Integrating Wavelet Theory into Convolutional Neural Networks

2023-2024

Éloi Navet

January 28, 2024

Contents

1 Introduction

1.1 Convolutional Neural Networks (CNNs) for Image Classification

Convolutional Neural Networks Convolutional Neural Networks (CNNs) represent a powerful class of deep neural networks meticulously crafted for processing structured grid data, with a primary focus on image analysis [1, 2, 3]. The foundational principles of CNNs encompass:

1. **Convolutional Layers:** Leveraging filters to discern features like edges and patterns via convolution operations;
2. **Pooling Layers:** Downsizing spatial dimensions to reduce complexity while preserving crucial information. These will be in the center of our field of study and discussed in section 2.3.1;
3. **Activation Functions:** Introducing non-linearities, such as ReLU, to capture intricate relationships;
4. **Fully Connected Layers:** Synthesizing features acquired from prior layers for conclusive decision-making;

They iteratively adjust their weights to minimize the disparity between predicted and actual outputs, through the process of back-propagation.

Image Classification Designed to tackle image classification challenges, CNNs have surpassed human performance on databases like ImageNET [4]. Archetypes such as AlexNET [5], ResNET [3], VGG [6], and LeNET [7] showcase the capability of CNNs in automatically learning hierarchical and spatially invariant features, establishing their efficiency in object recognition.

1.2 The problem of CNNs against image transformations

Deep convolutional networks generalize poorly to small image transformations primarily due to the combination of subsampling operations, commonly known as "stride," and the violation of the sampling theorem. The subsampling factor in modern CNNs, resulting from multiple layers of subsampling operations, can be large, making literal translation invariance applicable only to specific translations that align with the subsampling factors. This failure of translation invariance has been discussed in the context of shiftability, a weaker form of translation invariance [8].

The sampling theorem, as described by the Shannon-Nyquist theorem, plays a critical role. Shiftable representations, where the response of a feature detector can be linearly interpolated from the sampling grid, are crucial for preserving translation invariance during subsampling. However, the brittleness arises when the subsampling factor is large, preventing effective shiftability and resulting in sensitivity to small image transformations.

The poor generalization of deep convolutional networks to small image transformations can be attributed to the interplay of subsampling, violation of the sampling theorem, and the lack of effective shiftability in representations [9].

1.3 Use of Wavelets in Image Processing

Wavelets are mathematical functions characterized by their ability to decompose signals into distinct frequency components. They serve as versatile tools in various aspects of image processing. Their unique properties make them particularly well-suited for tasks such as:

1. **Denoising:** Wavelet transforms excel in isolating noise from image data. By decomposing an image into different frequency bands, wavelets facilitate the identification and removal of unwanted noise, preserving essential details and enhancing overall image quality [10, 11].
2. **Image Compression:** The adaptability of wavelet transforms is harnessed in image compression techniques. Wavelets allow for efficient representation of image information by concentrating on the most critical components and discarding less essential details. This results in compressed images with reduced storage requirements while maintaining perceptual quality [11]. The most known application of this is JPEG2000, which provided a state-of-the-art efficient compression method using wavelets [12].

3. **Feature Extraction:** Wavelet analysis aids in extracting meaningful features from images. The decomposition of an image into different frequency scales allows for the identification and isolation of specific patterns, edges, or textures, contributing to more effective feature extraction in various applications, for example in medicine [13, 14].
4. **Multiresolution Analysis:** The inherent multiresolution nature of wavelets enables a detailed analysis of images at different scales. This capability is particularly valuable in applications where a comprehensive understanding of image structures at varying levels of detail is required [15].

By leveraging these capabilities, wavelets play a pivotal role in addressing diverse challenges in image processing, making them indispensable tools in fields ranging from computer vision to medical imaging.

1.4 Significance of Integrating Wavelets into Neural Networks

Since wavelets are continuous mathematical tools, their implementation into the computer world goes through a discretization. The Discrete Wavelet Transform (DWT) decomposes data into frequency components, while the Inverse DWT (IDWT) reconstructs data from the DWT output (see section 2.1). In the subject of our study [16], DWT is applied for image denoising in neural networks, these being sensible of noise while doing their task of image classification.

Wavelets have been successfully integrated into neural networks for tasks like function approximation, signal representation, and classification. Early attempts focused on shallow networks to identify optimal wavelets, while recent efforts involve deeper networks for image classification, facing challenges due to computational costs. For instance, [17] combines wavelet transform with nonlinear modulus and average pooling, outperforming CNNs in tasks like handwritten digit recognition and texture discrimination.

In deep learning [18], wavelets are commonly employed for image preprocessing or postprocessing. Researchers have explored direct integration of wavelet transforms into deep network designs, treating them as sampling operations. Multi-level Wavelet CNN (MWCNN) integrates Wavelet Package Transform (WPT) for image restoration, concatenating low- and high-frequency components. Convolutional-Wavelet Neural Network (CWNN) uses dual-tree complex wavelet transform (DT-CWT) for noise suppression in SAR images. Another approach, Wavelet pooling, utilizes a two-level DWT.

Previous works often tested with a limited set of wavelets due to the lack of general wavelet transform layers. The central article of this report ([16]) systematically explores the potential of wavelet integration in standard image datasets like ImageNet, applying wavelet transforms as network layers for a more comprehensive understanding of their impact on deep networks.

It can be expected that wavelet-based neural networks will provide better results on image classification tasks, especially when the images are noisy. This hypothesis will be tested by implementing wavelet-based neural networks and comparing them to classical CNNs.

1.5 Outline of the Report

The subject of this report is to dive into the results proposed by [16]. Following their presentation, we will try to explain the background behind these results, reproduce and discuss on them.

First of all, we will present the theoretical background of wavelets and their use in image processing in section 2. Then, we will present the different models we implemented in section 3 and the results we obtained, which will be discussed in section 4. Finally, section 5 will summarize what have been achieved and the future work that can be done.

2 Construction of Wavelet Layers in CNNs

2.1 2D (Inverse) Discrete Wavelet Transform

2.1.1 Discrete Wavelet Transform

The Discrete Wavelet Transform (DWT) decomposes an approximation of a signal in a wavelet basis $P_{\mathbf{V}_j} f$ into a coarser approximation $P_{\mathbf{V}_{j+1}} f$ and wavelet coefficients carried by $P_{\mathbf{W}_{j+1}} f$.

Consider the orthonormal bases $\{\phi^2_{j,n}\}_{n\in\mathbb{Z}}$ and $\{\psi^k_{j,n}\}^{k\in[\![1,3]\!]}_{n\in\mathbb{Z}}$ of $\mathbf{V}_j$ and $\mathbf{W}_j$, respectively (with $k \equiv 1$ in the one-dimensional case). The projections of f in these bases for all scales 2^j and n ($= (n_1, n_2)$ in the two-dimensional case) are defined as follows:

$$a_j[n] = \langle f, \phi^2_{j,n} \rangle \quad \text{and} \quad d^k_j[n] = \langle f, \psi^k_{j,n} \rangle \quad \text{for} \quad 1 \leq k \leq 3. \tag{1}$$

For any pair of one-dimensional filters $y[m]$ and $z[m]$, the product filter $yz[n] = y[n_1]z[n_2]$ and $\bar{y}[m] = y[-m]$ are defined. Let $h[m]$ and $g[m]$ be the conjugate mirror filters associated with the wavelet ψ.

1D Discrete Wavelet Transform Before showing the 2D-result, let's write the formula in 1D. For any $p \in \mathbb{Z}$, in one dimension, the signal is decomposed using the formula:

$$a_{j+1}[p] = \sum_{n=-\infty}^{+\infty} h[n-2p]a_j[n] = a_j * \bar{h}[2p], \tag{2}$$

$$d_{j+1}[p] = \sum_{n=-\infty}^{+\infty} g[n-2p]a_j[n] = a_j * \bar{g}[2p]. \tag{3}$$

Note that here $k = 1$, since the case $k \in [\![1, 3]\!]$ is reserved to the 2D-case.

2D Discrete Wavelet Transform Knowing the decomposition at scale 2^j, the wavelet coefficients at scale 2^{j+1} are calculated from a_j with two-dimensional separable convolutions and sub-samplings. The decomposition formulas are obtained by applying the one-dimensional convolution formulas (2) and (3) to the separable two-dimensional wavelets and scaling functions for $n = (n_1, n_2)$:

$$a_{j+1}[n] = a_j * \bar{h}\bar{h}[2n], \tag{4}$$

$$d^1_{j+1}[n] = a_j * \bar{h}\bar{g}[2n], \tag{5}$$

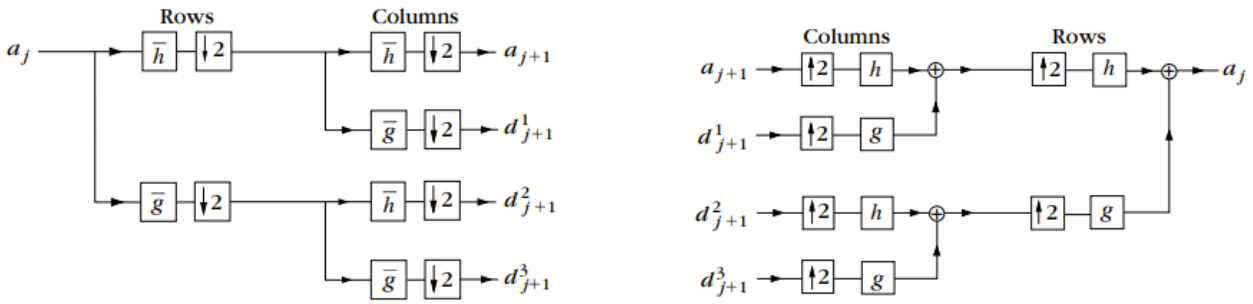
$$d^2_{j+1}[n] = a_j * \bar{g}\bar{h}[2n], \tag{6}$$

$$d^3_{j+1}[n] = a_j * \bar{g}\bar{g}[2n]. \tag{7}$$

Assuming separable filters h and g ($\forall p \in \mathbb{Z}^2,\ h[p_1, p_2] = h[p_1]h[p_2]$), a two-dimensional convolution can be factored into one-dimensional convolutions along the rows and columns of the image.

$$\begin{aligned} f * h[n] &= \sum_{p\in\mathbb{Z}^2} h[n-p]f[p] \\ &= \sum_{(p_1,p_2)\in\mathbb{Z}^2} h[n_1-p_1, n_2-p_2]f[p_1,p_2] \\ &= \sum_{(p_1,p_2)\in\mathbb{Z}^2} h[n_1-p_1][n_2-p_2]f[p_1,p_2] \text{ since } h \text{ is separable} \\ &= \sum_{p_1\in\mathbb{Z}} h[n_1-p_1] \sum_{p_2\in\mathbb{Z}} h[n_2-p_2]f[p_1,p_2]. \end{aligned} \tag{8}$$

This is a key point since it allows to do just six convolutions instead of eight: the first two are done on the rows and the ones on the columns use the first result, described in fig. 1a.



(a) Decomposition of a_j with six groups of one-dimensional convolutions and subsamplings along the image rows and columns.

(b) Reconstruction of a_j by inserting zeros between the rows and columns of a_{j+1} and d^k_{j+1}, and filtering the output.

Figure 1: DWT and IDWT in two steps, allowing to make only six operations instead of eight (Source [11]).

2.1.2 Inverse Discrete Wavelet Transform

The reverse process involves reconstructing each $P_{\mathbf{V}_j} f$ from $P_{\mathbf{V}_{j+1}} f$ and $P_{\mathbf{W}_{j+1}} f$.

1D Inverse Discrete Wavelet Transform In one dimension, the signal is retrieved from the decomposition coefficients using the formula:

$$\begin{aligned} a_j[p] &= \sum_{n=-\infty}^{+\infty} h[p-2n]a_{j+1}[n] + \sum_{n=-\infty}^{+\infty} g[p-2n]d_{j+1}[n] \\ &= \check{a}_{j+1} * h[p] + \check{d}_{j+1} * g[p]. \end{aligned} \tag{9}$$

Here there is also just one d coefficient, with $k = 1$.

2D Inverse Discrete Wavelet Transform The approximation a_j is recovered from the coarser-scale approximation a_{j+1} and the wavelet coefficients d^k_{j+1} with two-dimensional separable convolutions derived from the one-dimensional reconstruction formula (9):

$$a_j[n] = \check{a}_{j+1} * hh[n] + \check{d}^1_{j+1} * hg[n] + \check{d}^2_{j+1} * gh[n] + \check{d}^3_{j+1} * gg[n]. \tag{10}$$

These four separable convolutions can also be factored into six groups of one-dimensional convolutions along rows and columns.

2.2 Forward and Backward Propagation of Discrete Wavelet Transform (DWT) and Inverse Discrete Wavelet Transform (IDWT) Layers

Efficiently managing the forward and backward propagation of data is a critical consideration in the design of Discrete Wavelet Transform (DWT) and Inverse Discrete Wavelet Transform (IDWT) layers. In this context, our focus is on the 2D case, particularly for image processing applications, where signals are of finite size ($N \in \mathbb{N}$). The mathematical formalism aligns with a discrete framework, ensuring compatibility with future Python implementations.

2.2.1 Forward Propagation

1D Case Considering signals $\mathbf{a_j}$ and $\mathbf{d_j}$ of size N (with N being even, typically $N = 2^k$), matrix representations are introduced for $\{a_j[p]\}_{0\leq p\leq N}$ and $\{d_j[p]\}_{0\leq p\leq N}$. The DWT formulas (2)(3) are expressed in matrix form as:

$$a_{j+1}[p] = \mathbf{H}a_j[2p] \quad \text{and} \quad d_{j+1}[p] = \mathbf{G}d_j[2p] \tag{11}$$

where $\mathbf{H}, \mathbf{G} \in \mathcal{M}_{N/2,N}(\mathbb{R})$, $\mathbf{a_j} \in \mathbb{R}^N$, and $\mathbf{a_{j+1}} \in \mathbb{R}^{N/2}$. The matrices $\mathbf{H}$ and $\mathbf{G}$ respectively represent the low-pass and high-pass filters of an orthogonal wavelet in a matrix paradigm.

For the inverse wavelet transform, the reconstruction formula (12) is expressed as:

$$\mathbf{a_j} = \mathbf{H}^T\mathbf{a_{j+1}} + \mathbf{G}^T\mathbf{d_{j+1}} \tag{12}$$

2D Case In the 2D case, applicable to images, the DWT is performed as a 1D DWT on every row and column. The forward and backward propagation formulas are detailed as follows:

$$\mathbf{X}_{ll} = \mathbf{H}\mathbf{X}\mathbf{H}^T \tag{13}$$
$$\mathbf{X}_{lh} = \mathbf{G}\mathbf{X}\mathbf{H}^T \tag{14}$$
$$\mathbf{X}_{hl} = \mathbf{H}\mathbf{X}\mathbf{G}^T \tag{15}$$
$$\mathbf{X}_{hh} = \mathbf{G}\mathbf{X}\mathbf{G}^T \tag{16}$$

and

$$\mathbf{X} = \mathbf{H}^T\mathbf{X}_{ll}\mathbf{H} + \mathbf{G}^T\mathbf{X}_{lh}\mathbf{H} + \mathbf{H}^T\mathbf{X}_{hl}\mathbf{G} + \mathbf{G}^T\mathbf{X}_{hh}\mathbf{G}. \tag{17}$$

Notably, due to the computational symmetry, only 6 matrix operations are required instead of 8, optimizing the efficiency of the computations.

3D Case Extending the process to 3D data, such as videos, involves generating different sub-bands with corresponding formulas. The reconstruction formula is presented as:

$$\begin{aligned}\mathbf{X} = {} & \mathbf{H}^T\mathbf{X}_{lll}\mathbf{H}^T\mathbf{H} + \mathbf{G}^T\mathbf{X}_{llh}\mathbf{H}^T\mathbf{H} + \mathbf{H}^T\mathbf{X}_{lhl}\mathbf{G}^T\mathbf{H} + \mathbf{G}^T\mathbf{X}_{lhh}\mathbf{G}^T\mathbf{H} \\ & + \mathbf{H}^T\mathbf{X}_{hll}\mathbf{H}^T\mathbf{G} + \mathbf{G}^T\mathbf{X}_{hlh}\mathbf{H}^T\mathbf{G} + \mathbf{H}^T\mathbf{X}_{hhl}\mathbf{G}^T\mathbf{G} + \mathbf{G}^T\mathbf{X}_{hhh}\mathbf{G}^T\mathbf{G}.\end{aligned}$$

n-D Formalism Though not utilized in our program, the formalism for decomposing an n-dimensional signal X into one low-frequency signal and $2^n - 1$ high-frequency signals can be expressed as:

$$\mathbf{X}_{c_1\dots c_n} = (\downarrow 2)(\mathbf{f}_{c_1\dots c_n} * \mathbf{X}), \quad (c_1, c_1, \dots, c_n) \in \{h, g\}^N \tag{18}$$

using the notation introduced in [11], which means that one over two coefficients are taken in their matrix representation.

2.2.2 Backward Propagation

1D Case For 1D DWT backward propagation, one differentiates (11) to obtain:

$$\frac{\partial \mathbf{a_{j+1}}}{\partial \mathbf{a_j}} = \mathbf{H}^T \quad \text{and} \quad \frac{\partial \mathbf{d_{j+1}}}{\partial \mathbf{d_j}} = \mathbf{G}^T$$

Similarly, for 1D IDWT backward propagation, the differentiation of (12) gives:

$$\frac{\partial \mathbf{a_j}}{\partial \mathbf{a_{j+1}}} = \mathbf{H} \quad \text{and} \quad \frac{\partial \mathbf{d_j}}{\partial \mathbf{d_{j+1}}} = \mathbf{G}$$

2D Case The forward and backward propagations of the 2D case are more complicated but still similar to the one presented in the 1D case, following this time the formulas (13)(14)(15)(16). Formally:

$$\begin{aligned}\frac{\partial \mathbf{X}_{ll}}{\partial \mathbf{X}}(Y) &= \mathbf{H}^T Y \mathbf{H} \\ \frac{\partial \mathbf{X}_{lh}}{\partial \mathbf{X}}(Y) &= \mathbf{G}^T Y \mathbf{H} \\ \frac{\partial \mathbf{X}_{hl}}{\partial \mathbf{X}}(Y) &= \mathbf{H}^T Y \mathbf{G} \\ \frac{\partial \mathbf{X}_{hh}}{\partial \mathbf{X}}(Y) &= \mathbf{G}^T Y \mathbf{G}\end{aligned}$$

where Y is the output from the layer following the 2D DWT layer. The backward propagation of the 2D IDWT is given by:

$$\frac{\partial \mathbf{X}}{\partial \mathbf{X}_{ll}}(Y) = \mathbf{H}Y\mathbf{H}^T$$
$$\frac{\partial \mathbf{X}}{\partial \mathbf{X}_{lh}}(Y) = \mathbf{H}Y\mathbf{G}^T$$
$$\frac{\partial \mathbf{X}}{\partial \mathbf{X}_{hl}}(Y) = \mathbf{G}Y\mathbf{H}^T$$
$$\frac{\partial \mathbf{X}}{\partial \mathbf{X}_{hh}}(Y) = \mathbf{G}Y\mathbf{G}^T$$

where Y is the output from the layer following the 2D IDWT layer.

One rewrites 1D/2D DWT and IDWT as network layers in PyTorch, which are applicable to various discrete orthogonal and biorthogonal wavelets, like Haar, Daubechies, and Cohen. In the layers, we do DWT and IDWT channel by channel for multi-channel data.

2.2.3 Choice of Wavelets

Haar Wavelets In practice, wavelets with finite filters are commonly chosen. For example, the Haar wavelet has 1D filters given by $\mathbf{h} = \frac{1}{\sqrt{2}}\{1, 1\}$ and $\mathbf{g} = \frac{1}{\sqrt{2}}\{1, -1\}$. Therefore, it is possible to create 2D filters using the tensor product of 1D filters, namely:

$$\mathbf{hh} = \mathbf{h} \otimes \mathbf{h} = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$
$$\mathbf{hg} = \mathbf{h} \otimes \mathbf{g} = \frac{1}{2}\begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}$$
$$\mathbf{gh} = \mathbf{g} \otimes \mathbf{h} = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}$$
$$\mathbf{gg} = \mathbf{g} \otimes \mathbf{g} = \frac{1}{2}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

Daubechies Orthogonal Wavelets Daubechies wavelets are orthogonal [19, 20]. Here we present Daubechies wavelets of order p. Their low-pass filter is $\mathbf{h} = \{h_k\}$. The length of the filter is $2p$. The high-pass filter $\mathbf{g} = \{g_k\}$ can be deduced from

$$g_k = (-1)^k h_{N-k},$$

where N is an odd number. When $p = 1$, Daubechies wavelets are Haar wavelets.

Cohen Biorthogonal Wavelets Cohen wavelets are symmetric biorthogonal wavelets [20]. Each Cohen wavelet has four filters $\mathbf{h}, \mathbf{g}, \tilde{\mathbf{h}}$, and $\tilde{\mathbf{g}}$. While a signal is decomposed using filters $\mathbf{h}$ and $\mathbf{g}$, it can be reconstructed using the dual filters $\tilde{\mathbf{I}}$ and $\tilde{\mathbf{g}}$. Cohen wavelet has two order parameters p and $\tilde{p}$. Their high-pass filters can be deduced from

$$g_k = (-1)^k \tilde{h}_{N-k},$$
$$\tilde{g}_k = (-1)^k h_{N-k},$$

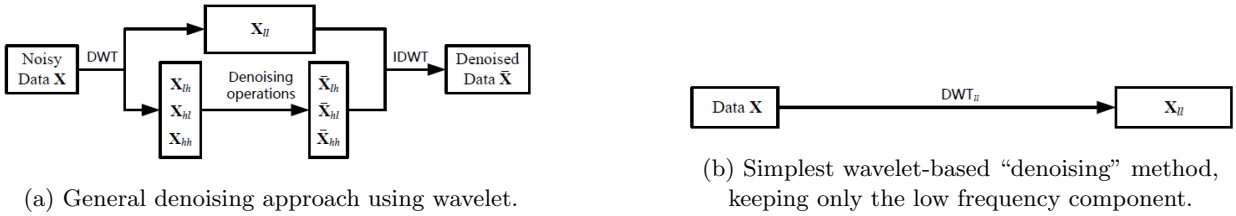
where N is an odd number. When $p = \tilde{p} = 1$, Cohen wavelets are Haar wavelets.

2.3 Integrating DWT/IDWT Layers in CNNs

From this point forward, we focus exclusively on 2D data, which remains finite. Given a signal $\mathbf{X}$, random noise predominantly manifests in its high-frequency components. As depicted in fig. 2, the general wavelet-based denoising process comprises three steps:

1. Decompose the noisy data $\mathbf{X}$ using DWT into a low-frequency component $\mathbf{X}_{II}$ and high-frequency components $\mathbf{X}_{lh}$, $\mathbf{X}_{hl}$, $\mathbf{X}_{hh}$.

2. Filter the high-frequency components, keeping only the low frequency component.
3. Reconstruct the data using IDWT with the processed components.



(a) General denoising approach using wavelet.

(b) Simplest wavelet-based "denoising" method, keeping only the low frequency component.

Figure 2: General denoising approach based on wavelet transforms and the method used in our models [16].

The following fig. 3 present the difference between keeping only the low frequency component or by keeping all the components. It can be seen that by keeping only the low frequency component we discard the noise and the image is softened.

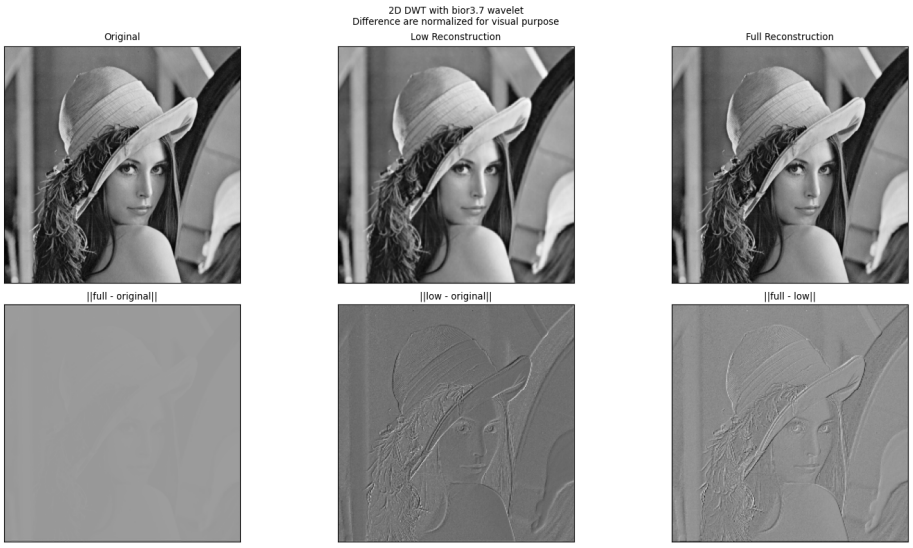


Figure 3: Visual comparison between an image reconstructed after discarding high frequency components and also by keeping them.

Since the goal is to get rid of the noise, the process presented in fig. 2b will be the one used here.

2.3.1 Pooling Layers

CNNs commonly incorporate pooling layers alongside convolutional layers to simplify information in the output. Usually placed after convolutional layers, pooling layers condense feature maps, reducing the spatial dimensions of the data [21]. An example of pooling layers is illustrated in fig. 4. Max-pooling is widely used, enabling the network to identify features within a region of the image while discarding precise positional information.

2.3.2 Integration of Wavelet-Based Pooling Layers

In this paper, we propose an alternative approach to pooling layers by integrating Discrete Wavelet Transform (DWT) and Inverse DWT (IDWT). DWT decomposes noisy data into low-frequency and high-frequency components, effectively serving as a denoising mechanism and replacing conventional downsampling operations.

While max-pooling discards precise positional information, DWT_{ll} removes high-frequency components, acting as a denoising step. The retained low-frequency component contains essential information for fea-

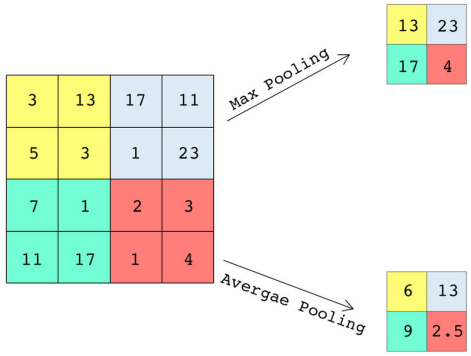


Figure 4: Example of pooling layers. Max pooling takes the maximum of each subsquare, while average pooling takes the average. Credits: [22].

ture extraction. Integrating DWT/IDWT layers not only reduces spatial dimensions but also resists noise propagation, preserving the basic object structure in feature maps.

Our approach involves the simplest wavelet-based denoising *i.e.* dropping high-frequency components, as shown in fig. 2b. This choice, focused on noise robustness, emphasizes the importance of low-frequency components. It's worth noting that retaining low-frequency components, for instance, by adding layers, could inject additional information into the learning process. DWT_{ll} transforms feature maps to the low-frequency component, and our new wavelet-based neural networks replace traditional downsampling with DWT_{ll}.

(a) Usual down-sampling operations in deep neural networks.

(b) Wavelet-integrated down-sampling in wavelet-based neural networks.

Figure 5: Pooling operations are replaced by taking only the low-frequency components. The convolutions are less strided while also taking the low-frequency DWT [16].

As fig. 5 shows, in wavelet-based neural networks, max-pooling and average-pooling are directly replaced by DWT_{ll}, while strided-convolution is upgraded using convolution with a stride of 1 followed by DWT_{ll}. Graphically:

$$\begin{aligned}\mathrm{MaxPool}_{s=2} &\to \mathrm{DWT}_{ll},\\ \mathrm{Conv}_{s=2} &\to \mathrm{DWT}_{ll} \circ \mathrm{Conv}_{s=1},\\ \mathrm{AvgPool}_{s=2} &\to \mathrm{DWT}_{ll},\end{aligned}$$

where `MaxPool`, $\texttt{Conv}_s$, and $\texttt{AvgPool}_s$ denote max-pooling, strided-convolution, and average-pooling with stride s, respectively.

While DWT_{ll} halves the size of the feature maps, it removes their high-frequency components and denoises them. The output of DWT_{ll}, *i.e.* the low-frequency component, retains the main information of the feature map for extracting identifiable features. During downsampling in wavelet-based networks, DWT_{ll} could resist noise propagation in deep networks and help maintain the basic object structure in the feature maps. Therefore, DWT_{ll} would accelerate the training of deep networks and lead to better noise robustness and increased classification accuracy.

2.3.3 `Pytorch` Code to Implement the DWT/IDWT Layers

In this part, we present a smplified version of the implementation of DWT.

DWT `Pytorch` Function Using `Autograd` function, one can define its own `Pytorch` functions. This allows to define the function to compute the discrete wavelet transform of a 2D signal, keeping only the low frequency result as described in fig. 2b. It uses the formulas defined in eqs. (13) and (17).

```python
class DWTFunction_2D_Low(Function):
    @staticmethod
    def forward(self, input, matrix_h, matrix_h_t):
        self.save_for_backward(matrix_h, matrix_h_t)
        input_ll = matrix_h @ input @ matrix_h_t
        return input_ll

    @staticmethod
    def backward(self, grad_ll):
        matrix_h, matrix_h_t = self.saved_variables
        grad_output = matrix_h.t() @ grad_ll @ matrix_h_t.t()
        return grad_output, None, None
```

Listing 1: DWT 2D function keeping only X_{ll}.

In this class, we have to define a forward and backward method. The documentation is given here Forward Method and here Backward Method.

DWT `Pytorch` Module With this function made, it is possible to define the module (see Defining a Pytorch module) to compute the matrix **H** using **??** and the coefficients given by the wavelet coefficients, using the module `Pywt`.

```python
class DWT_2D_Low(Module):
    def __init__(self, wavename):
        super(DWT_2D_Low, self).__init__()

        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

        # Get wavelet coefficients (low pass filter)
        wavelet = pywt.Wavelet(wavename)
        self.band_low = torch.tensor(wavelet.rec_lo, device=self.device)

        # Validate the size of the coefficients
        self.band_length = len(self.band_low)
        assert (self.band_length % 2 == 0), f"self.band_length: {self.band_length} % 2 != 0"

    def build_low_filter(self, input_height, input_width):
        max_size = max(input_height, input_width)
        input_half = max_size // 2
        end = None if self.band_length // 2 == 1 else (-self.band_length // 2 + 1)

        # Build the matrix H
        matrix_hb = torch.zeros((input_half, max_size + self.band_length - 2), device=self.
    device)
        for i in range(input_half):
            for j in range(self.band_length):
                matrix_hb[i, i * 2 + j] = self.band_low[j]
        matrix_h = matrix_hb[0 : input_height // 2, 0 : input_height + self.band_length -
    2][:, self.band_length // 2 - 1 : end]
        matrix_h_t = matrix_hb[0 : input_width // 2, 0 : input_width + self.band_length -
    2][:, self.band_length // 2 - 1 : end].t()

        return matrix_h, matrix_h_t

    def forward(self, input):
        assert len(input.size()) == 4, f"len(input.size()): {len(input.size())} != 4"
        matrix_h, matrix_h_t = self.build_low_filter(input.size()[-2], input.size()[-1])
        return DWTFunction_2D_Low.apply(input, matrix_h, matrix_h_t)
```

Listing 2: DWT 2D module using listing 1 function replace pooling and convolution with low frequency filtering.

Down-Sample Layer As said in the previous part, in wavelet-based CNN we replace the pooling layers and enhance the convolution process by adding a `Downsample` function using the previously crafted functions.

```python
class Downsample(Module):
    def __init__(self, wavename="haar"):
        super(Downsample, self).__init__()
        self.dwt = DWT_2D_Low(wavename = wavename)
```

```
5
6       def forward(self, input):
7           input_ll = self.dwt(input)
8           return input_ll
```

Listing 3: Downsample function to replace pooling and enhance convolutions.

2.3.4 Training of the Neural Netwrok

The training is performed using a code structure similar to the one available in the PyTorch official examples for training a model on ImageNet. The dataset utilized is from ImageNet [4], which is substantial in size (approximately 160 GB, which was a pain for training by the way).

The models are trained with the following configurations:

- Batch size: 256;
- Learning rate: 0.1;
- Learning rate decay (divided by 10) every 30 epochs;
- Learning rate scheduler to improve training accuracy;
- Data augmentation through random horizontal flips and random resizing;
- Optimizer: Stochastic Gradient Descent (SGD), known for its reliability in achieving convergence and computational simplicity [23];
- Criterion: Cross-Entropy [24]

The training code can be referenced from the `PyTorch` ImageNet example available at `this link`.

The dataset used can be found on Kaggle's ImageNet Object Localization Challenge.

3 A Brand New CNN Design for Image Classification

3.1 AlexNET

In the realm of deep convolutional networks, AlexNet stands as a pivotal contributor, notably transforming the field of machine learning [5]. Its 2012 triumph in the ImageNet LSVRC-2012 challenge, achieving an unprecedented 84.7% accuracy, marked a breakthrough in the history of deep learning and image classification. Comprising five convolutional layers and three fully connected layers, AlexNet's architectural prowess is underscored by its strategic utilization of the ReLU activation function.

Executing a paradigm shift from traditional activation functions, AlexNet's adoption of **ReLU mitigated the vanishing gradient problem**, enabling faster convergence. Local response normalization was concurrently employed to address potential challenges arising from ReLU's unbounded nature, fostering stability in the network.

The network, operates with an input size of 227x227x3 and yields a 1000x1 probability vector as output. Beyond architectural innovations, AlexNet's combat against overfitting is noteworthy. Dropout layers, incorporating a 50% connection omission probability during training, fortify the model against overfitting tendencies, though a trade-off in convergence iterations [7]. In synthesizing these elements, AlexNet emerges as a pioneering force with enduring impact on the efficacy and stability of deep convolutional networks.

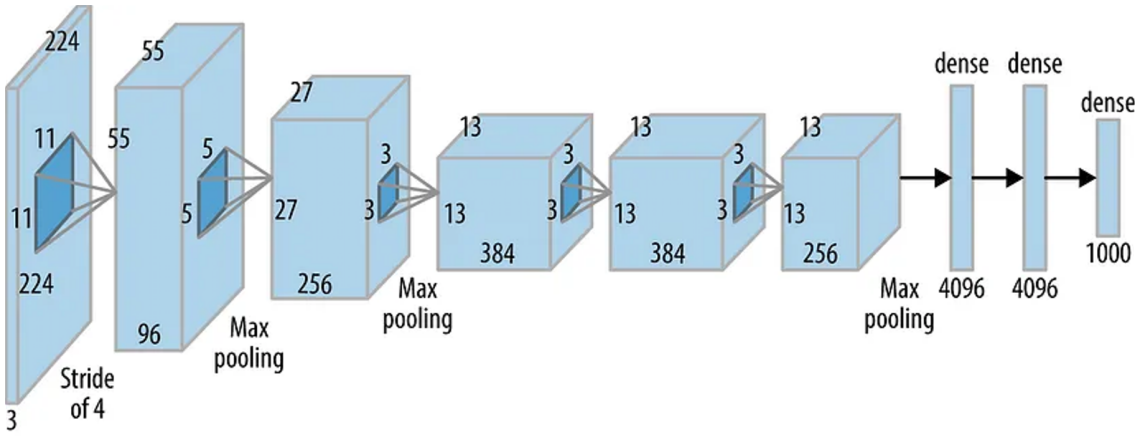


Figure 6: AlexNET architecture (Source: AlexNET).

In this work, following fig. 5, eahc of the three max-pooling layers are replaced by downsample layers described in listing 3. Also the first convolutional layer is preceded by a downsampling operation, while reducing the stride of this layer from 4 to 2.

```
class AlexNet(Module):
    def __init__(self, num_classes = 1000, wavename = "haar"):
        super(AlexNet, self).__init__()

        self.features = Sequential(
            Conv2d(3, 64, kernel_size=11, stride=2, padding=0),
            ReLU(inplace=True),
            Downsample(wavename=wavename),
            Downsample(wavename=wavename),
            Conv2d(64, 192, kernel_size=5, padding=2),
            ReLU(inplace=True),
            Downsample(wavename=wavename),
            Conv2d(192, 384, kernel_size=3, padding=1),
            ReLU(inplace=True),
            Conv2d(384, 256, kernel_size=3, padding=1),
            ReLU(inplace=True),
            Conv2d(256, 256, kernel_size=3, padding=1),
            ReLU(inplace=True),
            Downsample(wavename=wavename),
        )

```

```
        self.avgpool = AdaptiveAvgPool2d((6, 6))

        self.classifier = Sequential(
            Dropout(),
            Linear(256 * 6 * 6, 4096),
            ReLU(inplace=True),
            Dropout(),
            Linear(4096, 4096),
            ReLU(inplace=True),
            Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = x.view(x.size(0), 256 * 6 * 6)
        x = self.classifier(x)
        return x
```

Listing 4: AlexNET after replacing the pooling operation by a wavelet downsample (see fig. 5).

This model has a little more than 61 million parameters. Obviously, since the downsample layer is deterministic and only replace pooling layers, it doesn't add any parameter.

3.2 VGGNet

VGGNet, born in 2014 amid significant events like the International Year of Family Farming and Crystallography, addressed the imperative of reducing parameters in convolutional layers to enhance efficiency and training speed in deep learning [6].

VGGNet's architecture is exemplified by VGG16, comprising approximately 138 million parameters. Notably, all convolutions utilize 3x3 kernels, and maxpooling employs 2x2 kernels with a stride of 2.

The crux of innovation lies in employing 3x3 convolutions. For instance, a 5x5x1 input layer processed by a 5x5 convolutional layer yields a 1x1 output. However, employing two 3x3 convolutional layers achieves the same output, reducing trainable variables from 25 to 18, a 28% reduction. This strategy extends further; a 7x7 conv layer's effect can be replicated by three 3x3 conv layers, reducing trainable variables by 44.9%, fostering faster learning and heightened robustness against overfitting.

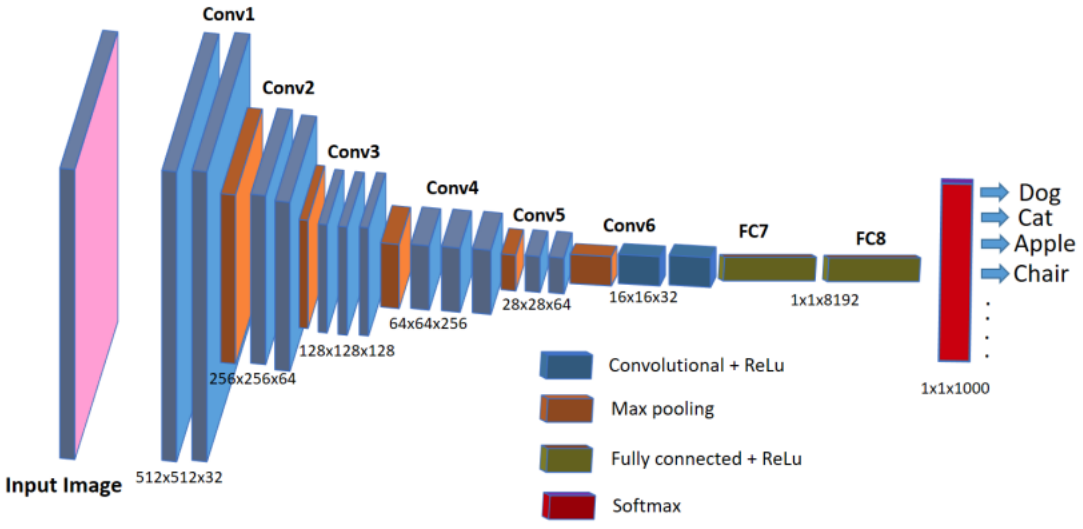


Figure 7: VGGNet architecture (Source: [25]).

Likewise, we replace the pooling layers by downsampling operations. Here, the difference is that we don't have a downsample layer preceding a convloution, like we done with AlexNET in section 3.1.

```
class VGG(Module):
    def __init__(self, num_classes = 1000, wavename = "haar"):
        super(VGG, self).__init__()

        self.features = Sequential(
            Conv2d(3, 64, kernel_size=3, stride=1, padding=1),
            ReLU(inplace=True),
            Conv2d(64, 64, kernel_size=3, stride=1, padding=1),
            ReLU(inplace=True),
            Downsample(wavename=wavename),
            Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
            ReLU(inplace=True),
            Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
            ReLU(inplace=True),
            Downsample(wavename=wavename),
            Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
            ReLU(inplace=True),
            Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            ReLU(inplace=True),
            Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
            ReLU(inplace=True),
            Downsample(wavename=wavename),
            Conv2d(256, 512, kernel_size=3, stride=1, padding=1),
            ReLU(inplace=True),
            Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            ReLU(inplace=True),
            Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            ReLU(inplace=True),
            Downsample(wavename=wavename),
            Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            ReLU(inplace=True),
            Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            ReLU(inplace=True),
            Conv2d(512, 512, kernel_size=3, stride=1, padding=1),
            ReLU(inplace=True),
            Downsample(wavename=wavename),
        )

        self.avgpool = AdaptiveAvgPool2d((7, 7))

        self.classifier = Sequential(
            Linear(512 * 7 * 7, 4096),
            ReLU(inplace=True),
            Dropout(),
            Linear(4096, 4096),
            ReLU(inplace=True),
            Dropout(),
            Linear(4096, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = x.view(x.size(0), 512 * 7 * 7)
        x = self.classifier(x)
        return x
```

Listing 5: VGGNet after replacing the pooling operation by a wavelet downsample (see fig. 5).

3.3 ResNET

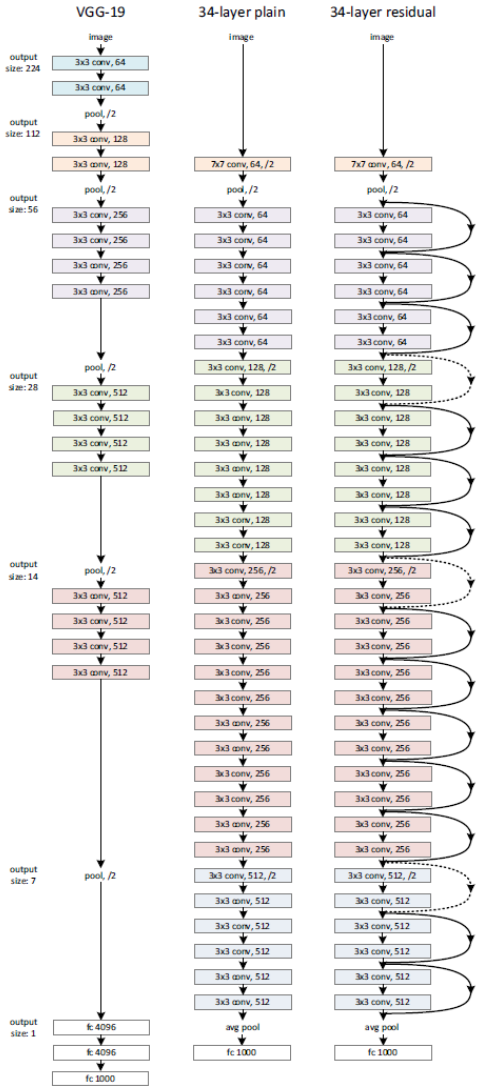


Figure 8: ResNet architecture (Source: [3]).

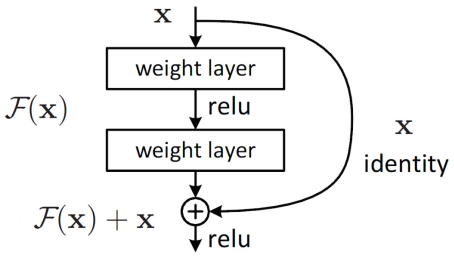


Figure 9: ResNet block (Source: [3]).

ResNet employs identity and projection layers to combat vanishing gradients in deep convolutional neural networks. This strategic approach allows ResNet models like ResNet50 and ResNet101 to achieve increased depth without succumbing to gradient-related challenges. The Residual block's focus on residual mapping further enhances learning efficiency, emphasizing adaptability in managing dimension discrepancies through Projection and Identity shortcuts. The figure fig. 8 presents the network, especially compared to VGGNet discussed in section 3.2. As one sees, this network is really deep. Since it is divided into little blocks, it does not really multiplied the number of parameters, which is still more than 21 million for ResNET-34. This means harder and longer training, espcially memorly speaking. However, in practice, it does perform really well.

Since the `Python` implementation is really long, it is not presented in this paper but it is done the same way as the two previous networks listings 4 and 5.

4 Results and Evaluation

This section presents the outcomes of training networks detailed in section 3 on the ImageNET dataset, following the methodology outlined in section 2.3.4. The evaluation encompasses various networks and wavelet types to assess and compare their effectiveness.

4.1 Classification Efficiency

The primary focus of this study is on classification. The classification efficiency of different models with various wavelets is detailed in table 1.

Table 1: Top-1 accuracy of wavelet-based networks on the ImageNet validation set, *higher is better* (extracted from [16]).

Wavelet		WVGG16bn	WResNet18	WResNet34	WResNet50	WResNet101	WDenseNet121
None (baseline)		73.37	69.76	73.30	76.15	77.37	74.65
Haar		74.10(+0.73)	71.47(+1.71)	74.35(+1.05)	**76.89**(+**0.74**)	78.23(+0.86)	75.27(+0.62)
Biorthogonal	bior2.2	74.31(+0.94)	**71.62**(+**1.86**)	74.33(+1.03)	76.41(+0.26)	78.34(+0.97)	75.36(+0.71)
	bior3.3	**74.40**(+**1.03**)	71.55(+1.79)	74.51(+1.21)	76.71(+0.56)	**78.51**(+**1.14**)	**75.44**(+**0.79**)
	bior4.4	74.02(+0.65)	71.52(+1.76)	**74.61**(+**1.31**)	76.56(+0.41)	78.47(+1.10)	75.29(+0.64)
	bior5.5	73.67(+0.30)	71.26(+1.50)	74.34(+1.04)	76.51(+0.36)	78.39(+1.02)	75.01(+0.36)
Daubechies	db2	74.08(+0.71)	71.48(+1.72)	74.30(+1.00)	76.27(+0.12)	78.29(+0.92)	75.08(+0.43)
	db3		71.08(+1.32)	74.11(+0.81)	76.38(+0.23)		
	db4		70.35(+0.59)	73.53(+0.23)	75.65(−0.50)		
	db5		69.54(−0.22)	73.41(+0.11)	74.90(−1.25)		
	db6		68.74(−1.02)	72.68(−0.62)	73.95(−2.20)		

* corresponding to the results of original CNNs, i.e., VGG16bn, ResNets, DenseNet121.
Important (personal) note: This table was taken from the article, and due to resource constraints, not all networks could be reproduced. Specific networks were selected for visual results in section 4.2. Additionally, note that *ch* has been replaced with *bior*, following the wavelet documentation of `Pywt`.

The top-1 accuracy of wavelet-based neural networks on the ImageNet validation set (50,000 images) is presented in table 1, highlighting the impact of wavelet choices on classification performance. Parenthesized numbers represent accuracy differences compared to baseline results (taken from [26] according to the article [16]). Wavelets were introduced in section 2.2.3.

Results indicate that Haar and Cohen wavelets enhance classification accuracy across various CNN architectures. However, the performance of asymmetric Daubechies and Cohen wavelets diminishes with increasing approximation order. For Daubechies, shorter filters exhibit accuracy improvement, while longer filters lead to decreased accuracy compared to the baseline. Overall, Cohen wavelets appear to be more efficient across all networks, with an overall improvement of up to +1.02%, followed by Haar with up to +0.95% improvement, and Daubechies with a modest improvement of +0.13% over the baseline.

The findings suggest that downsampling operations in CNNs can lead to noise accumulation and disruption of basic object structures during inference. Incorporating wavelets in neural networks mitigates these drawbacks, contributing to faster convergence during training and ultimately resulting in improved classification accuracy.

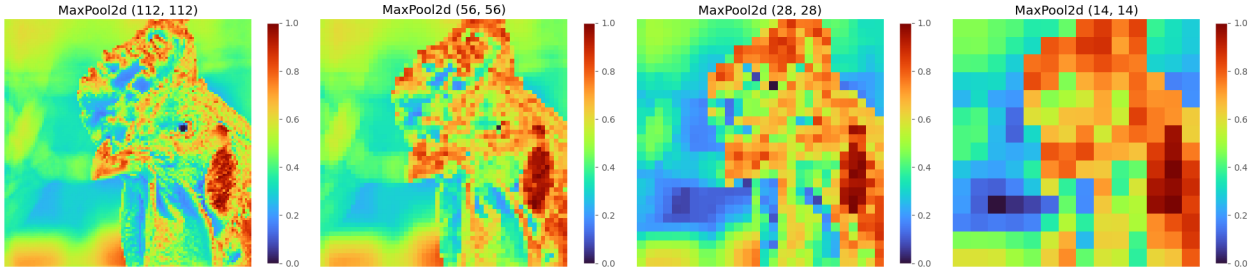
In a related context, prior work [27, 28] notes the increased classification accuracy of CNNs when filtering is integrated into downsampling. Additionally, [29] indicates that ImageNet-trained CNNs are biased towards recognizing textures rather than shapes. Experimental results align with the notion that commonly used downsampling operations may contribute to breaking object structures and accumulating noise in feature maps.

4.2 Visual Results

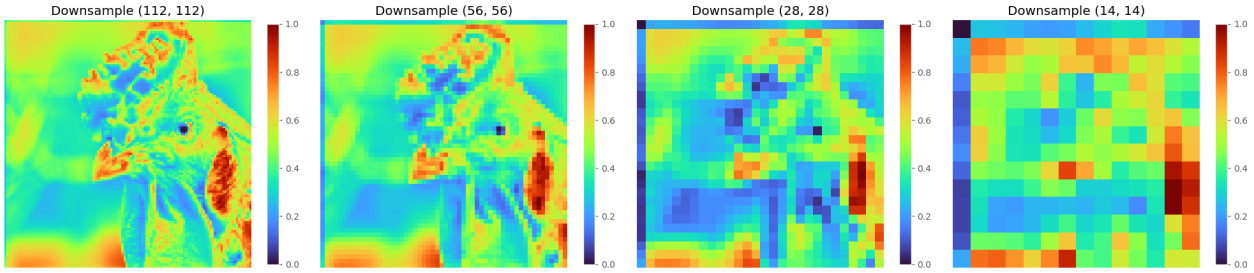
The following section delves into the network's internal processes by examining the `Downsampling` and `MaxPooling` layers introduced in listing 3 and section 2.3.1. Images from the test dataset were used, following our splitting. The analysis involves both the traditional and wavelet-based versions.

Now, the visual effect of filters from different networks on fig. 10 is examined.

Figure 10: Rooster (class 7 of ImageNet [4])



(a) `MaxPool` layers (from the vanilla version)



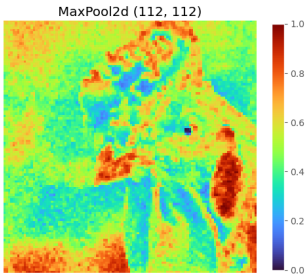
(b) `Downsample` layers (from the modified version)

Figure 11: Effect of filters for `VGG16bn` 3.2 with wavelet `bior2.2` (vanilla and wavelet-based).

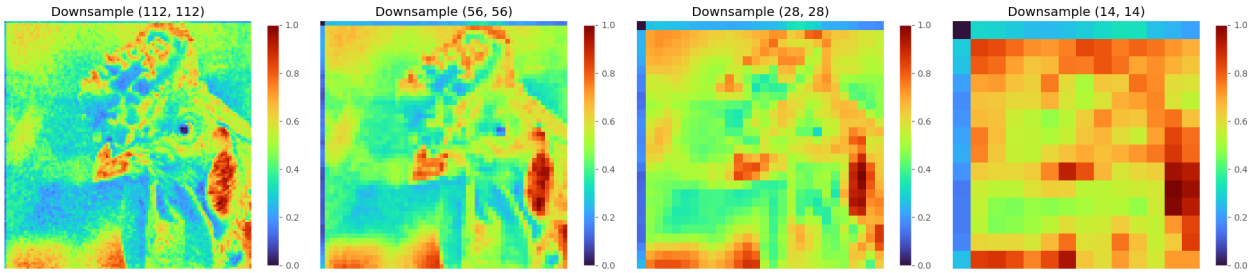
The feature maps presented in fig. 11 showcase a comparison between well-trained traditional CNNs and wavelet-based CNNs, as illustrated in fig. 10. Each line in the figure represents a different image processing approach (`MaxPool` or wavelet `Downsample`), while each column corresponds to a specific depth of the filter, diminishing as we progress deeper into the network.

The analysis of fig. 11 highlights notable differences in the feature maps generated by wavelet-based `VGG` compared to those from traditional CNNs. Particularly, the backgrounds in the wavelet-based feature maps exhibit a cleaner appearance. The most significant impact is observed in the representation of the rooster's head. In the second column (56x56), while the backgrounds appear relatively similar, the information pertaining to the rooster's head is notably smoother in the `Downsample` layer compared to the more noisy `MaxPool` layer. This difference becomes even more pronounced in the layer with a size of 28x28, where the rooster's head is considerably distorted in the pooling layer, while discernible features are retained in the wavelet layer. While this visual interpretation may lack scientific rigor, it provides a meaningful insight into why the new layers yield improved results.

Note that in this plot, we only have one `MaxPooling` layer (as done in `ResNet` implementations) but 4 `Downsample` layers. Indeed, the first `Downsample` layer actually corresponds to the pooling layer, but the



(a) `MaxPool` layers (from the vanilla version)



(b) `Downsample` layers (from the modified version)

Figure 12: Effect of filters for `ResNET101` 3.3 with wavelet `bior3.3` (vanilla and wavelet-based).

three others are the ones added before the convolutions on the downsampling steps already present in the architecture, following the strategy introduced in fig. 5.

The observations regarding the differences between pooling and downsampling are the same as with the `VGG` network. Indeed, the background and the face are clearer for the version with `Downsample`.

4.3 Noise Robustness

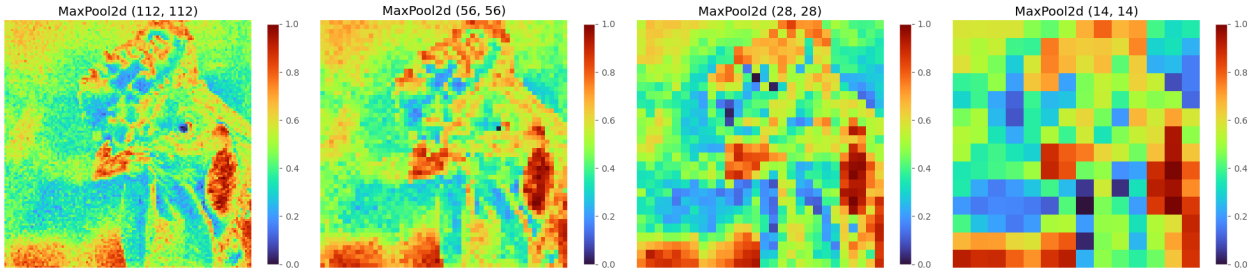
4.3.1 Visual Effect of the Noise Robustness

Given the inherent denoising capabilities of wavelets on signals [11, 30], it is reasonable to anticipate superior noise robustness in wavelet-based neural networks compared to their vanilla counterparts. To assess this, we introduced noise to the original rooster image (fig. 10), resulting in the noisy version shown in fig. 13.

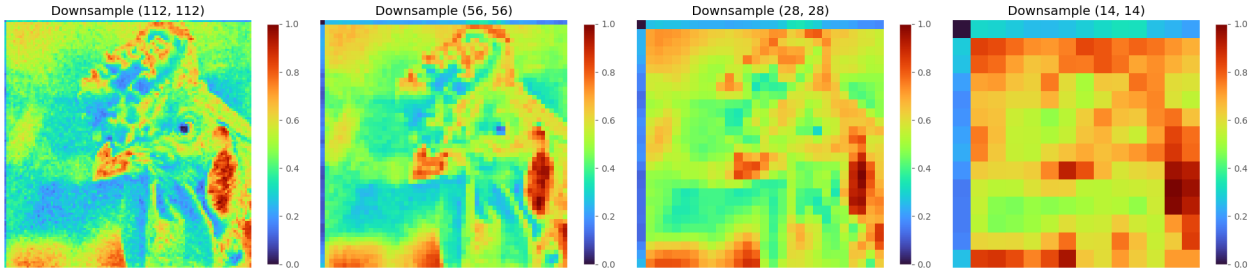
Figure 13: Noisy version of the rooster image from fig. 10 with added normal noise.

The objective is to evaluate the performance of the newly developed networks on this noisy image. For a comprehensive comparison, we employed the same networks as previously mentioned.

The observations presented in this section echo those discussed in section 4.2. Notably, the wavelet-based



(a) `MaxPool` layers (from the vanilla version)



(b) `Downsample` layers (from the modified version)

Figure 14: Effect of filters on a noisy image for `VGG16bn` 3.2 with wavelet `bior2.2` (vanilla and wavelet-based).

networks exhibit even greater efficiency in the presence of noise, as evident in fig. 14a. The noise noticeably perturbs the images in this setup, emphasizing the challenges posed by noisy conditions. In contrast, the `Downsample` layer demonstrates a remarkable denoising effect, as illustrated in fig. 14b.

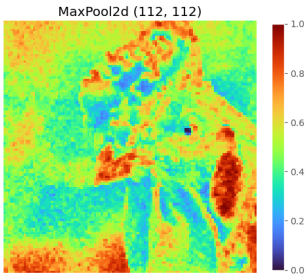
Specifically, in the first column (112x112), a subtle denoising effect is discernible. This effect becomes more pronounced in the second image (56x56), where noise significantly impacts the signal, leading to noticeable distortions that could impede accurate recognition. The contrast between the downsampled and maxpooled noisy-rooster images is most perceptible in the third figure (28x28). Here, the downsampled noisy-rooster appears smoother and less affected by the perturbing noise compared to its maxpooled counterpart. These findings underscore the efficacy of the `Downsample` layer in mitigating the impact of noise on image features, supporting the notion that wavelet-based networks excel in maintaining robustness, especially in challenging, noisy conditions.

The impact of the `Downsample` layer is particularly striking, as exemplified in fig. 15. Here, the noise exhibits a pronounced smoothing effect across different stages of downsampling. This enhanced denoising quality is even more remarkable when compared to the `VGGbn16` version, especially evident in the fourth image (14x14). The superior noise robustness observed in `ResNET101` is likely attributed to the choice of the wavelet used in the process.

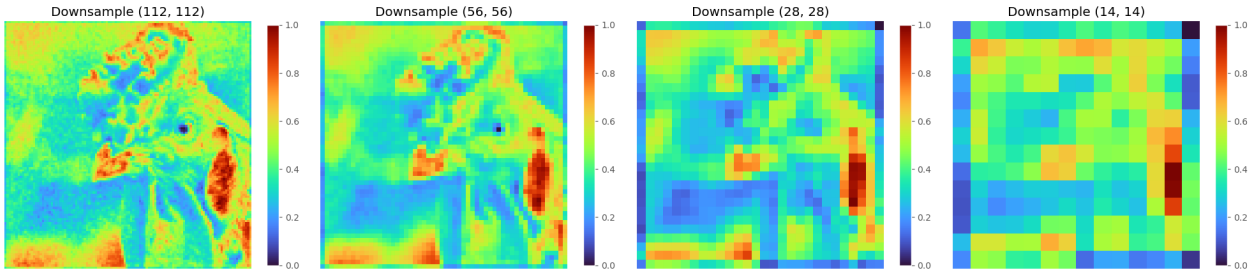
This examination serves to highlight how effectively the introduced networks preserve image features amid challenging, noisy conditions. The distinctive denoising characteristics of the `Downsample` layer underscore its crucial role in augmenting the noise robustness of wavelet-integrated deep neural network architectures.

4.3.2 Quantification of Noise Robustness

To quantitatively assess the noise robustness of the neural networks, we adopt a formula introduced in [31]. This formula serves as a benchmark to quantify noise robustness through the Corruption Error (CE_c^f) metric. The experiments were conducted using the ImageNet-C dataset, a collection of the ImageNet validation set corrupted with 15 visual distortions across various categories, including noise (Gaussian noise, shot noise, impulse noise), blur (defocus blur, frosted glass blur, motion blur, zoom blur), weather (snow, frost, fog, brightness), and digital (contrast, elastic, pixelate, JPEG compression).



(a) `MaxPool` layers (from the vanilla version)



(b) `Downsample` layers (from the modified version)

Figure 15: Effect of filters on a noisy image for `ResNET101` 3.3 with wavelet `bior3.3` (vanilla and wavelet-based)

The Corruption Error metric for a trained classifier f is computed as eq. (19):

$$\mathrm{CE}_c^f = \sum_{s=1}^{5} \mathrm{E}_{s,c}^f \Big/ \sum_{s=1}^{5} \mathrm{E}_{s,c}^{\mathrm{AlexNet}} \tag{19}$$

Here, $\mathrm{E}_{s,c}^f$ denotes the top-1 error of the trained classifier f for corruption type c at severity level s, and $\mathrm{E}_{s,c}^{\mathrm{AlexNet}}$ is the top-1 error of AlexNet for the same corruption type and severity level. This normalization accounts for variations in performance across different corruptions.

We introduce $mCEf_{\mathrm{noise}}$, representing the average Corruption Error across three noise types: Gaussian, shot, and impulse. The results reveal that the newly created wavelet-based networks, employing wavelets such as `dbx`, `chx,y` (also denoted `biorx,y`) and `haar` exhibit enhanced noise robustness for specific architectures.

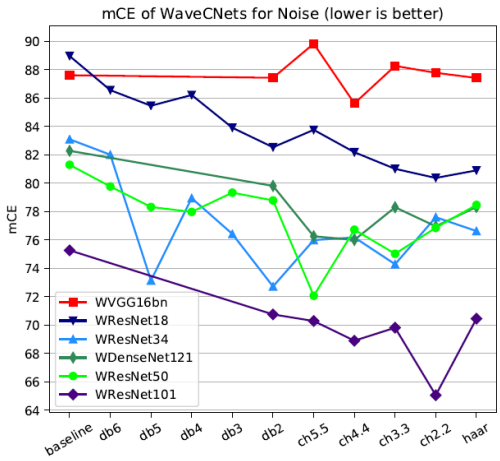


Figure 16: $mCEf_{\mathrm{noise}}$ of wavelet-based CNNs, *lower is better* (Source: [16]).

Feature map visualizations underscore the efficacy of wavelets in suppressing noise and preserving object

structures during inference, outperforming original CNNs. Nevertheless, variations in noise-robustness improvements are observed among different architectures. For instance, `VGG16bn` exhibits limited improvement, suggesting that the effectiveness of the proposed method is architecture-dependent.

Note: fig. 16 has been copied from [16] for the same reason as with table 1, since it helps the reader understanding the noise robustness of the different model but I didn't manage to train enough networks on enough wavelets to have satisfying results.

4.4 Using High Frequencies as Additional Information

In listing 3, we introduced a downsampling method utilized in training our noise-robust models. However, considering the wavelet's role in information extraction and compression [32, 33], we explore a novel avenue to enhance pooling layers. This involves not only retaining the low-frequency components, as done in traditional downsampling, but also preserving the high-frequency components, potentially contributing to improved classification. While our primary focus remains on noise robustness, incorporating high-frequency details may offer advantages in other aspects of image processing.

Two alternative downsampling functions are presented in listings 6 and 7.

```python
class DownsampleConcat(Module):
    def __init__(self, wavename="haar"):
        super(DownsampleConcat, self).__init__()
        self.dwt = DWT_2D(wavename=wavename)

    def forward(self, input):
        input_ll, input_lh, input_hl, input_hh = self.dwt(input)
        return torch.cat([input_ll, input_lh + input_hl + input_hh], dim=1)
```

Listing 6: Downsample function concatenating the outputs (both low and high frequency).

The listing 6 function concatenates the results of the Discrete Wavelet Transform (DWT), preserving all layers of the wavelet transform while introducing additional ones.

```python
class DownsampleSum(Module):
    def __init__(self, wavename="haar"):
        super(DownsampleSum, self).__init__()
        self.dwt = DWT_2D(wavename=wavename)

    def forward(self, input):
        input_ll, input_lh, input_hl, input_hh = self.dwt(input)
        return torch.sum(input_ll + input_lh + input_hl + input_hh, dim=[2, 3])
```

Listing 7: Downsample function which sums the outputs (both low and high frequency).

On the other hand, listing 6 averages the results of the DWT, avoiding the introduction of additional layers but potentially leading to information loss due to the non-invertibility of the averaging operation.

fig. 17 from [16] visually summarizes these alternative downsampling operations.

(a) Downsampling operation by **concatenating** the results of the wavelet transform (implemented in listing 6).

(b) Downsampling operation by **averaging** the results of the wavelet transform (implemented in listing 7).

Figure 17: Alternative downsampling strategies preserving high-frequency components in addition to low-frequency ones (adapted from [16]).

4.5 Other Methods

Alternative approaches leveraging wavelets and downsampling operations have been proposed in the literature, with notable mention of [34], as referenced in [16].

In [34], a novel methodology is introduced, combining the multi-level wavelet packet transform (WPT) with Convolutional Neural Networks (CNNs), resulting in a powerful framework known as Multi-level Wavelet-CNN (MWCNN). The technique involves applying the Discrete Wavelet Transform (DWT) followed by downsampling to obtain subband images. The biorthogonal property of DWT ensures precise reconstruction of the original image through Inverse DWT (IDWT).

Expanding upon this foundation, the method introduces multi-level WPT, wherein subband images undergo further processing with DWT in a recursive manner, generating decomposition results. Nonlinearities specific to particular tasks, such as image denoising and compression operations (e.g., soft-thresholding and quantization), are applied to enhance the overall architecture.

To refine WPT, [34] proposes MWCNN, integrating a CNN block between any two levels of DWTs. This architecture is illustrated in fig. 18, where subband images serve as inputs to a CNN block after each level of transform. MWCNN is positioned as a generalization of multi-level WPT, offering the flexibility to safely incorporate subsampling operations due to the biorthogonal property.

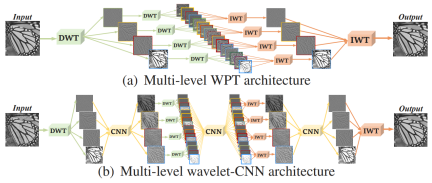


Figure 18: Transition from WPT to MWCNN. Intuitively, WPT can be seen as a special case of our MWCNN without CNN blocks (Source [34]).

The text hints at the advantages of MWCNN over conventional CNNs, emphasizing the frequency and location characteristics of DWT that contribute to preserving detailed textures during various image processing tasks.

MWCNN is positioned as a versatile framework, combining wavelet transforms and CNNs, adaptable to diverse image processing tasks. Future discussions are promised to delve into the connections between MWCNN and dilated filtering, as well as the nuances of subsampling.

5 Conclusion

5.1 Article Conclusion

In this project, we have explored the efficiency of wavelet integrated deep networks for image classification. By transforming the Discrete Wavelet Transform (DWT) and Inverse DWT (IDWT) into general network layers, we have designed wavelet integrated convolutional networks that can effectively preserve object structures and suppress data noise during network inference.

Our preliminary results have shown that this new type of neural network achieve higher image classification accuracy and better noise-robustness compared to commonly used network architectures. This highlights the potential of wavelet integration in deep learning for image classification tasks.

Moving forward, there are several avenues to explore in this field. We didn't discuss in detail of the different possibilities of using wavelets, such as merging the low and high frequencies or concatenating them, to further enhance the performance of wavelet integrated networks. Also on could use these networks for segmentation purposes, as discussed at the beginning.

Indeed, the potential applications of wavelet integrated deep networks extend beyond image classification. For example, in the field of medicine, wavelet integrated encoder-decoder networks have been used for neuron segmentation [35], while `WaveSNet` has been developed for image segmentation [36]. Additionally, wavelet neural networks have been applied in GPS/INS systems during GPS outages [37], and wavelet transform integrated generalized neural networks have been used for short-term load forecasting [38].

In conclusion, our project has shed light on the potential of wavelet integrated deep networks for image classification. Further research and exploration in this area can lead to advancements in various fields, including medicine, segmentation, and classification.

5.2 Personal Conclusion

In this project, we aimed to explore the efficiency of wavelet integrated deep networks for image classification. The first parts of introduction and setup were pretty easy since this was more a state-of-the-art research. However, the problems began after we presented all the theory, so between section 3 and section 4. Due to the large size of the dataset, it was not feasible to train so much model configurations on my own computer or even on platforms like Google Colab or Ensimag computers. Indeed, just one training took days. Despite the challenges faced during the development of the training script, I achieve good and promising results on the trained networks.

The partial results obtained demonstrate the potential of wavelet integrated deep networks in image classification tasks. The use of wavelets allows for efficient feature extraction and robustness to noise, which are crucial factors in real-world applications, as we talked in section 5.1 with medical segmentation for example.

It's a shame that I didn't have the time nor the resources to investigate more on the different possibilities of using wavelets, such as merging the low and high frequencies or concatenating them, to further enhance the performance of wavelet integrated networks. Also doing the segmentation would be fun.

Overall, this project has provided valuable insights into the potential of wavelet integrated deep networks for image classification. Despite the limitations encountered, I found it enriching and I saw the potential of wavelet in this field of deep learning which I'm interested in.

References

[1] Jianxin Wu. "Introduction to convolutional neural networks". In: *National Key Lab for Novel Software Technology. Nanjing University. China* 5.23 (2017), p. 495.

[2] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network". In: *2017 international conference on engineering and technology (ICET)*. Ieee. 2017, pp. 1–6.

[3] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[4] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[6] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[7] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1–9.

[8] Aharon Azulay and Yair Weiss. "Why do deep convolutional networks generalize so poorly to small image transformations?" In: *Journal of Machine Learning Research* 20.184 (2019), pp. 1–25.

[9] Fisher Yu and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions". In: *arXiv preprint arXiv:1511.07122* (2015).

[10] Nick Kingsbury. "Image processing with complex wavelets". In: *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences* 357.1760 (1999), pp. 2543–2560.

[11] Stéphane Mallat. *A wavelet tour of signal processing*. Elsevier, 1999.

[12] David S Taubman, Michael W Marcellin, and Majid Rabbani. "JPEG2000: Image compression fundamentals, standards and practice". In: *Journal of Electronic Imaging* 11.2 (2002), pp. 286–287.

[13] Stefan Pittner and Sagar V. Kamarthi. "Feature extraction from wavelet coefficients for pattern recognition tasks". In: *IEEE Transactions on pattern analysis and machine intelligence* 21.1 (1999), pp. 83–88.

[14] Lori Mann Bruce, Cliff H Koger, and Jiang Li. "Dimensionality reduction of hyperspectral data using discrete wavelet transform feature extraction". In: *IEEE Transactions on geoscience and remote sensing* 40.10 (2002), pp. 2331–2338.

[15] Björn Jawerth and Wim Sweldens. "An overview of wavelet based multiresolution analyses". In: *SIAM review* 36.3 (1994), pp. 377–412.

[16] Qiufu Li et al. *Wavelet Integrated CNNs for Noise-Robust Image Classification*. 2020. arXiv: 2005.03337 [cs.CV].

[17] Joan Bruna and Stéphane Mallat. "Invariant scattering convolution networks". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1872–1886.

[18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[19] Ingrid Daubechies. *Ten lectures on wavelets*. SIAM, 1992.

[20] Albert Cohen, Ingrid Daubechies, and J-C Feauveau. "Biorthogonal bases of compactly supported wavelets". In: *Communications on pure and applied mathematics* 45.5 (1992), pp. 485–560.

[21] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015.

[22] Nura Aljaafari. "Ichthyoplankton Classification Tool using Generative Adversarial Networks and Transfer Learning". PhD thesis. Feb. 2018. DOI: 10.25781/KAUST-K902H.

[23] Herbert Robbins and Sutton Monro. "A stochastic approximation method". In: *The annals of mathematical statistics* (1951), pp. 400–407.

[24] David R Cox. "The regression analysis of binary sequences". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 20.2 (1958), pp. 215–232.

[25] Navdeep Kumar et al. "Recent Advances in Bioimage Analysis Methods for Detecting Skeletal Deformities in Biomedical and Aquaculture Fish Species". In: *Biomolecules* 13.12 (2023). ISSN: 2218-273X. DOI: 10.3390/biom13121797. URL: https://www.mdpi.com/2218-273X/13/12/1797.

[26] Adam Paszke et al. "Automatic differentiation in pytorch". In: (2017).

[27] Richard Zhang. "Making convolutional networks shift-invariant again". In: *International conference on machine learning*. PMLR. 2019, pp. 7324–7334.

[28] Hubert Leterme et al. "On the Shift Invariance of Max Pooling Feature Maps in Convolutional Neural Networks". In: *arXiv preprint arXiv:2209.11740* (2022).

[29] Robert Geirhos et al. "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness". In: *arXiv preprint arXiv:1811.12231* (2018).

[30] Sylvain Sardy, Paul Tseng, and Andrew Bruce. "Robust wavelet denoising". In: *IEEE Transactions on Signal Processing* 49.6 (2001), pp. 1146–1152.

[31] Dan Hendrycks and Thomas Dietterich. "Benchmarking neural network robustness to common corruptions and perturbations". In: *arXiv preprint arXiv:1903.12261* (2019).

[32] John D Villasenor, Benjamin Belzer, and Judy Liao. "Wavelet filter evaluation for image compression". In: *IEEE Transactions on image processing* 4.8 (1995), pp. 1053–1060.

[33] G.G. Yen and K.-C. Lin. "Wavelet packet feature extraction for vibration monitoring". In: *IEEE Transactions on Industrial Electronics* 47.3 (2000), pp. 650–667. DOI: 10.1109/41.847906.

[34] Pengju Liu et al. "Multi-level wavelet-CNN for image restoration". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 773–782.

[35] Qiufu Li and Linlin Shen. "Neuron segmentation using 3D wavelet integrated encoder–decoder network". In: *Bioinformatics* 38.3 (2022), pp. 809–817.

[36] Qiufu Li and Linlin Shen. "Wavesnet: Wavelet integrated deep networks for image segmentation". In: *Chinese Conference on Pattern Recognition and Computer Vision (PRCV)*. Springer. 2022, pp. 325–337.

[37] Xiyuan Chen et al. "Novel hybrid of strong tracking Kalman filter and wavelet neural network for GPS/INS during GPS outages". In: *Measurement* 46.10 (2013), pp. 3847–3854. ISSN: 0263-2241. DOI: https://doi.org/10.1016/j.measurement.2013.07.016. URL: https://www.sciencedirect.com/science/article/pii/S0263224113003151.

[38] D.K. Chaturvedi, A.P. Sinha, and O.P. Malik. "Short term load forecast using fuzzy logic and wavelet transform integrated generalized neural network". In: *International Journal of Electrical Power Energy Systems* 67 (2015), pp. 230–237. ISSN: 0142-0615. DOI: https://doi.org/10.1016/j.ijepes.2014.11.027. URL: https://www.sciencedirect.com/science/article/pii/S0142061514007091.