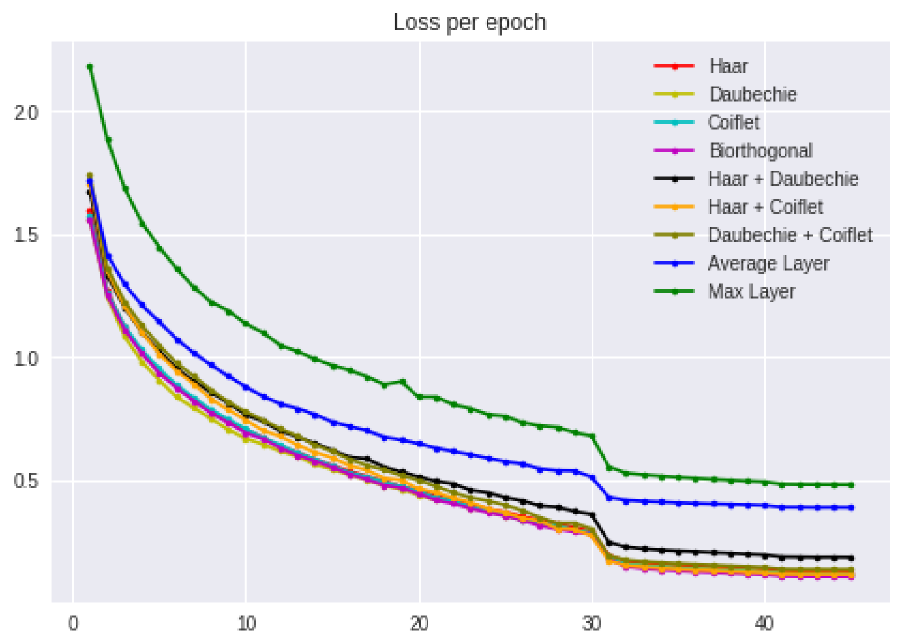
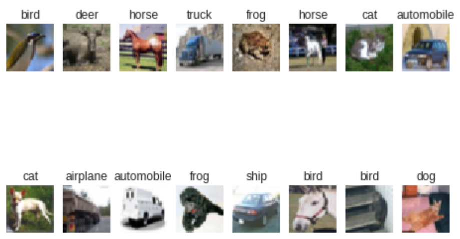


Figure 21: Cifar-10 without dropout loss function.

(a) Average results.



(b) Daubechie results.

(c) Multiwavelet haar and coiflet results.

Figure 22: Predictions for each different pooling method.

For the case with dropout, Tables 6, 7 and 8 show that the multiwavelet method outperforms the max pooling and the average pooling. In this case, the Daubechie basis comes again to a close second to the one with the highest accuracy, outperforming the average one. We can see that all of the different choices of our method outperform the max pooling, thus being a solid and safe choice for this dataset as well.

	Haar	Average	Max
Accuracy (%)	77.89	78.49	70.5

Table 6: Comparison for CIFAR-10 with dropout.

	Haar	Daubechie	Coiflet	Biorthogonal
Accuracy %	77.89	78.95	77.36	77.44

Table 7: Different choice of wavelets for CIFAR-10 with dropout.

	Haar + Daubechie	Haar + Coiflet	Daubechie + Coiflet
Accuracy (%)	**79.33**	78.8	79.19

Table 8: Combinations of multiwavelet for CIFAR-10 with dropout.

In Figure 23, we can observe that the multiwavelet pooling techniques converge much faster than average and max. Haar was, so far, the best convergence basis, but in this case, it performs similarly to average pooling. Daubechie, which yields the third best result with this architecture, converges just as fast as the other multiwavelet layers. Notice how, on epoch 40, the change of learning rate is noted for every method except for the max pooling. For epoch 60, we have another change of learning rate, but it is not as noticeable as the other one, since all methods have almost completely converged.

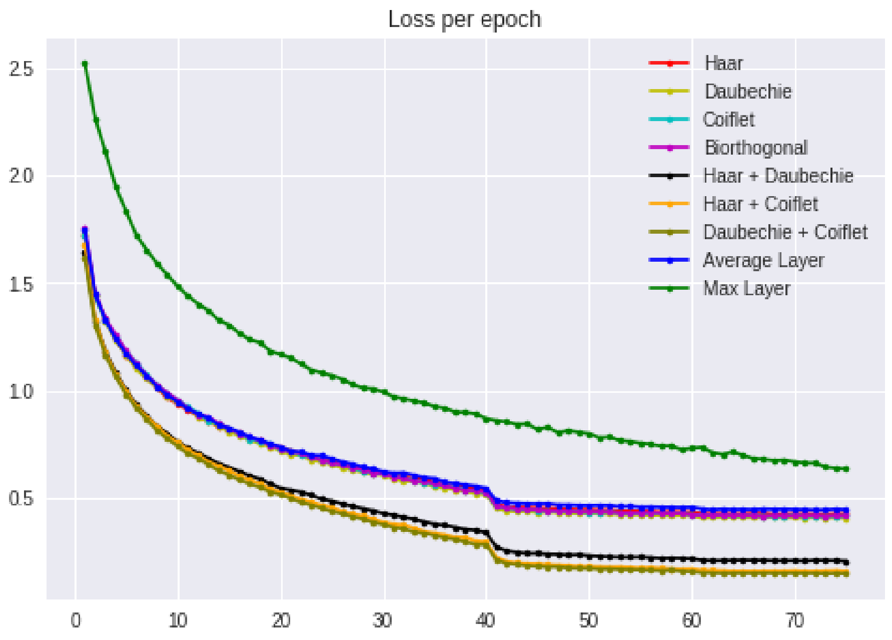


Figure 23: Cifar-10 with dropout loss function.

In Figure 24, we can see a selection of predicted images for the three best performances: multiwavelet Haar and Daubeche, Multiwavelet Daubechie and Coiflet and Daubechie wavelet. For the three images, the first row represents correct predictions and the second row, wrong predictions.

(a) Multiwavelet haar and daubechie results.

(b) Multiwavelet daubechie and coiflet results.

(c) Daubechie results.

Figure 24: Predictions for each different pooling method.

Results from the case with dropout are much more interesting since, nowadays, seeing an architecture without dropout is very rare. We consider this special case, then, as an academic exercise and as a means to test each method tolerance to overfitting. However, it is the case with dropout that hints the true potential of every method: and, in this case, wavelet and multiwavelet pooling outperform the max and the average pooling.

5.4.3 SVHN

We have conducted the same experiments over SVHN that we did with CIFAR-10, but only focusing on the case without dropout due to computational resources. In Figure 25 we can see the CNN architecture.

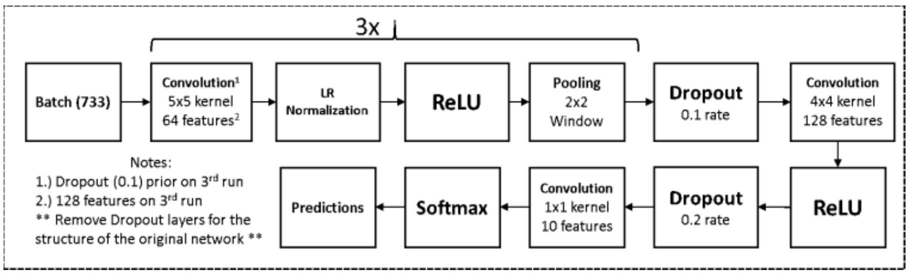

3x
Batch (733)
Convolution¹ 5x5 kernel 64 features²
LR Normalization
ReLU
Pooling 2x2 Window
Dropout 0.1 rate
Convolution 4x4 kernel 128 features
ReLU
Dropout 0.2 rate
Convolution 1x1 kernel 10 features
Softmax
Predictions
Notes:
1.) Dropout (0.1) prior on 3rd run
2.) 128 features on 3rd run
** Remove Dropout layers for the structure of the original network **

Figure 25: CNN block diagram structure for SVHN, image from [20].

	Haar	Average	Max
Accuracy (%)	89.23	88.27	88.51

Table 9: Comparison for SVHN without dropout.

	Haar	Daubechie	Coiflet	Biorthogonal
Accuracy %	89.23	88.27	89.20	89.42

Table 10: Different choice of wavelets for SVHN without dropout.

	Haar + Daubechie	Haar + Coiflet	Daubechie + Coiflet
Accuracy (%)	**91.01**	90.61	90.88

Table 11: Combinations of multiwavelet for CIFAR-10 with dropout.

In this case, Tables 9, 10 and 11 show that our proposed method, in any of its variants, outperforms both the max and the average pooling. Again, the multiwavelet method using Haar and Daubechie basis is the one with highest accuracy and the other multiwavelet choices come to close second. All wavelet choices outperform max and average pooling, but in this case Haar proves to be more consistent.

In Figure 26, we can see that max pooling performs really well on this dataset, being one of the fastest convergence method. However, again, the three combinations of the multiwavelet pooling perform just as fas as max pooling. Any other choice of a wavelet basis converges as fast as the average pooling.

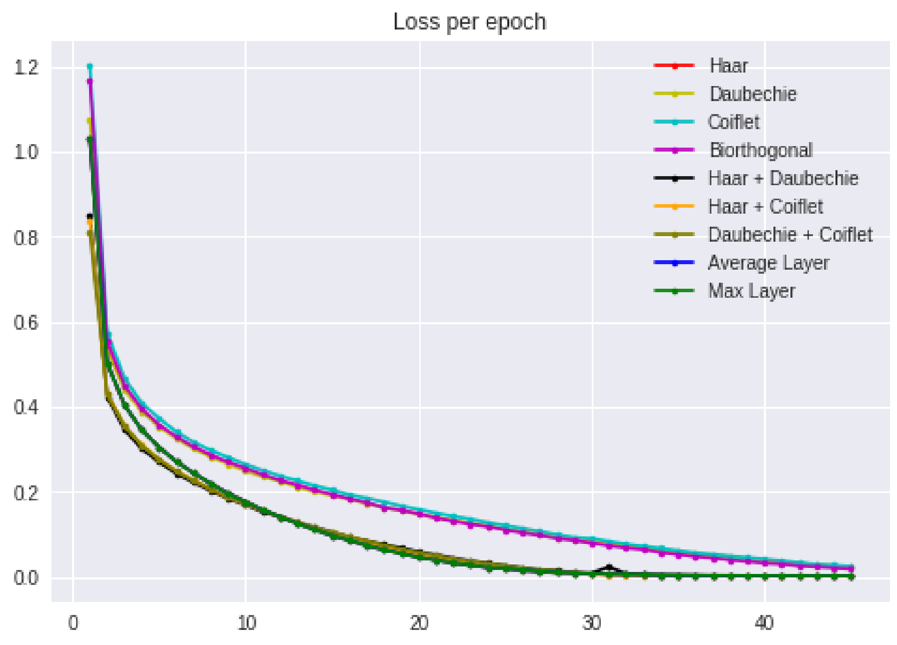
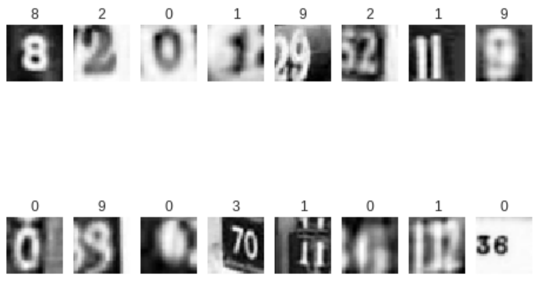
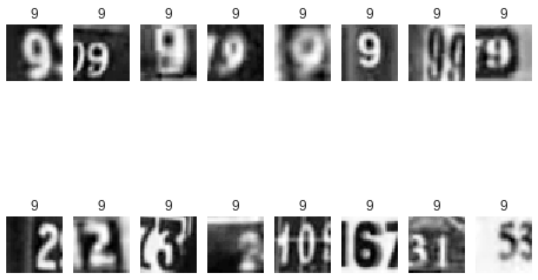


Figure 26: SVHN architecture loss function.

We can see in Figure 27 a selection of predicted images by the two highest accuracy networks. For every set of images, the first row represents correct predictions and the second row, wrong predictions. The result, in the first case is quite good: for the multiwavelet with Haar and Daubechie basis, the first three misspredicted photos could actually be what the neural network says it should be. The fifth, sixth and seventh ones could be interpreted as the neural network thinks, although it is not as clear as the first two pictures. The last set of images proves that the last network is ”obsessed” with the number nine.



(a) Multiwavelet haar and daubechie results.



(b) Multiwavelet daubechie and coiflet results.

Figure 27: Predictions for different pooling methods.

5.5 Comparison

In Table 12 we can see the standing ranking of pooling methods, ordered by highest accuracy achieved. In general, any of our different pooling methods are proven to be safe choices for those datasets. The lower performance in MNIST can be explained by the over simplicity of the dataset. For all of the other cases, our methods either outperform the max and average or come second to one of them by a short difference. The Haar basis has very consistent results, while the other wavelet basis choices (Daubechie, Coiflet, Biorthogonal) are more sensible to changes. Multiwavelet combinations require much more computational time, but in general, yield better results and outperform the other methods.

	MNIST	CIFAR-10 no dropout	CIFAR-10 dropout	SVHN
First	Max	Average	Haar + Daubechie	Haar + Daubechie
Second	Average	Daubechie	Daubechie + Coiflet	Daubechie + Coiflet
Third	Daubechie	Haar + Coiflet	Daubechie	Haar + Coiflet

Table 12: Ranking of pooling methods.

In terms of convergence, we have seen that multiwavelet combinations are the fastest convergence methods, outperforming both max and average pooling. The Haar basis proves to be very consistent as well: it is not the best method in terms of convergence, but is never the worse. The Daubechie method converges really fast as well, sometimes at he same level as the multiwavelet combinations.

In summary, any of our pooling methods can be used as a standard choice at the same level as max and average pooling are used nowadays.

6 Conclusions and future lines

Wavelet pooling has proven to be a solid pooling method. The different choice of basis offer a wide variety of possibilities, which means that this method is very versatile and can be adapted to many datasets. When trying new things, one can chose max pooling or average pooling because they are standard methods that work decently in most cases; however, all of our variants of wavelet pooling have been proven to be perfect candidates for new architectures. They offer a safe choice: not always outperforming, but being consistent across different datasets and architectures. What is more, in some cases, they are the best option.

Wavelet pooling takes longer to perform because it is based on matrices. For an average pooling with windows size 2×2, we will compute 4 operations (adding and dividing); for a wavelet pooling, we will have to perform n^2 operations, where n is the size of our feature image. This means that the max pooling and average pooling time performance depends exclusively of the window size; for wavelet pooling, however, it depends on the image size. What is more, with max and average we can freely change the window size (together with stride) for a better setting of the output dimension. In the wavelet case, we can only perform reductions of step 2 (this means that first we reduce by half, then by a quarter and so on). In every step, we need to perform a matrix multiplication, and so the cost rises. For the case of multiwavelet pooling, this cost is doubled as we are performing two wavelet pooling methods in parallel and then concatenating them. However, every method has proved to converge, if not the same, faster than average and max pooling. This means that the same result could be achieved with less iterations, thus saving computational time.

In conclusion, wavelet pooling and multiwavelet pooling have been proven to be pooling methods capable of competing with their counterparts max and average pooling. Due to the good results of this work, we will try to publish in the ECCV, at the *Women in Computer Vision* workshop.

Finally, some insight about what could come next:

1. Due to computation resources, the extent of datasets is reduced. It would be interesting to further test the consistency of our pooling method against bigger datasets, such as CIFAR-100 and CALTECH-101/256.

2. The natural continuation of wavelets are *shearlets*. Shearlets are mathematical functions that were first introduced in 2006 for analysis and sparse approximation of functions [28]. Their implementation is very similar to wavelets, but they are based on parabolic scaling matrices and shear matrices. This makes them an useful tool for studying anisotropic features. It would be interesting to study the possibility of introducing shearlets as a pooling method, combining the pooling capacity together with feature detection.

3. This work has been very focused on the importance of pooling and reducing dimensions, but always working with powers of 2. T. Lindeberg introduced in 1994 the theory of *scale space* [29]. The notion of scale selection refers to methods for estimating characteristic scales in images for automatically determining locally appropriate scales, so as to adapt subsequent processing to the local image structure and compute scale invariant image features and image descriptors [30]. It would be

interesting to preprocess our dataset to obtain a statistic to decide which image size of reduction would perform better for our data.

References

[1] Albert Boggess and Francis J. Narcowich. *A First Course in Wavelets with Fourier Analysis*, 2nd ed. Wiley, 2009, Hoboken, NJ.

[2] James S. Walker. Wavelet-based Image Compression, University of Wisconsin–Eau Claire.

[3] Wavelets for Partial Differential Equations. In: Numerical Solutions of Partial Differential Equations. Advanced Courses in Mathematics - CRM Barcelona. Birkhäuser Basel (2009).

[4] Melani I. Plett. Transient Detection With Cross Wavelet Transforms and Wavelet Coherence, IEEE Transactions on Signal Processing (Volume: 55, Issue: 5, May 2007).

[5] Tang, Y. Front. Comput. Sci. China (2008) 2: 268. https://doi.org/10.1007/s11704-008-0012-0

[6] Guan-Chen Pan. A Tutorial of Wavelet for Pattern Recognition, National Taiwan University, Taipei, Taiwan, ROC.

[7] S. Livens, P. Scheunders, G. van de Wouwer, D. Van Dyck. Wavelets for texture analysis, an overview, 1997 Sixth International Conference on Image Processing and Its Applications.

[8] Manojit Roy, V. Ravi Kumar, B. D. Kulkarni, John Sanderson, Martin Rhodes, Michel vander Stappen. Simple denoising algorithm using wavelet transform, November 8, 2013.

[9] Mikko Ranta. Wavelet Multiresolution Analysis of Financial Time Series, Vaasan Yliopisto, April 2010.

[10] Jonathan N Bradley, Christopher M. Brislaw, Tom Hopper. The FBI WaveletScalar Quantization Standard for grayscale ngerprint image compression.

[11] Raghuram Rangarajan, Ramji Venkataramanan, Siddharth Shah. Image Denoising Using Wavelets.

[12] Sachin D. Ruikar, Dharmpal D. Doye. Wavelet Based Image Denoising Technique. (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 2, No.3, March 2011

[13] David H. Hubel, Tosrten N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1359523/pdf/jphysiol01247-0121.pdf

[14] David H. Hubel, Tosrten N. Wiesel. Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat. https://www.physiology.org/doi/pdf/10.1152/jn.1965.28.2.229

[15] Michael W. Frazier, *An Introduction to wavelets through linear algebra* [Recurs electrònic] / Michael W. Frazier, New York, Springer, cop. 1999

[16] Patrick J. Van Fleet. *Wavelet Workshop*. University of St. Thomas, June 7-10, 2006. `http://personal.stthomas.edu/pjvanfleet/wavelets/ust2006workshop.html`

[17] Python PyWavelets library. `https://pywavelets.readthedocs.io/en/latest/`

[18] Ingrid Daubechies. Ten Lectures on Wavelets. SIAM, 1992, Philadelphia, PA.

[19] G. Beylkin, R. Coifman, and V. Rokhlin (1991), *Fast wavelet transforms and numerical algorithms*, Comm. Pure Appl. Math.

[20] Travis Williams, Robert Li, *Wavelet pooling for convolutional neural network*, ICLR 2018. `https://openreview.net/pdf?id=rkhlb8lCZ`

[21] Standford class, *Module 1: Neural Networks*, `http://cs231n.github.io/neural-networks-1/`

[22] Standford class, *Module 2: Convolutional Neural Networks*, `http://cs231n.github.io/convolutional-networks/`

[23] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks.* `http://papers.nips.cc/paper/4824-imagenet-classification -with-deep-convolutional-neural-networks`

[24] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In NIPS Workshop on Deep Learning and Unsupervised Feature Learning, 2011.

[25] Chen-Yu Lee, Patrick W. Gallagher, and Zhuowen Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, pp. 464–472, 2016.

[26] Matthew Zeiler and Robert Fergus. Stochastic pooling for regularization of deep convolutional neural networks. In Proceedings of the International Conference on Learning Representation (ICLR), 2013.

[27] Yu Chen, I. Lopez-Moreno, T. N. Sainath, M. Visontai, R. Alvarez, C. Parada. Locally-Connected and Convolutional Neural Networks for Small Footprint Speaker Recognition, Google INTERSPEECH 2015.

[28] Guo, Kanghui, Gitta Kutyniok, and Demetrio Labate. ”Sparse multidimensional representations using anisotropic dilation and shear operators.” Wavelets and Splines (Athens, GA, 2005), G. Chen and MJ Lai, eds., Nashboro Press, Nashville, TN (2006): 189–201

[29] T. Lindeberg. “Scale-space theory: A basic tool for analysing structures at different scales”, Journal of Applied Statistics 21(2): 224-270, (1994).

[30] T. Lindeberg.“Scale selection”, Computer Vision: A Reference Guide, (K. Ikeuchi, ed.), pages 701-713, (2014).