UNIVERSIDAD AUTÓNOMA DE SAN LUIS POTOSÍ

FACULTAD DE CIENCIAS

Implementation and application of the wavelet transform into deep learning techniques for texture classification and object detection

TESIS

PARA OBTENER EL GRADO DE:

DOCTOR EN CIENCIAS APLICADAS

PRESENTA:

M.C. Juan Manuel Fortuna Cervantes

DIRECTOR(A) DE TESIS:

Dr. Marco Tulio Ramírez Torres

Dra. Marcela Mejía Carlos

SAN LUIS POTOSÍ, SLP AGOSTO 2022

Título:
Implementation and application of the wavelet transform into deep learning techniques for texture classification and object detection

Nombre del estudiante:
Juan Manuel Fortuna Cervantes

Comité que acepta la Tesis:

Dr. Marco Tulio Ramírez Torres (Asesor) _____

Dra. Marcela Mejía Carlos (Co-Asesor) _____

Dr. José Salomé Murguía Ibarra (Sinodal) _____

Dr. Carlos Soubervielle Montalvo (Sinodal) _____

Dr. Oscar Fernando Núñez Olvera (Sinodal) _____

Dr. José Martinez Carranza (Sinodal - externo) _____

AGOSTO-2022

Implementation and application of the wavelet transform into deep learning techniques for texture classification and object detection

Abstract:

Texture characterization in digital images has become an analysis tool in computer vision. Texture in visual perception is a very important physical property since it provides information about the structural composition of surfaces and objects in the image. This research involves two areas of knowledge, wavelet analysis and deep learning, both of which functioned as feature extraction methods for image processing with textures and materials. This work aimed to study the adaptability of deep learning with wavelet analysis and implement a detection and classification system for aerial navigation. The first approach analyzes the extracted information (spatial domain vs. wavelet domain) in object detection and aerial navigation. In addition, to evaluate the learning performance of a binary classifier. In the second approach, a multi-class classifier is proposed for the following databases: KTH-TIPS-2B (KT2B), Describable Textures Dataset (DTD), and Flickr Material Database (FMD). The possibility of merging both domains is evaluated since Convolutional Neural Networks (CNNs) do not learn spectral information, important information for texture recognition. In the third approach, a classification system for textured objects in aerial navigation tasks is implemented, where texture is involved as a physical property of the object. A classification model is developed using the knowledge transfer method and wavelet features. In the fourth approach, it is shown that internal pooling layers often lead to information loss. A classification system with a new pooling method called Discrete Wavelet Transform Pooling (DWTP) is proposed to solve this problem. The combination of these methods achieves acceptable classification performance. The learning plots reflect that all three datasets show learning generalization. In addition, the images obtained from the virtual environment show learning generalization for some classes in the DTD database. Moreover, the fusion of deep learning with wavelet analysis is recommended for small datasets of images with textures. Due to the limitation of learning about spectral information that is lost in conventional CNNs. Furthermore, it is argued that this helps to eliminate overfitting. The results show that it is possible to integrate this methodology into the technological development of applications, such as image classification or restoration tasks and object detection.

A mis padres
Maria y Juan Manuel

A mis hermanas
Elisa, Aurora y Delia.

Acknowledgment

A mi familia, por siempre darme la confianza de seguir superándome en cada paso de vida que doy. Gracias por apoyarme en esta etapa de vida profesional.

Al Dr. Marco Tulio Ramírez Torres por apoyarme en la realización de este proyecto de tesis. Aprecio el tiempo durante el cual me ha guiado en esta etapa de mi formación científica.

Agradezco a la Dra. Marcela Mejía Carlos por todo el apoyo, los consejos y sobre todo por los conocimientos que me ha brindado a lo largo de este trabajo.

Al Dr. Raúl Balderas Navarro por toda la disposición y el apoyo brindado durante su etapa de coordinador para seguir con mis estudios de posgrado, sus importantes comentarios y su compromiso con la ciencia.

Agradezco a mis profesores Dr. Salomé Murguía, Dr. Carlos Souberville, Dr. Oscar Nuñez y al Dr. José Martinez Carranza los cuales han compartido sus conocimientos conmigo, así como su impulso a comprender la ciencia y tecnología desde otro punto de vista.

A mis amigos de generación Lore, Beto y Balta, por todo el apoyo brindado, las conversaciones y discusiones, por todos los momentos divertidos y sobre todo por los sabios consejos que me han brindado.

A todo el personal del IICO por el apoyo, la confianza y las atenciones brindadas durante toda mi estancia en el instituto.

Al grupo de Drones (Arturo, Aldrich, Oyuki y Matus) del INAOE a cargo del Dr. José Martines Carranza por apoyarme y trasmitirme sus conocimientos en el área de las ciencias computacionales.

Expreso mi agradecimiento al Instituto de Investigación en Comunicación Óptica y al Consejo Nacional de Ciencia y Tecnología que a través del programa nacional de posgrados de calidad se me permitió realizar este trabajo de investigación, No. de becario 776118.

Contents

List of Figures

List of Tables

1

Introduction.

This chapter will give a brief overview of the development and implementation of the wavelet transform into deep learning techniques applied to texture classification and object detection in order to understand the importance of this research study and the distribution of the chapters of the work done.

Visual perception is a human ability that allows us to recognize objects around us, where the optical and nervous systems are involved. They are able to capture and process visual information to obtain a meaning, to be able to interpret and understand what is composed of the object. The necessity to have autonomous systems with the same visual capacity has allowed the development of these systems. Nowadays, interesting challenges can be found in the area of aerial robotics. Thus, some systems have integrated computer vision to obtain the information that describes the content of the image. Object detection is a problem for robots performing tasks in real scenarios and in real-time, given the lighting conditions, indeterminate orientations, object identity, shape, color, and texture. In addition, the information may differ in outdoor and indoor environments, thus varying the target information [6]. Therefore, image processing techniques such as wavelet analysis, deep neural networks, or Convolutional Neural Networks (CNNs) have become a compelling alternative [7, 8].

In image processing, texture can be defined from neighboring pixels and intensity distribution over the image. Besides, some classification methods for texture analysis include statistical, geometric, model, and spectral [9]. Spectral methods such as wavelet analysis describe the texture in the frequency domain. They are based on the decomposition of a signal in terms of basis functions and use the expansion coefficients as feature vector elements. Deep learning in the last decade has positioned itself as a new solution in the areas of robotics, computer vision, and natural language [10–12]. In particular, CNNs are a category of deep learning, as they are adapted to object analysis by learning and extracting complex features [13, 14]. Although CNNs are a universal extractor, in practice, it is unclear whether CNNs can learn to perform spectral analysis — a methodology that can provide better texture and material classification performance [9]. In this sense, a fusion of methods is needed to address this problem, combining both spatial and spectral approaches.

Some systems employ deep learning and wavelet analysis in visual processing. In image classification tasks, it has been proposed to convert images to the wavelet domain, thus obtaining temporal and frequency features [15]. In the field of image restoration, a multilevel wavelet CNN method was proposed to provide a balance between the size of the receptive area and computational efficiency [16]. The method is based on a U-Net architecture and the Inverse Wavelet Transform (IWT) for the reconstruction stage with high resolution. In the automatic coding of an image, the design of the CNN architecture has a significant weight. In this case, the designed network is a siamese CNN that receives merged information from infrared and visible images. The fusion is performed by multiscale decomposition of the image using wavelet analysis, and the reconstruction result is more perceptive to the human visual system [17]. Following the same approach, the work proposed in [18] presented two methods to highlight the edges of images in the classification

area. The first method decomposes the images and subsequently reconstructs them in a limited way. The second method, which develops enhanced images, introduces the local maximum wavelet coefficients. Both methods are applied before entering the CNN architecture. Regarding the texture classification in image processing applications [19–21], the authors propose an architecture to generalize spectral information lost in conventional CNNs. This information is beneficial for texture classification as it usually contains sufficient information about the shape of the object.

Following this, some proposals have also been made to decompose the feature maps through a wavelet pooling layer, achieving efficient processing comparable to that of the spatial domain. In [22], they proposed another alternative called wavelet pooling as a pooling layer within the CNN architecture. This method decomposes the image into two subbands, discarding the first level to reduce the size of the feature map. The approach allows for structured data compression, reducing the creation of denoised edges and other artifacts in the image. Moreover, it is found that existing pooling methods lose relevant information [23]. A method has been developed for vehicle-type classification tasks that combine CNN layers with compression and excitation modules and the Haar wavelet as a pooling layer [24]. According to this idea, the development of a method that contemplates multiple wavelet transforms has also been found because they work similarly to CNN filters [25].

In semantic segmentation tasks, encoder-decoder type networks have been used [26]. This type of CNNs usually uses pooling to reduce computational costs, improve invariance against certain distortions, and extend the receptive field. Therefore, a pooling method based on wavelet operations has been proposed to partition it into regions of interest. In [27], the authors presented an approach based on wavelets and deep learning for 3D neuron segmentation. In this case, the neuron segmentation method can completely extract the target neuron in noisy neuron images. A U-Net architecture based on wavelet transform pooling is proposed in [28]. This work aims to segment multiple sclerosis (MS) lesions in magnetic resonance imaging (MRI). One advantage is its multiresolution analysis; thus, its use improves the detection of lesions of different sizes and segmentation.

In recent years, Micro Aerial Vehicles (MAVs) applications have been studied and developed for object detection tasks [8, 29]. Several approaches use deep learning, giving excellent performance in the applications. In some autonomous navigation tasks for obstacle detection and avoidance, AlexNet (network) has been integrated to classify the images captured from the on-board camera of the drone [30]. The learning of this architecture is transferred from the ImageNet database [31]. Moreover, a detection system is presented for object detection and autonomous landing in [6]. It is about implementing the SSD7 (network) aboard the MAV. This SSD7 network is chosen for its fast performance on low-budget microcomputers without GPU.

The method proposed in [32] is an architecture called YOLO, which presents an essential performance in real-time image detection and processing. Moreover, in [33], the authors propose a deep learning approach to estimate the object center robustly due to varying illumination conditions, object geometry, and overlap in the image plane.

On the other hand, the importance of integrating wavelet analysis into deep learning is to improve the network's learning and find a new way to obtain necessary information for object recognition, whose main characteristic is repetitive patterns such as texture. In addition, implementing a classification system with both approaches improves detection efficiency. In this case, the classification and detection system is a good solution for integration into an aerial navigation system. Today, the capability and features of aerial navigation systems have been increased by using ROS — an operating system for robots, and in its development of Python scripts, OpenCV libraries, and deep learning algorithms.

This work proposes a general approach based on wavelet analysis as a method of spectral feature extraction. The first approach is a complete analysis of the learning behavior, using the information in the spatial and wavelet domains. In the second approach, a system is developed that combines wavelet analysis and deep learning to integrate it for texture classification. From this work, in the third approach, it is decided to incorporate transfer learning to improve classification performance. In the fourth approach, a new pooling method based on the wavelet transform is developed, allowing spectral analysis integration into the CNN architecture. The design of each process allows us to know and learn more about the characteristics of the objects. Moreover, the validation and implementation of a classification system and an aerial navigation system demonstrate the importance of the proposed approach.

- In this regard, the first approach is presented where the architecture design explicitly accepts that the inputs are images in the wavelet domain. This requirement implies the creation of a wavelet dataset with approximation and detail features — time-frequency information through multiresolution analysis. Using images in the wavelet domain improves the learning capability at the training stage, in contrast to the exclusive use of images in the spatial domain. Furthermore, it avoids overfitting in learning generalization when there is a small training dataset. A virtual navigation scenario has been proposed to validate the learning model, where the MAV or drone can recognize and learn the object with a repetitive pattern such as texture. The final basis of the proposal is a CNN with results in binary classification. In this way, the detection model predicts frame-by-frame and classifies the images captured from the on-board camera of the drone.

- From the analysis with spatial and wavelet domain data, the second approach focuses on classifying textures in images. The idea is to obtain structural information of surfaces, such as texture. The basis of the classification system is an approach to the Wavelet CNN architecture, which was proposed in [20] for texture classification and tasks on multiple labeling concerning image content. The implementation of this system is developed with the fusion of two approaches, using the spatial domain, specifically CNNs, and the spectral domain, the Haar wavelet transform [13, 14, 34]. Internally this system is divided into two stages: the first corresponds to feature extraction and the second to the classification stage. With respect to the feature extraction stage, the created tensor has a set of numerical parameters that describe the image content, such as color, texture, or shape of the object. Therefore, the feature extraction stage is vital for the overall success of any image classification and recognition system. A combination of both feature extraction methods, in particular, is shown to achieve accuracies competitive to those reported in the literature, with significantly fewer trainable parameters than using only one method. The system is validated with three datasets: KTH-TIPS-2B (KT2B), Describable Textures Dataset (DTD), and Flickr Material Database (FMD) [35–37]. As a result, the model is easier to train, its learning generalizes across the combination of information, and has a lower computational cost.

- Given the proposed architecture results, the third approach focuses on implementing a system for the classification (detect objects with textures). The goal is that the MAV performs aerial navigation (inside the virtual environment) in order for the classification system to recognize the object. This work focuses on prior information (in the data collected by the MAV) and structural recognition of the object (with a given texture) into a region of interest in the image plane. The system implementation is developed with the fusion of two approaches. The first is in the spatial domain, using transfer learning. The VGG16 architecture with the ImageNet database features is taken as a reference [31, 38]. The second approach focuses on the spectral domain, applying the Haar wavelet transform in two dimensions to obtain features across different scales [34]. VGG16 network has been selected for its fast performance and implementation with transfer learning and adaptability with wavelet analysis. In this case, the Describable Texture Database (DTD) is used to train the model, containing 47 texture classes, with 120 images per class [36]. On average, it has been tested with some textures for the classification task in the virtual environment. The prediction can be performed correctly, with an average processing rate of 2 fps.

- Despite the promising results obtained in texture classification, the architecture only fuses features that are lost with the spatial approach [19, 21]. Moreover, regularization methods are known to focus only on the *convolutional* layer. In contrast, the pooling layers' operations have not been updated [22]. In this sense, it has been decided to integrate wavelet analysis inside deep learning before merging the spatial and spectral approaches, i.e., to make it part of the learning process using the pooling method. Motivated by the above reasons, a classification system with a new pooling method called Discrete Wavelet Transform Pooling (DWTP) is proposed in the fourth approach. The pooling approach is based on the decomposition of the image into subbands. The method is implemented and developed using Python and Keras API with Tensorflow as Backend. Besides, the method is validated on three datasets: CIFAR-10, Describable Textures Dataset (DTD), and Flickr Material Database (FMD) [36, 37, 39]. The approach differs from traditional methods because it is not a subsampling methodology using neighboring regions, but rather wavelet pooling retains its role as a reduction layer. Wavelets allow localization in scale (i.e., frequency) and space. In other words, wavelets can be used to analyze local and spatial transients in data, such as edges or surfaces of an image [40]. Therefore, a complete evaluation of texture and material classification performance in images is presented. In addition, it preserve the most relevant information of textures and materials, which is sometimes lost with traditional methods such as Max-Pooling (MaxP) and Ave-Pooling (AveP). The idea is to evaluate the adaptability of deep learning with wavelet pooling.

The thesis has been organized as follows. Chapter 2 presents a brief review of the methodology and concepts required to develop this research. Chapter 3 presents a detailed description of the design and implementation of each proposed approach, the validation of the method, the discussion of the results obtained, and their application in object detection and image classification. Finally, Chapter 4 includes the conclusions, the analysis of the planned objectives, and the future work of the research.

2

Background.

This chapter introduces the essential elements to perform the proposed approaches using techniques such as Wavelet Analysis, Convolutional Neural Networks, and Transfer Learning to improve classification performance (object detection, textures and materials). Also, the concepts of pooling are discussed in-depth to integrate wavelet pooling.

Contents

2.1 Wavelet Analysis.

Wavelets represent functions as simpler, fixed building blocks at different scales and positions [15]. The one-dimensional wavelet transform can be easily extended to a Two-Dimensional Wavelet Transform (2D-WT), which is widely applied to two-dimensional signals such as images [41,42]. It has greatly impacted image processing tasks such as edge detection, image recognition, and image compression [9].

2D-WT considers a two-dimensional scale function $\Phi(x,y)$, and three two-dimensional wavelet functions $\Psi^H(x,y)$, $\Psi^V(x,y)$ and $\Psi^D(x,y)$, resulting in a lower resolution image than original, as well as detailed information on the horizontal (H), vertical (V) and diagonal (D) perspectives. Each function corresponds to the product of a function of scale φ and its corresponding wavelet ψ; in this way we have:

$$\Phi(x,y) = \varphi(x)\,\varphi(y) \tag{2.1}$$
$$\Psi^H(x,y) = \psi(x)\,\varphi(y) \tag{2.2}$$
$$\Psi^V(x,y) = \varphi(x)\,\psi(y) \tag{2.3}$$
$$\Psi^D(x,y) = \psi(x)\,\psi(y) \tag{2.4}$$

Scale and translation base functions are defined by:

$$\Phi_{j;m,n}(x,y) = 2^{\frac{j}{2}}\Phi\left(2^j x - m, 2^j y - n\right) \tag{2.5}$$
$$\Psi_{j;m,n}(x,y) = 2^{\frac{j}{2}}\Psi^d\left(2^j x - m, 2^j y - n\right) \tag{2.6}$$

where $j,m,n \in \mathbb{Z}$ and the superindex d assumes the values H, V and D to identify the directional wavelets given in equations (2.2)-(2.4). Considering that the equations (2.5)-(2.6) constitute an orthonormal basis for $\mathbf{L}^2\left(\mathbb{R}^2\right)$, the expansion of a finite energy function $f(x,y)$ is then defined as:

$$f(x,y) = \frac{1}{\sqrt{MN}}\sum_m\sum_n \mathrm{a}_{j0;m,n}\Phi_{j0;m,n}(x,y)$$

$$+\frac{1}{\sqrt{MN}}\sum_{d=H,V,D}\sum_{j=j0}\sum_{m}\sum_{n}\mathrm{d}^{d}_{j;m,n}\Psi^{d}_{j;,m,n}\left(x,y\right) \quad (2.7)$$

where the scale coefficients $\mathrm{a}_{j;m,n}$ and wavelet $\mathrm{d}^{d}_{j;m,n}$ are defined by:

$$a_{j;m,n}=\iint f\left(x,y\right),\Phi_{j;m,n}\left(x,y\right)dxdy \quad (2.8)$$

$$d^{d}_{j;m.n}=\iint f\left(x\right),\Psi^{d}_{j;m,n}\left(x,y\right)dxdy \quad (2.9)$$

Expressions (2.7) and (2.8)-(2.9) represent the equations of synthesis and analysis of the original image, and together they constitute the Two-Dimensional Discrete Wavelet Transform (2D-DWT) [1].

2.1.1 Multiresolution Analysis

The Mallat-Multiresolution Analysis (M-MA) algorithm makes it possible to calculate the coefficients numerically $a_{j;m,n}$ and $d_{j;m,n}$ of the two-dimensional functions or image (denoted by $x\left[m,n\right]$), with a theoretical basis of the Fast Two-Dimensional Wavelet Transform (2D-FWT) [43]. Also, the algorithm provides a connection between the wavelets and the filter banks [44]. The multiresolution decomposition of an image is represented by a series of approximations and details in sub-images. In the first level of decomposition, two filters are applied respectively, one low-pass (h) and one high-pass (g), each followed by a subsampling operation by a factor of 2, as illustrated in Figure 2.1.

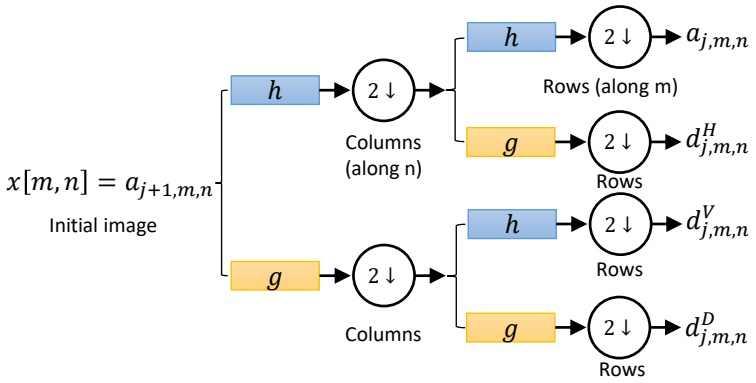


Figure 2.1: The first level of decomposition applied to an image using the filter bank [1].

The result of applying three levels of wavelet decomposition to an image ($x[m, n]$), sized $M \times N$, is illustrated in Figure 2.2. After the two-dimensional signal passes through the filter bank structure shown in Figure 2.1, four sub-images with $M/2$ rows and $N/2$ columns are obtained; i.e., each one of the four sub-images has a quarter of the pixels of the input image. The approximation sub-image is achieved by the approximation calculations along the rows of the original image, followed by the approximation calculations across the columns. This sub-image is an average version of the image ($x[m, n]$), with a quarter resolution and statistical properties similar to the original signal [42]. The rest of the sub-images show specific characteristics of the original image in a particular direction, providing the detail coefficients: horizontal, vertical, and diagonal. The same wavelet transform is applied only to the approximation sub-image to determine the next level of decomposition. Therefore, we get four sub-images but now with dimensions of $M/2^2$ rows and $N/2^2$ columns. This iteration is repeated until the desired resolution level or until the level allowed by the image's dimensions [1].

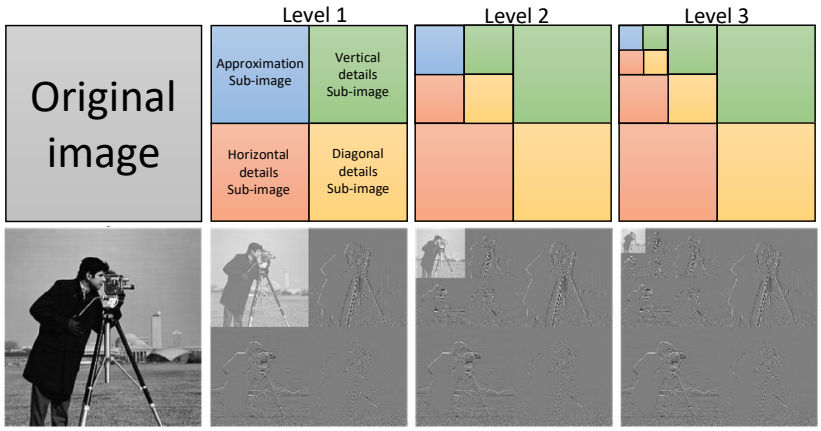


Figure 2.2: Decomposition process applying three levels of the filter bank, which results are some approximation and detail sub-images.

In general, multiresolution decomposition for a two-dimensional signal reveals differences in resolution levels. It shows details in different orientations, properties that indicate that the 2D-DWT is well suited for detecting important information from the original two-dimensional signal or image.

The Haar wavelet is the most straightforward and is one of the fundamental examples to illustrate the wavelet theory better. It is widely used when there are signals with abrupt changes. It has dramatically impacted image processing tasks such as edge detection, image recognition, and image compression [9].

The concept of the wavelet transform is easy to generalize to two-dimensional functions such as images. This can be done in several ways. Only two are mentioned here: the standard decomposition and the non-standard or pyramidal decomposition. The process with the bank filter describes the non-standard decomposition. It is the basis of this research work. Because most of the existing techniques for image management are usually complex, difficult to implement, and have a high computational cost, recently, wavelet methods have been applied to different stages of image processing because they are very *efficient* computationally and *easy* to implement numerically [7].

2.1.2 2D Discrete Wavelet Transform

Given an image x, it is possible to use 2D-Discrete Wavelet Transform (2D-DWT) with four convolution filters, i.e., low-pass filter f_{LL} and high-pass filters f_{LH}, f_{HL}, and f_{HH}, to decompose x into four sub-images, i.e., x_{LL}, x_{LH}, x_{HL}, and x_{HH} [34]. Note that the four filters have fixed parameters with convolutional stride 2 during the transformation [16, 24]. Taking the Haar wavelet as an example, these filters are defined in Equation (2.10).

$$f_{LL} = \begin{bmatrix} +1 & +1 \\ +1 & +1 \end{bmatrix}, f_{LH} = \begin{bmatrix} -1 & -1 \\ +1 & +1 \end{bmatrix}, f_{HL} = \begin{bmatrix} -1 & +1 \\ -1 & +1 \end{bmatrix}, f_{HH} = \begin{bmatrix} +1 & -1 \\ -1 & +1 \end{bmatrix} \tag{2.10}$$

Moreover, the operation of DWT is defined in Equation (2.11):

$$\begin{aligned} x_{LL} &= (f_{LL} \otimes x) \downarrow_2, \quad x_{LH} = (f_{LH} \otimes x) \downarrow_2 \\ x_{HL} &= (f_{HL} \otimes x) \downarrow_2, \quad x_{HH} = (f_{HH} \otimes x) \downarrow_2 \end{aligned} \tag{2.11}$$

where $\otimes$ denotes convolution operator, and $\downarrow_2$ means the standard downsampling operator with factor 2. In other words, 2D-DWT mathematically involves four fixed convolution filters with stride 2 to implement the downsampling operator. Moreover, according to the theory of Haar transform [34], the (i, j)th value of x_{LL}, x_{LH}, x_{HL}, and x_{HH} can be written in Equation (2.12).

$$\begin{aligned} x_{LL}(i,j) &= x(2i-1, 2j-1) + x(2i-1, 2j) + x(2i, 2j-1) + x(2i, 2j) \\ x_{LH}(i,j) &= -x(2i-1, 2j-1) - x(2i-1, 2j) + x(2i, 2j-1) + x(2i, 2j) \\ x_{HL}(i,j) &= -x(2i-1, 2j-1) + x(2i-1, 2j) - x(2i, 2j-1) + x(2i, 2j) \\ x_{HH}(i,j) &= x(2i-1, 2j-1) - x(2i-1, 2j) - x(2i, 2j-1) + x(2i, 2j) \end{aligned} \tag{2.12}$$

Given that the derivability of the Haar transform is a good property for end-to-end backpropagation [45], Haar wavelet decomposition is used as a pooling layer in the proposed structure.

2.2 Convolutional Neural Networks

Artificial Neural Networks (ANNs) are a class of machine learning algorithms that learn from data. Moreover, the algorithm is inspired by the structure and function of the brain. For example, deep learning is in this area. In other words, deep learning methods transform information into multiple levels of representation obtained by composing nonlinear but straightforward modules [46].

The history of neural networks and deep learning began in the 1940s. McCulloch and Pitts proposed the first neural network in 1943 [47]. Capable of recognizing two different classes called perceptron. However, the process was performed manually to obtain the correct weights for each category. Therefore, Rosenblat [48, 49] in 1950 developed an algorithm that automatically learned the weights required for the classifier. On the other hand, Minsky and Papert in 1969 [50] showed that a perceptron with a linear activation function could not solve nonlinear problems, particularly on the XOR function dataset problem. Moreover, within the third winter stage of deep learning, the backpropagation algorithm and the research of Werbos, Rumelhart, and LeCun recently succeeded in resurrecting neural networks. The proposed algorithm made it possible to train multilayer neural networks combined with nonlinear activation functions, learning nonlinear functions, and the XOR problem. Today, we can train networks with many more hidden layers that can obtain hierarchical learning. In which simple features are learned in the lower layers and more complex attributes in the higher layers of the network.

On this idea of hierarchical learning, the Convolutional Neural Network (CNN) proposed by LeCun in 1988 could recognize handwritten characters automatically using filters [51]. In many applications, CNNs are considered the most powerful image classifier and today are responsible for driving the state-of-the-art in the subfields of computer vision [46]. A CNN comprises three main layers: convolution, pooling, and fully connected [24].

Convolutional Neural Networks are a type of neural network capable of processing time series and images. The name *convolutional* neural network indicates that the network uses a mathematical operation called convolution, which is a linear operation. Indeed, neural networks use convolution instead of general matrix multiplication inside their layers [14]. Moreover, an initial condition of CNNs is that it uses a spatial structure. This spatial relationship is inherited from layer to layer, and each feature is based on a small local spatial region. It is essential to maintain this condition because the convolution operation and transformation critically depend on these relationships [2]. The design of CNNs is divided into feature extraction and a classification stage. The feature extraction stage is initially

composed of three layers: *convolution*, *pooling*, and *ReLU* function. Each layer is a three-dimensional (tensor) structure with a *height*, *width*, and *depth*. Inside the classification stage, the set of layers is usually fully connected and mapped in a specific way to the set of output nodes.

The input data of the CNN usually have a two-dimensional structure. The point values of each section of the structure are called pixels. In particular, each pixel represents a spatial location inside the image. On the other hand, the input layer (qth) receives a three-dimensional structure (*height* L_q, *width* B_q, and *color_ depth* d_q), as two dimensions represent the spatial relationships and a third one represents the independent properties of the color channels, as shown in Figure 2.3a. In contrast, in the hidden layers, these independent properties correspond to patterns extracted from local regions of the image. These new regions are called *feature maps* or *activation maps* [2].

An essential element is the generation of the learning weights, which depends on the filter set. For example, Figure 2.3a shows the application of a filter with size 5. It should be emphasized that the value of F_q is small and odd and that they are commonly used with sizes 3, 5, or 7. Besides, the filter and the image define the height and spatial width of the next hidden layer. In other words, by performing convolutions on the qth layer, one can align the filter at $L_{q+1} = (L_q - F_q + 1)$ along the height and $B_{q+1} = (B_q - F_q + 1)$ along the width of the image. Therefore, the greater the number of filters, the greater the number of feature maps. Also, the number of filters used in each layer controls the number of parameters.

Note that each filter will identify one type of spatial pattern, so many filters are required to capture a wide variety of patterns. For example, the filter shown in Figure 2.3b represents a horizontal edge detector in a grayscale image. In this case, the resulting feature has a high activation at every position where a horizontal edge is seen. However, a vertical edge will give zero activation, while a slanted edge might give intermediate activation. About the convolution operation from the qth layer to the $(q+1)$th layer, it can be defined as follows:

$$h_{ijp}^{(q+1)} = \sum_{r=1}^{F_q} \sum_{s=1}^{F_q} \sum_{k=1}^{d_q} w_{rsk}^{(p,q)} h_{i+r-1,j+s-1,k}^{(q)} \quad \begin{array}{r} \forall i \in \{1..., L_q - F_q + 1\} \\ \forall j \in \{1..., B_q - F_q + 1\} \\ \forall p \in \{1..., d_{q+1}\} \end{array} \tag{2.13}$$

where the filter pth of layer qth has parameters denoted by the three-dimensional tensor $W^{(p,q)} = [w_{ijk}^{(p,q)}]$, the indices i, j, k denote the positions along the height, width, and depth of the filter. The feature maps of layer qth are represented by the three-dimensional tensor $H^{(q)} = [h_{ijk}^{(q)}]$. The underlying convolutional operation is a simple dot product over the filter volume, repeated over all valid spatial positions (i, j) and filters (indexed by p). Figure 2.4 shows two examples of convolution

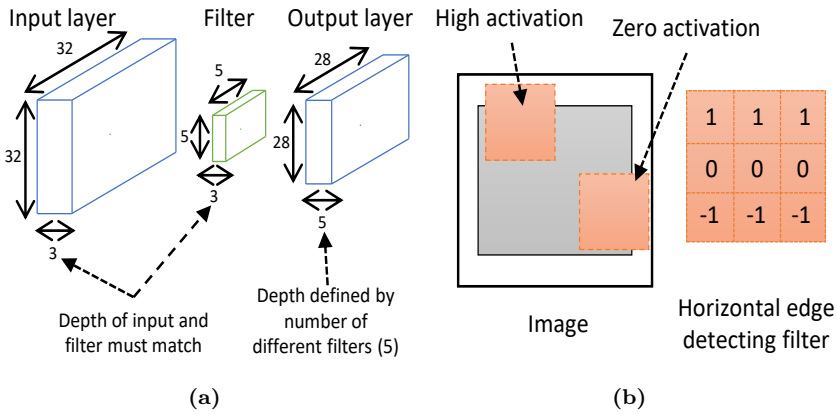


Figure 2.3: (a) Convolution between an input layer of size 32×32×3 and a filter of size 5×5×3. (b) Sliding a filter around the image is tried to look for a particular feature [2].

operations with a 7×7×1 input layer and a filter of size 3. In this case, the depth of the layer must match that of the filter or kernel. Besides, it is essential to mention that the contributions from the overall feature maps of the scalar product will result in a single output feature value in the next layer. Also, the activation map of the next layer is shown in the upper right part of Figure 2.4.

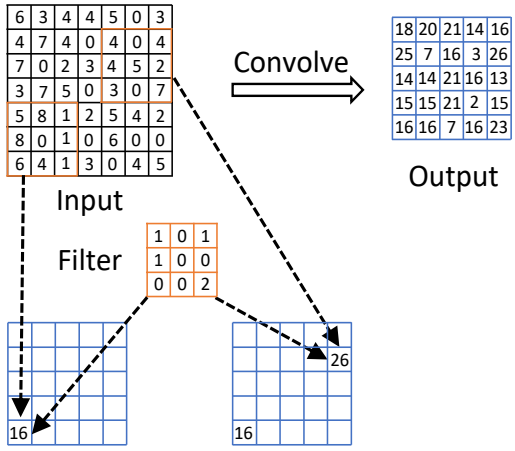


Figure 2.4: Convolution between a 7×7×1 input and a 3×3×1 filter with an interval of 1. Each filter will create a single feature map, regardless of its depth [2].

Hence, the number and order of layers considered for architecture design can improve feature extraction and thus better classification. However, the pooling layer, where dimensionality reduction happens, can also improve efficiency. In the next section, some popular pooling methods are described.

2.2.1 Pooling Method

Some authors describe the pooling method as a subsampling methodology [2, 14, 23, 52]. The pooling method also transforms the activation map into a new feature map. The pooling operation works on small regions of size $P_q \times P_q$ on the qth layer, usually after each convolutional layer. The pooling method has two main purposes. The first is to reduce the number of parameters and, thus, reduce the computational cost. The second is to control overfitting [22, 28]. The expectation is that an ideal pooling method extracts only useful information and discards irrelevant details [24]. Also, the pooling method does not change the number of activation maps. In other words, the depth of the layer created keeps the same dimension as the layer on which the pooling operation has been carried out [2, 16].

Figure 2.5 shows examples of pooling with strides 1 and 2. In this case, pooling over 3×3 regions is used. On the other hand, by setting pooling (3×3) with a stride of 2, pooling can reduce the size of the activation maps. Therefore, it is pretty common to use this type of configuration. Additionally, when using an array of 2 (pooling 2×2), the result between the different regions that are pooled would have no overlapping.

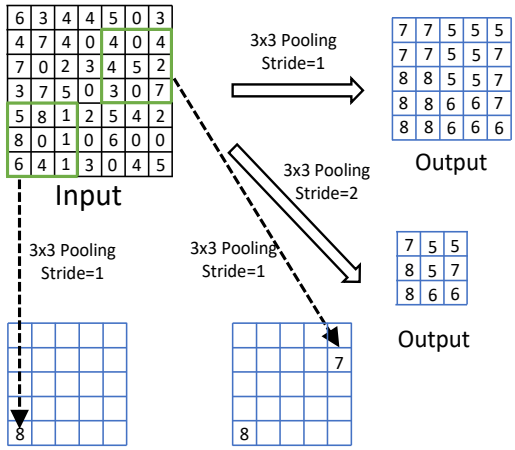


Figure 2.5: The Max-Pooling operation to an activation map of size 7×7 with strides of 1 and 2. Unlike convolution, each activation map is processed independently. Therefore, the number of output activation maps is exactly equal to the number of input activation maps [2].

The pooling method has two main purposes. The first is to reduce the number of parameters and thus reduce the computational cost. The second is to control overfitting [22, 28]. Besides, reducing the image size makes it possible for the neural network to tolerate small changes in the input image. The expectation is that an ideal pooling method extracts only useful information and discards irrelevant details [24]. In general, pooling takes two forms that are most commonly used: Max-Pooling (MaxP) and Ave-Pooling (AveP) [22, 52–57].

- Ave-Pooling: an Average Pooling layer performs top-down sampling by dividing the input into rectangular regions and calculating the mean values of each region. It was first introduced by LeCun *et al.* [53] and used in the first convolution-based neural network [54]. The average pooling function is expressed as:

$$a_{kij} = \frac{1}{|R_{ij}|} \sum_{(p,q)\in R_{ij}} a_{kpq} \tag{2.14}$$

  where a_{kij} is the output activation of the k^{th} feature map at (i, j), a_{kpq} is the input activation at (p, q) within R_{ij}, and $|R_{ij}|$ is the size of the pooling region.

- Max-Pooling: a Max-Pooling operator [55] can be applied to downsample the convolutional output bands, therefore reducing variability. The max-pooling operator pulls the maximum value inside each rectangular region. The Max-pooling function is expressed as:

$$a_{kij} = max_{(p,q)\in R_{ij}}(a_{kpq}) \tag{2.15}$$

  An example of these two pooling methods is shown in Figure 2.6.

These forms of pooling are deterministic, efficient, and simple but have shortcomings that hinder the learning potential of CNN. Depending on the data, Max-Pooling can erase details from an image [22]. Hence, this happens if important details have less intensity than insignificant details. Moreover, it generates noise accumulation, and it is not possible to restore lost information [27]. Max-Pooling is sensitive to overfitting the dataset used for training and hinders generalization [58]. Average pooling, depending on the data, can dilute the relevant details of an image. Averaging data with values far below important details cause this action [22]. Figure 2.6 illustrates these shortcomings with the example of a toy image.

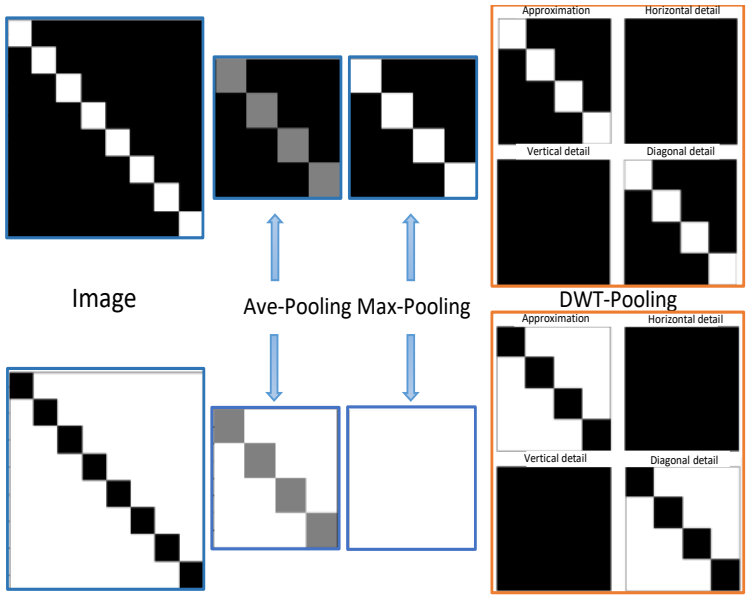


Figure 2.6: Example of the shortcoming of Max and Average Pooling against the contribution of wavelet pooling, preserving the essential features.

Suppose DWT filters are considered convolutional filters with predefined weights. In that case, it is observed that DWT is a particular case of Fully Connected Layers (FCN) without the layers of nonlinearity. The original image can be decomposed by DWT and then reconstructed exactly by the DWT inverse without losing information [16]. On the other hand, the wavelet theory opens the possibility to represent the image details inside learning CNNs, thanks to the frequency and location features generated by the wavelet transform (see Figure 2.6).

2.2.2 Transfer Learning

A common and highly effective approach to deep learning on small image datasets is to use a pre-trained network. In this case, a large CNN trained on the ImageNet dataset (1.4 million labeled images and 1,000 different classes) is considered [31]. Furthermore, feature extraction involves using representations that have already been obtained through previous training in the architecture. The purpose is to extract relevant characteristics from new examples. These characteristics are used in a new classifier trained from scratch.

A previous section mentioned that the CNN is composed of two parts (feature extraction and classification stage) for image classification tasks. The first part is called the convolutional-base. In this case, feature extraction consists of taking the convolutional-base part of a previously trained CNN, running the new data through it, and training a classifier on top of the output, see Figure 2.7.

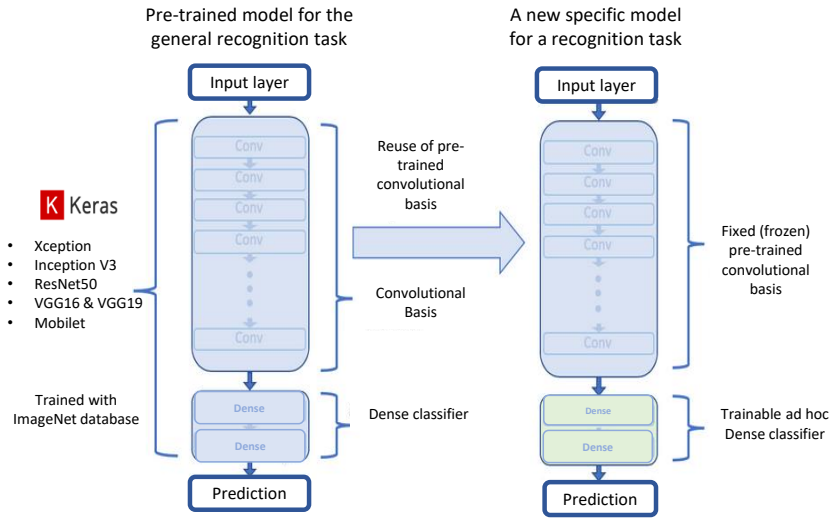


Figure 2.7: Swapping classifiers while keeping the same convolutional base [3]

The feature maps of a convolutional-base are generic concept maps over an image, valid for any given computational vision problem. On the other hand, densely connected features are largely useless for problems where object location is essential [4].

The level of the representations extracted by convolutional layers depends on the depth of the layer in the model. Layers earlier in the model extract local and generic feature maps (such as visual edges, colors, and textures), while layers higher in the model extract more abstract concepts. Therefore, if a new dataset differs significantly from the dataset on which the original model was trained, it may be better to use only the first few layers of the model to feature extraction over the complete convolutional-base. In this case, the ImageNet dataset includes multiple classes.

It is beneficial to reuse the information contained in the densely connected layers of the original model. However, it is decided not to do so in order to solve the more general case in which the set of classes of the new problem is correlated with the set of classes of the original model. This is implemented by using the convolutional-base of the network (selected in each case study), trained with ImageNet, to extract interesting features from the images and then train a classifier (objects with textures,

etc.) on these features. The VGG16 model, among others, comes pre-packaged with Keras. It can be imported from the `keras.applications` module. These models can be used for prediction, feature extraction, and fine-tuning. The list of image classification models (all pre-trained on the ImageNet dataset) is available as part of `keras.applications`:

- Xception [59].
- Inception V3 [60].
- ResNet50 [61].
- VGG16 and VGG19 [38].
- MobileNet [62].

2.2.3 Evaluating Deep Learning Models

In deep learning, the goal is to achieve models that generalize-that perform well on never-seen data—and overfitting is the main obstacle. You can only control that which you can observe, so it is crucial to be able to reliably measure the generalization power of the model. The following sections look at strategies for mitigating overfitting and maximizing generalization. In addition, to measuring generalization: how to evaluate machine learning models [4].

2.2.3.1 Training, Validation, and Test Sets

The four approaches presented in Chapter 3 divided the data into a training set, a validation set, and a test set. The reason for not evaluating the models with the same data with which they were trained is so that the three models do not overfit. That is, their performance on the unseen data may stagnate (or get worse) compared to their performance on the training data, which always improves as training progresses.

A good practice is to split our dataset using the Hold-Out Cross-Validation sampling technique [4]. The technique tests the model's predictive performance and how well it performs on the test or unseen data. The dataset is initially separated into two sets: training and test; then, the training set is split into two subsets: training and validation. The idea is that each set contains representative images of each class. Therefore, it is achieved to have balanced and random sets. Typically,

one uses about 80% of the training data for training and 20% for validation [14]. Schematically, Hold-Out validation looks like Figure 2.8.

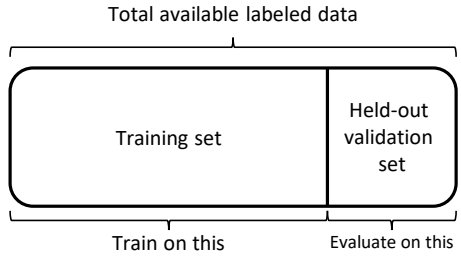


Figure 2.8: Simple Hold-Out validation split [4]

2.2.3.2 Compile and Train Model

To make the network ready for training, it is needed to pick three things as part of the *compilation* step [4]:

- A *loss function*–How the network will be able to measure its performance on the training data and thus how it will steer itself in the right direction, also called the *objective function*.
- An *optimizer*—The mechanism through which the network will update itself based on the data it sees and its loss function. It implements a specific variant of Stochastic Gradient Descent (SGD) [63].
- *Metrics to monitor during training and testing*—Here, it will only care about accuracy (the fraction of the images that were correctly classified).

In order to compile the network with Keras, the `compile()` function is used.

Listing 2.1: Keras Code.

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Training a CNN means finding the best set of weights, which map the inputs (*images*) to the outputs (*labels*) in the training dataset and, at the same time, in the validation dataset. Training is processed over epochs. An epoch is an iteration through all samples of the training dataset. Moreover, it is common for an epoch to be split into minibatches. Each minibatch consists of one or more samples. After each batch iteration, the weights of the network will be updated. In order to train the network with Keras, we use the `fit()` function.

2.2.3.3 Gradient Descent in Keras

The fundamental trick in deep learning is to use a *Loss score* as a feedback signal to adjust the value of the weights a little in a direction that will lower the loss score for the current example. This adjustment is the job of the optimizer, which implements what's called the Backpropagation algorithm: the central algorithm in deep learning.

Gradient descent is an algorithm that uses calculus concept of gradient to try and reach local or global minima. It works by taking the negative of the gradient in a point of a given function, and updating that point repeatedly using the calculated negative gradient, until the algorithm reaches a local or global minimum, which will cause future iterations of the algorithm to return values that are equal or too close to the current point, as shown in Figure 2.9. It is widely used in machine learning applications [5].

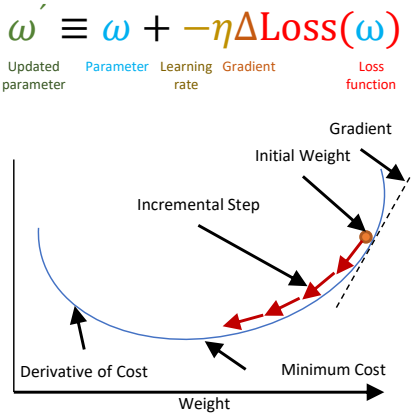


Figure 2.9: What is Gradient Descent? [5]

Also, there is Stochastic Gradient Descent (SGD) which iteratively updates a set of parameters to minimize an error function [63]. While in GD, you have to run through all the samples in your training set to do a single update for a parameter in a particular iteration, in SGD, on the other hand, you use only one or a subset of training samples from your training set to do the update for a parameter in a particular iteration. If you use a subset, it is called Minibatch Stochastic Gradient Descent [64].

In Minibatch Gradient Descent, the batch size must be between 1 and the size of the training dataset. As a result, we get k batches. The weights of the neural network are updated after each mini-batch iteration. The `batch_size` argument is

set to a number, which is more than 1 and less that the size of the training dataset. For example, we can use a batch size of 64:

Listing 2.2: Keras Code.

```
model.fit(X_train, y_train, epochs=150, batch_size=64)
```

Additionally, multiple variants of SGD differ by taking into account previous weight updates when computing the next weight update, rather than just looking at the current value of the gradients. There is, for instance, SGD with momentum, Adam, RMSProp, and several others [65, 66].

2.2.3.4 Evaluation Metrics

To quantitatively evaluate the classification model based on the combination of deep neural networks with pooling methods, this paper adopts the metrics *Accuracy*, *Recall*, *Precision*, *F1-score*, and the *confusion matrix* to evaluate the classification index [67]. *Accuracy* measures the percentage of cases that the model predicted correctly. In this case, it functions well because the classes are correctly balanced. The indicator is expressed by equation (2.16). Also, the quality of the classification model can be measured with the *Precision* metric, equation (2.17). The *Recall* metric will tell us how many cases the classification model can identify, equation (2.18). The *F1* value is used to combine the *Accuracy* and *Recall* metrics into a single value, equation (2.19). The *F1-score* value allows us to compare the performance of the combined *Precision* and the *Recall* model among several solutions.

$$Acc = \frac{T_P}{Total\ number\ of\ images} \tag{2.16}$$

$$P = \frac{T_P}{T_P + F_P} \tag{2.17}$$

$$R = \frac{T_P}{T_P + F_N} \tag{2.18}$$

$$F1 = 2 * \frac{P * R}{P + R} \tag{2.19}$$

where T_P is the number of positive samples correctly predicted, and F_P is the number of samples where negative samples are predicted as positive. F_N is the number of positive samples that are predicted as negative samples. The Scikit learn library provides us with a classification report to evaluate the quality of the predictions of a classification algorithm. The method shows us the main classification metrics (`classification_report`).

3

Proposed approach.

Throughout this chapter, the approaches will be described, and the application of each method. The first approach is a complete analysis of the learning behavior, using the information in the spatial or wavelet domains. In the second approach, a system is developed that combines wavelet analysis and deep learning to integrate it for texture classification. From this work, in the third approach, it is decided to incorporate transfer learning to improve classification performance. In the fourth approach, a new pooling method based on the wavelet transform is developed, allowing spectral analysis integration into the CNN architecture.

Contents

In order to provide the context in which this chapter is developed, a brief description of the general problem statement, which is divided into four approaches, is described below. In addition, the advantages and disadvantages of each experiment.

The first approach assumes that object detection is a problem for robots performing tasks in real-world, real-time scenarios, given lighting conditions, indeterminate orientations, object identity, shape, color, and texture. In addition, the information may differ in outdoor and indoor environments, thus varying the target information. Then, as the sensing task becomes more robust, its description capability is limited to recognizing textures as a physical property of the object. Therefore, the main objective of this research is to develop a methodology for object detection and texture classification, using the wavelet transform, applied to the processing of real-time images captured from the on-board camera of the UAV.

The first approach focuses on developing most of the specific objectives:

- To establish a diagram between the ROS operating system and the flight control of the Parrot UAV.
- Identify the factors that affect the performance of autonomous navigation in the AR.Drone or Bebop and ROS.
- Determine if it is viable to use OpenCV libraries based on discrete wavelet transform and multiresolution analysis.
- Consider the use of wavelet families in texture characterization with a multi-class approach.
- Develop an algorithm involving image processing to detect textured objects in AR.Drone or Bebop images or video.

The workflow for this case study allows for visualizing the general idea of the research project. To realize a binary classifier and evaluate its classification performance, see Figure 3.1. In this case, a complete analysis is performed by using the spatial information and the information in the wavelet domain during the learning behavior. The advantages of this configuration are the following:

- The architecture is modular toward another type of dataset.
- It eliminates overfitting during the training stage, the wavelet domain.
- It uses scaled information, therefore fewer parameters to train, the wavelet domain.
- A new data set is generated during an aerial survey in the virtual world to validate the binary detection system.

Disadvantages:

- An empirical and experimental methodology is used in the development and implementation of CNN by Francois Chollet in his book Deep Learning with Python [4].
- Keras Callbacks are not proposed to adjust, control, and monitor the learning of the model.

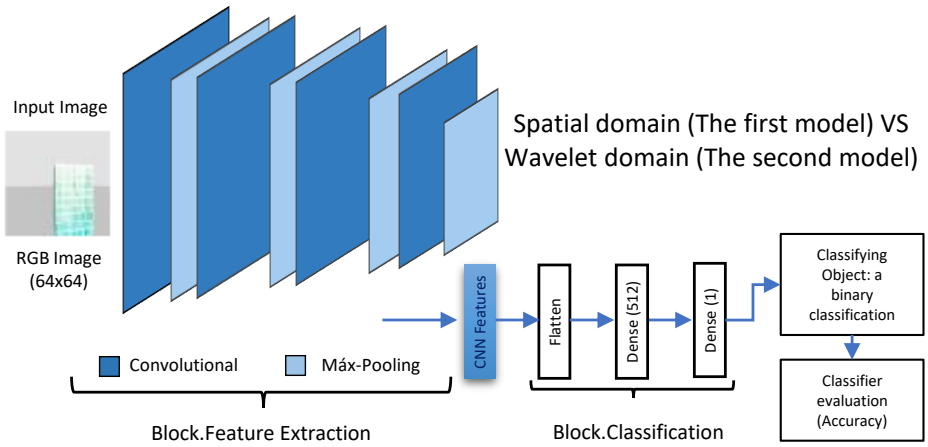


Figure 3.1: Workflow 1: Spatial domain VS Wavelet domain.

The second approach is based on the fact that texture characterization in digital images has become an analysis tool in computer vision. Besides, in practice, CNNs cannot analyze spectral, noteworthy information for texture analysis. Therefore, it is decided to merge spatial and spectral data to improve model learning.

For this case study, the main idea is to create a texture and material classification system. The workflow is modified; these stages are highlighted in orange, as shown in Figure 3.2. It can be seen that this can be concatenated to the CNN architecture from the creation of information in the wavelet domain. Therefore, one model - three inputs are achieved. The new model becomes a multi-class classifier. In addition, the accuracy metric for evaluating the performance of the classifier is still preserved. The advantages of this configuration are as follows:

- The model stops being sequential and changes to a functional model: multiple inputs and outputs, with arbitrary connections between layers.
- Regarding the design of the CNN, an experimental methodology based on the VGG16 architecture, which is one of the most recognized in the state-of-the-art, is used [38].
- The optimizer is replaced by Adam, which has achieved a good performance in classification tasks [65].

- Keras Callbacks are proposed to adjust, control, and monitor model learning.

Disadvantages:

- The model is slow to compile.
- No regularization methods are used to eliminate overfitting.
- Only two wavelet feature vectors are established because one limitation is that the images must be square. In this case, when applying the 3rd level of decomposition, pixels would have to be applied or added to have an image of M×M. It is necessary to emphasize that it is not feasible to alter the information. From this point of connection inside the CNN, the feature extraction falls entirely on the CNN.

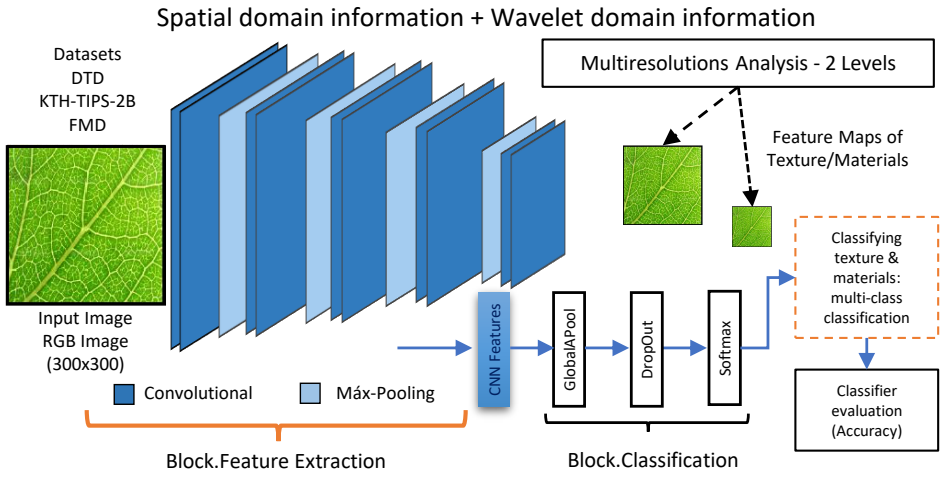


Figure 3.2: Workflow 2: Spatial domain information + Wavelet domain information.

From the second approach, in the third approach, it is decided to add transfer learning to improve classification performance. In context, the third approach is that Micro Aerial Vehicles (MAVs) have increased in engineering and civil applications to explore environments without prior information. In particular, in autonomous navigation, a fundamental part is detecting and locating targets of interest. For this reason, computer vision has become an analysis tool.

For this case study, the main idea is to create an object classification system for aerial navigation tasks. A new stage is added to the workflow proposal, which has to do with integrating the aerial classification system and modifying the feature extraction stage. These stages are highlighted in orange, as shown in Figure 3.3. It can be seen that the information in the wavelet domain can be concatenated after using transfer learning. The advantages of this configuration are as follows:

- The model remains functional: multiple inputs and outputs with an arbitrary connection.
- As for the CNN design, the pre-trained VGG16 architecture is used, and the synaptic weights are taken from the ImageNet database [31, 38].
- Regularization methods are used to eliminate overfitting at the classification stage.
- Keras Callbacks are proposed to adjust, control, and monitor the learning of the model.
- The classification model generalizes its learning toward textured objects in the navigation application.

Disadvantages:

- The model is slow to compile.
- Without a GPU machine, the classifier latency increases, and the performance in the simulator decreases.

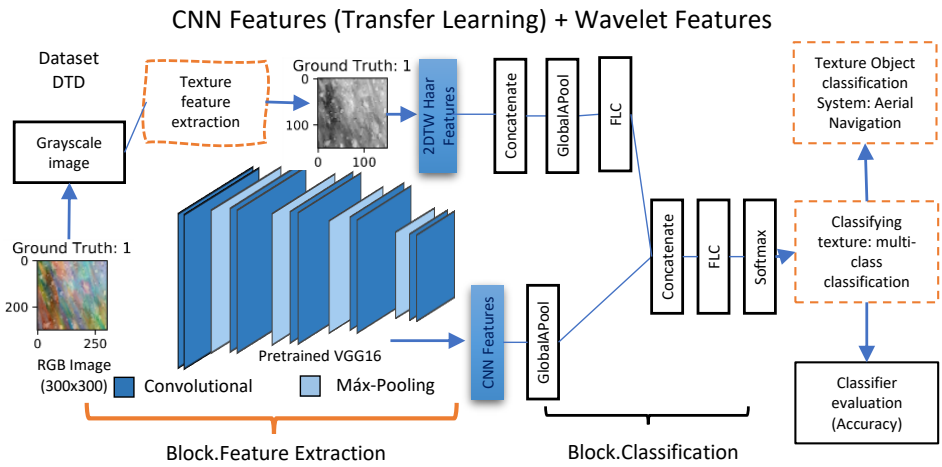


Figure 3.3: Workflow 3: CNN Features (Transfer Learning) + Wavelet Features.

The fourth approach takes the approach that CNNs have recently been proposed as a solution for texture and material classification in computer vision. However, inside CNNs the internal pooling layers often cause a loss of information, which is detrimental to the architecture. While Max-Pooling and Ave-Pooling are efficient and simple methods, they also have shortcomings. Therefore, a methodology is designed to reduce the size of feature maps and preserve image information.

For this case study, the workflow only focuses on the analysis and implementation of the pooling methods, which are highlighted in orange as shown in Figure 3.4. It can be seen that the accuracy metric is still preserved to evaluate the performance of the classifier. The advantages of this configuration are as follows:

- The model is sequential, which allows to speed up model compilation and training.
- The architecture is modular toward another type of dataset and another type of layer.
- The CNN design is based on the VGG16 architecture with only three convolutional blocks [38].
- Regularization methods are used to eliminate overfitting in the feature extraction and classification stages.
- Keras Callbacks are proposed to adjust, control, and monitor the learning of the model.

Disadvantages:

- An empirical and experimental methodology is used in the development and implementation of CNN by Francois Chollet in his book Deep Learning with Python [4].
- The analysis is only performed with traditional pooling methods (Max-Pooling and Ave-Pooling).
- The wavelet pooling method is implemented only with the Haar wavelet transform.

Each of the fourth approaches is described in depth below.

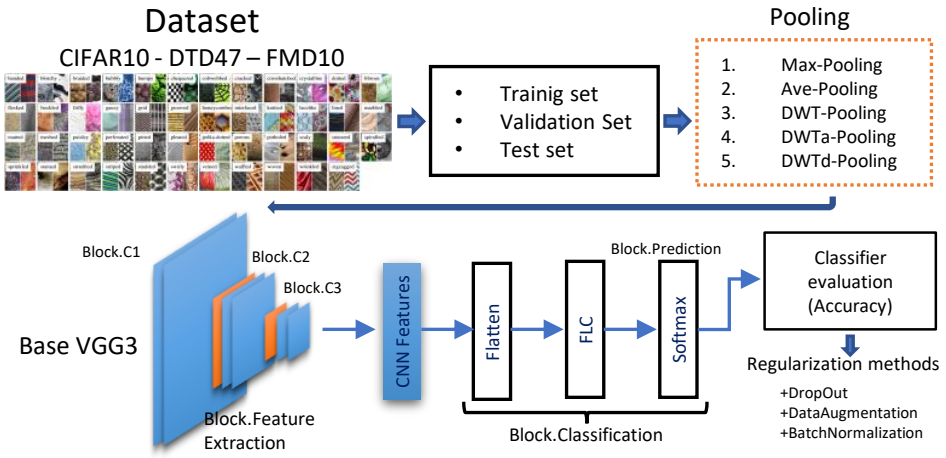


Figure 3.4: Workflow 4: Discrete Wavelet Transform Pooling.

3.1 Object Detection in Aerial Navigation using Wavelet Transform and Convolutional Neural Networks: A first Approach.

This work proposes a first approach based on wavelet analysis inside image processing for object detection. The idea is to detect objects with a repetitive pattern and use a binary classification system to navigate simulated environments. Currently, it has become common to use algorithms based on Convolutional Neural Networks (CNNs) to process images obtained from the on-board camera of Micro Aerial Vehicles (MAVs), being useful in detection and classification tasks. CNN architecture can receive images without pre-processing as input in the training stage. This advantage allows us to extract the characteristic features of the image.

Nevertheless, in this work, it is argued that characteristics at different frequencies, low and high, also affect the performance of CNN during training. Therefore, a CNN architecture is proposed complemented by the 2D Discrete Wavelet Transform, which is a feature extraction method. Wavelet analysis allows us to use time-frequency information through a multiresolution analysis. The information improves the learning capacity, eliminates overfitting, and achieves better target detection efficiency. Also, the learning model was evaluated in the aerial navigation of a drone.

3.1.1 Materials and Methods.

In this section, the experimental structure is presented with two detection models. The first model involves images in the spatial domain, original images without pre-processing. The second model uses images in the wavelet domain, so the images are pre-processed at three decomposition levels before entering the architecture. The two datasets are previously acquired in a recognition and navigation stage in the Gazebo simulation environment. Besides, it is essential to mention that the ConvNet architecture shown in Figure 3.5 was used in both models. The neural network was trained using the machine learning library Keras, and TensorFlow was used as the backend.

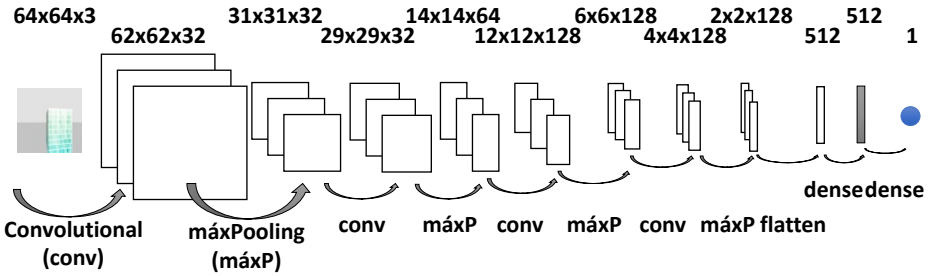


Figure 3.5: Architecture of deep neural network (ConvNet) with binary output.

As a rule, a way to evaluate the learning capacity of the model and the detection performance is to compare the accuracy and loss metrics. These two statistics are usually sampled during the training, validation, and testing. Finally, this model is evaluated in the aerial navigation application, using the Robot Operating System (ROS) and the Gazebo simulation environment.

3.1.1.1 Experimental setup.

Keras is one of the deep learning frameworks with tools to create new and pre-trained models. Besides, it is an open-source project and makes deep learning easy to implement sequentially using blocks [4]. Integrating the learning model with the ROS framework uses the special libraries of Keras and OpenCV — a free artificial vision library. For this case study that is proposed with a small dataset to train the model, it is decided to use a ConvNet architecture, with results in the classification of Dogs against Cats [4]. This dataset is not distributed in the Keras framework, so it had to create our experimental dataset. The CNN design is a stack of 2D convolutional layers with a *Rectified Linear Unit* (ReLU) activation function alternated with a `MaxPooling2D` layer. Also, the depth value of the hidden layers progressively increases from 32 to 128, while the size of the feature maps decreases from 62×62 to 2×2, as shown in Figure 3.5. By having a binary classification approach, the network will end with a single unit (a dense layer of size 1) and a *Sigmoid* activation function — a non-linear function where it is a guarantee that the output of this unit will always be between 0 and 1. It means that the unit will encode the probability that the architecture is oriented to one class or another.

3.1.1.2 Original and Wavelet Dataset.

The ConvNet architecture in the two detection models is parameterized to train and predict only images with a size of 64×64 pixels (a decision that depends on the computational power of the CPU, and at the next level, the image quality would decrease) and the three channels RGB. In the scaling functions of the OpenCV library is the `cv.resize(src,dst,dsize,fx,fy,interpolation)` function. The return type of this function is void, and the input arguments are the source image, the target image, the size of the target image, the x and y scaling factors, and the *interpolation* technique. The default scaling algorithm is Bilinear Interpolation, often used to improve image quality after performing spatial transformation operations such as digital zoom or rotation [68]. Bilinear interpolation takes a weighted average of the four neighborhood pixels to calculate its final interpolated value. This technique gives better results than nearest neighbor interpolation and requires less computation time than bicubic interpolation [69].

Therefore, for the first detection model, the original images (640×380 pixels) are resized to 64×64 pixels. While the second model uses the wavelet images, first, the original image (640×380 pixels) is conditioned to a size of 512×512 pixels for multiresolution analysis. Three decomposition levels are applied to get the network input in this case. The process generates one approximation sub-image and three detail sub-images (horizontal, diagonal, and vertical). As illustrated in Figure 3.6, the wavelet process is applied to the two classification datasets (classes). In this case, it is proposed to use only the approximation sub-sets (images of 64×64 pixels) because it is where the most energy is conserved. A Haar mother function is used in multiresolution decomposition, as it is one of the most common functions in image processing and feature extraction. The two detection models have a binary output, so the detection task is focused on two classes; the first class detects the presence of a textured object in the image plane, and the second class where the object is out of the scene. Mainly, both data classes are divided into 700 images for the training stage, 150 for validation, and 105 for testing (955 per class). Each split contains the same number of samples from each class: this is a balanced binary-classification problem, which means classification accuracy will be an appropriate measure of success [4].

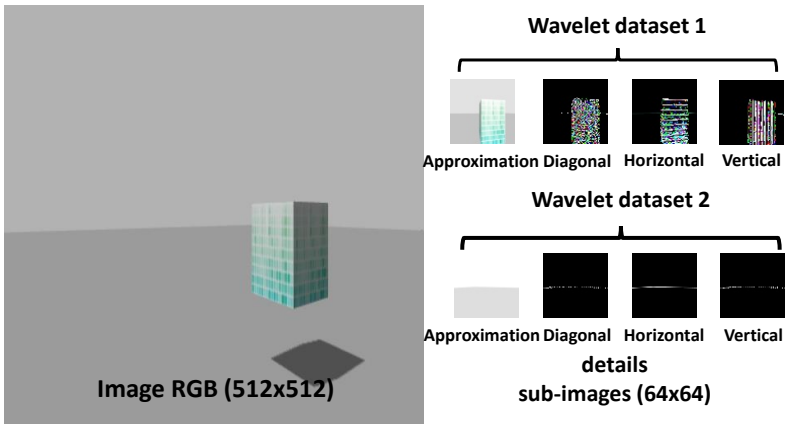


Figure 3.6: Wavelet sub-image dataset for the second detection model; in this case, it only focuses on the approximation sub-images.

3.1.2 Experimental Results.

The two detection models are used to demonstrate the contribution of wavelet analysis in conjunction with CNN architectures. Furthermore, the experimental development allows us to visualize the learning behavior of the two models, so the *Accuracy* and *Loss* metrics are sampled in ten epochs (number of iterations in which the dataset must be learned) within the training and validation stages. Hence, the two detection models are trained with a total of 504,001 parameters, using a computer with Intel® CPU Core™ i5-2450M.

The first learning model, characterized by using only the original images and the ConvNet for binary classification, shows the following results in all ten epochs. The *Accuracy* in the training stage (green line) starts with 78%, then reaches almost 100% in the second epoch; from this point, the learning behavior is random between 98% and 100%. As for *Accuracy* in the validation stage (blue line), the generalization of learning decreases when new data are learned, see Figure 3.7. This effect is caused by overfitting (after three epochs); it begins to learn patterns specific to the training data but misleading or irrelevant when it comes to new data. Meanwhile, Figure 3.8 shows the *Loss* metric to evaluate the learning model. A *Loss* value near-zero is obtained very quickly at the training stage (green line); this effect makes the network more susceptible to overfitting. For the new data, the loss in the validation stage (blue line) decreases and increases considerably (three and seven epochs), preserving the overfitting effect.

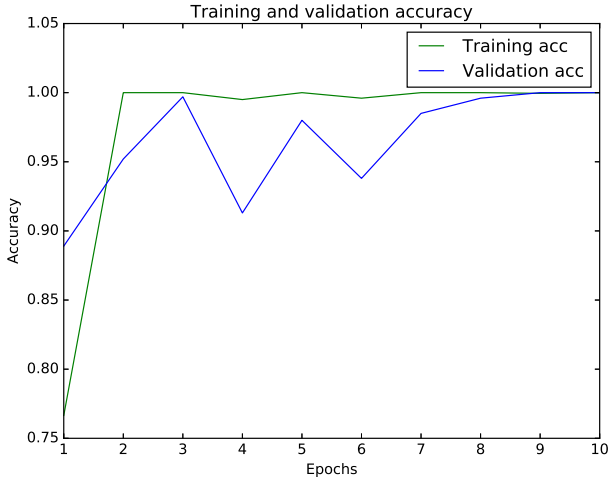


Figure 3.7: Capacity of the model in the accuracy of training and validation, without pre-processing of the input images to the ConvNet.

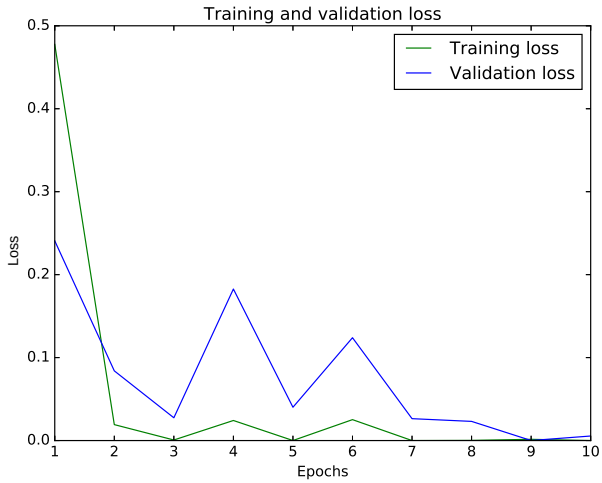


Figure 3.8: Capacity of the model in the loss of training and validation, without pre-processing of the input images to the ConvNet.

Differing from the first model, the second learning model is under the same conditions as the ConvNet architecture but now with the conjunction of the images in the wavelet domain (only approximation sub-images). The approach with wavelet analysis has the following results in the ten training and validation epochs. As illustrated in Figure 3.9, the model's capability has a better performance in the learning generalization and avoids overfitting to the new validation dataset. In the

training stage, it initially reaches 68% in the *Accuracy* metric (green line), but it is until the fourth epoch that it gets almost 100%. The *Accuracy* metric performs better in the validation stage (blue line) since the learning generalization is higher than the training data, avoiding overfitting new data and achieving almost 100% in the second epoch. Moreover, the model achieves its *Loss* in the training stage (green line) very slowly, approaching zero until the seventh epoch; this effect allows us to have a model that is not susceptible to overfitting new data, see in Figure 3.10. In the case of the validation stage, a near-zero loss value is obtained very quickly (blue line); this is due to the learning capacity and the quality of the images in the wavelet domain.

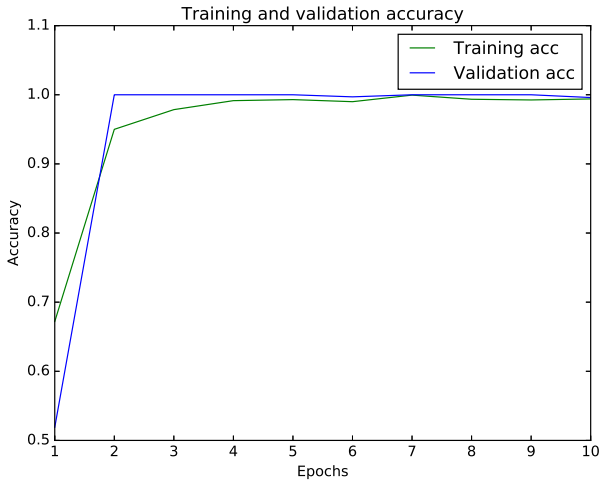


Figure 3.9: Capacity of the model in the accuracy of training and validation, with the wavelet dataset.

Therefore, a model with higher detection and classification performance is obtained, unlike the first learning model using the original dataset. The generalization of knowledge allows us to learn new information adequately; the result is a smaller difference between the *Loss* metric in the training stage and the *Loss* in the validation stage. The contribution of this work allows optimizing the learning capacity for object detection in aerial navigation applications. Besides, some advantages of converting the data to the wavelet domain are to have images with meaningful learning of the physical characteristics of the object, oriented to the details of the texture, and eliminate the overfitting when having a small training dataset. Finally, Table 3.1 shows the success rate for the wavelet test dataset. The *Accuracy* and *Loss* metrics validate the learning model created by the information in the spatial domain and by the information in the wavelet domain. Likewise, it is the third dataset that has never been observed by the model.

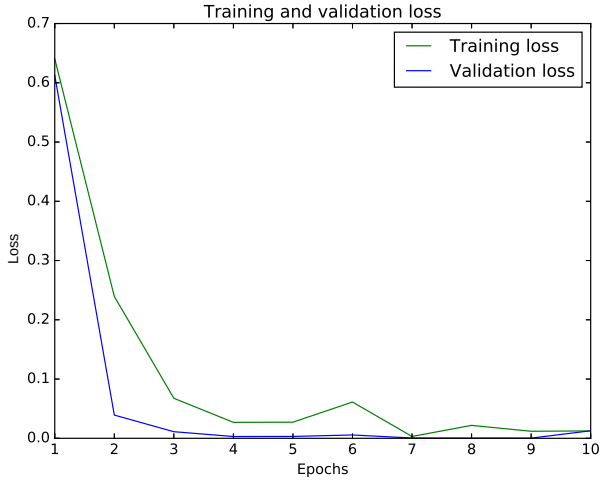


Figure 3.10: Capacity of the model in the loss of training and validation, with the wavelet dataset.

Table 3.1: The test set results for each learning model (Spatial domain VS Wavelet domain).

Topic	Model 1 [%]	Model 2 [%]
Test Accuracy	100	100
Test Loss	0.41	0.61

3.1.2.1 Simulation Experiments.

A simulator (Gazebo) is used to design UAVs like the Parrot AR.Drone 2.0 and the development of realistic 3D scenarios [70]. Specifically, it allows fast execution of algorithms, provides a user interface, and controls the navigation of the AR.Drone in a fluid way. Also, Gazebo gives an identical control to the one used in real vehicles. In the development of the applications, it allows the connection of the ROS operating system — a system created under a Berkeley Software Distribution (BSD) license with the open-source trend. ROS provides the functionality of an operating system in a heterogeneous computer cluster. The way to transmit messages in ROS is through the nodes, which allows them to be programmed in any language with client libraries for ROS (like C, C++, Python, Java, and Matlab) [71].

Concerning the second detection model, the simulator provides real-time support for evaluation and execution. It is important to emphasize that the image acquisition in the evaluation stage are images from the on-board camera of AR.Drone (front

view) as if it were the real camera. Images initially have a size of 640×380 pixels, so the images are conditioned to a size of 512×512 pixels. After, these images are converted to the wavelet domain via multiresolution analysis at a scale level equal to three. Therefore, the result is four sub-images with a resolution of 64×64 pixels (value allowed for the detection model).

The real-time execution it is showed different perspectives from the on-board camera of the drone. In this case, two classes of prediction are displayed correctly; the first class in which the object with texture appears entirely in the scene, as illustrated in Figure 3.11, and the second class in which it does not appear in the image plane, see Figure 3.12. A video of this work for review purposes is available at `https://youtu.be/MOSrJyf14T8`. Also, the average detection rate reaches 98% in different perspectives (scene scale and lighting variations), as illustrated in Figure 3.13. Proof of this, the detection model resulted in an excellent performance in prediction times, as shown in Table 3.13.

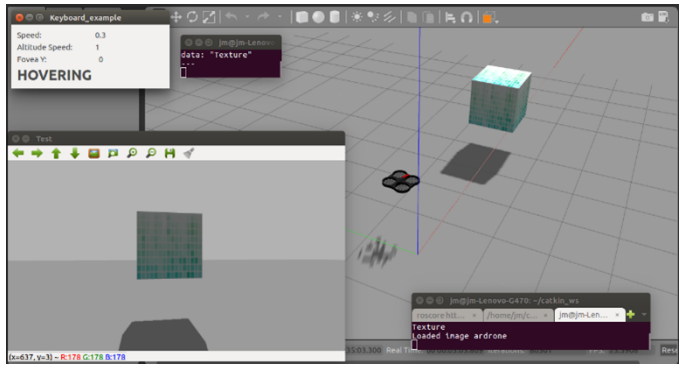


Figure 3.11: Test 1: Object detection in scene applying the proposed method with the approach of wavelet analysis and deep learning.

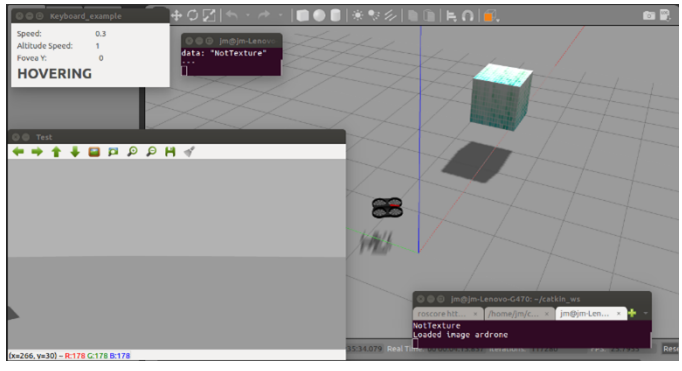
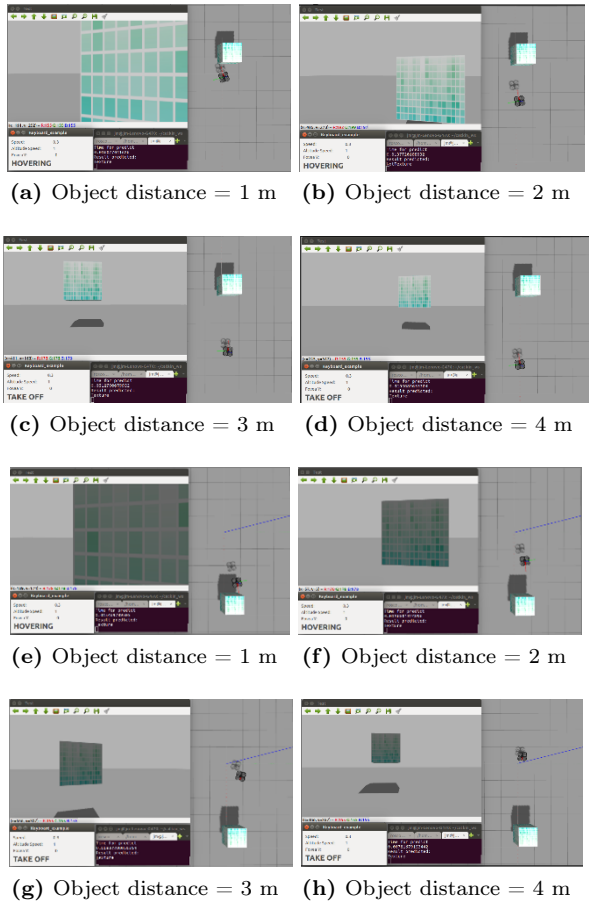


Figure 3.12: Test 2: Object is not detected on the scene, as it is out of view from the on-board camera of the drone.

It is demonstrated that the interaction of a ConvNet neural network with a wavelet dataset in robotic applications could achieve promising results in the learning stage. The simulation environment provides a complete analysis of the proposed method; therefore, it can be adapted to real conditions for classifying and detecting objects with textures. Nevertheless, without leaving aside the prediction time reported, it is advisable to use a computer with high processing capacity, given that the simulation consumes many resources for its operation.



(a) Object distance = 1 m **(b)** Object distance = 2 m

(c) Object distance = 3 m **(d)** Object distance = 4 m

(e) Object distance = 1 m **(f)** Object distance = 2 m

(g) Object distance = 3 m **(h)** Object distance = 4 m

Figure 3.13: Detection results at four different distances from the target to the drone camera, and different scales and illumination (a)-(d) are results based on a perspective in the environment, while (f)-(h) are results based on an opposite perspective in the environment (the image of the target contains shadow).

3.1.2.2 Object Detection in Aerial Navigation.

This section shows two virtual worlds for aerial navigation. The goal is to validate the proposed methodology in an environment that provides a real-world perspective. Two training sets are offered in the wavelet domain in both virtual worlds, as shown in Figure 3.14 and Figure 3.16. These two wavelet datasets are used as input to the CNN architecture, which is shown in Figure 3.5. For the first virtual world shown in Figure 3.15, it is obtained an *Accuracy* rate of 99.8% in the test stage (out of 300 frames) and a *Loss* value of 0.44%. A video of this work is found at `https://youtu.be/6WbQjKRFI_E`.

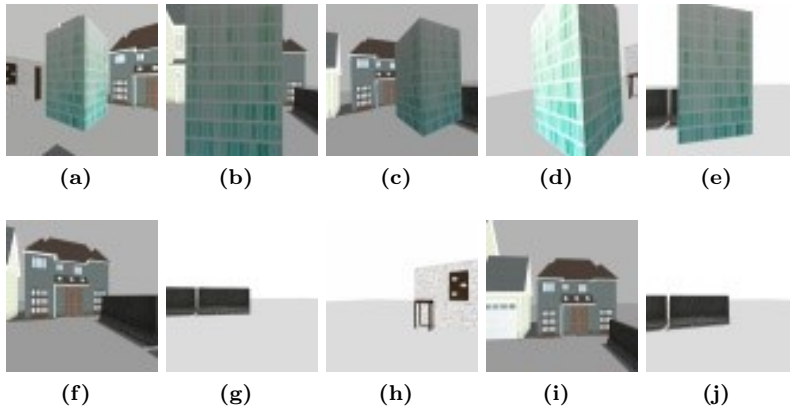


Figure 3.14: Test dataset 1: Images of the virtual environment to perform the binary classification, (a)-(e) Class *CubTexture*: images with the textured cube and (f)-(j) Class *NotCubTexture*: images without the textured cube.

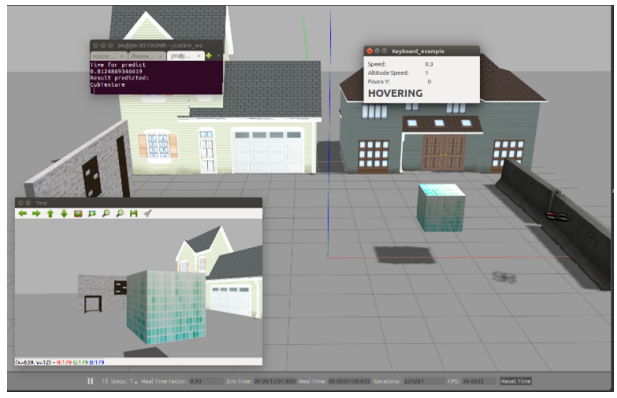


Figure 3.15: Virtual world 1: The learning model achieves object detection on the image plane.

For the case of the second virtual world, as shown in Figure 3.17, in the test stage, it is obtained a 99.7% *Accuracy* rate (out of 450 frames) and a *Loss* value of 0.89%. A video of this work for review purposes is available at `https://youtu.be/tGo4wpvsoyI`. The results show that both environments have a high performance in detection capability.

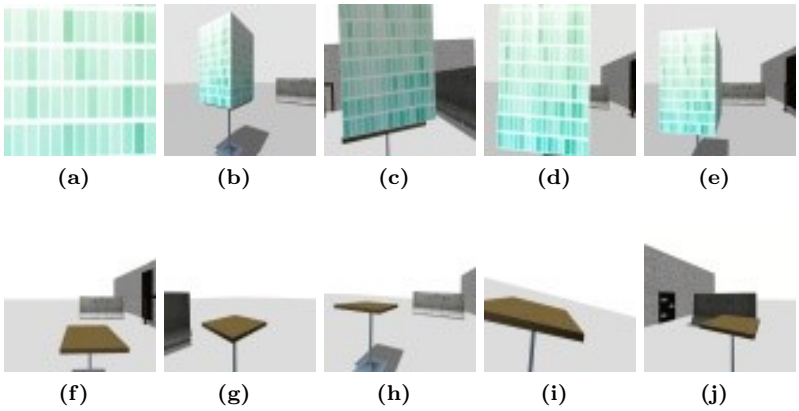


Figure 3.16: Test dataset 2: (a)-(e) Class *CubTexture*: Images with the textured cube on the table and (f)-(j) Class *NotCubTexture*: images of the environment and without the cube on the table. Both wavelet datasets are trained for binary classification.

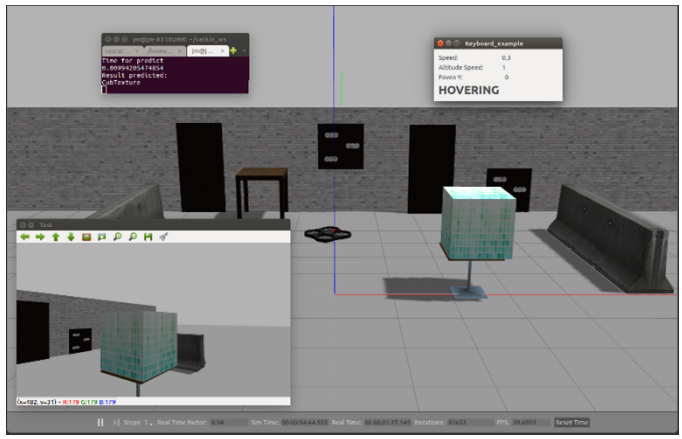


Figure 3.17: Virtual world 2: Example of detection. The on-board camera of the drone detects the textured object on the table.

3.2 System for Image Texture Classification using Deep Learning and Wavelet Features.

Texture characterization in digital images has become an analysis tool in computer vision. Texture in visual perception is an important physical property because it provides information about the structural composition of surfaces and objects in the image. On the other hand, texture analysis in machine learning plays an essential role in object classification, detection, and localization tasks.

In image processing, texture can be defined from neighboring pixels and intensity distribution over the image [9]. In addition, there are some classification methods for texture analysis such as statistical, geometric, model, and spectral. Meanwhile, spectral methods describe the texture in the frequency domain. They are based on the decomposition of a signal in terms of basis functions and use the expansion coefficients as feature vector elements.

In this work, a classifier is made for the databases: KTH-TIPS-2B (KT2B), Describable Textures Dataset (DTD), and Flickr Material Database (FMD). Moreover, the adaptivity of deep learning with the wavelet transform is studied, particularly an approach to the Wavelet CNN architecture [21]. Also, an empirical and experimental methodology is used in developing and implementing Convolutional Neural Network (CNN) and wavelet analysis, both as feature extraction methods.

Internally this system is divided into two stages: the first corresponds to feature extraction and the second to the classification stage. Regarding the feature extraction stage, the created tensor has a set of numerical parameters that describe the image content, such as the color, texture, or shape of the object. Therefore, the feature extraction stage is important for the overall success of any image classification and recognition system.

3.2.1 Materials and Methods.

Deep learning is a subfield of machine learning, a new way of learning features from data, such as text, audio, and images [72]. In this area, the term *deep* does not refer to any architecture; rather, it represents the idea of successive layers of representations at different levels. These new representations are getting more and more meaningful. On the other hand, inside deep learning are CNNs, specialized neural networks for data processing.

The proposed methodology for improving performance on the learning model is summarized in three stages, shown in Figure 3.18. Each step is described below.

Figure 3.18: Deep learning process.

For the first stage, select the data with which the model learning will be generalized. In this case, the images of KT2B, DTD, and FMD, which contain images with different textures and materials in natural conditions, are used. These datasets will be described below.

3.2.1.1 Dataset.

The first dataset selected is KTH-TIPS-2B (KT2B), containing 432 images classified into 11 classes. Each class consists of four samples, and each sample has 108 images [35]. Figure 3.19 illustrates the 11 classes with four random samples.

Figure 3.19: Example images from the KTH-TIPS-2B dataset.

The second dataset is the Describable Textures Dataset (DTD) which contains 47 classes of 120 images in the wild, meaning that the images were acquired in uncontrolled conditions [36]. This dataset includes ten available divisions with 40 training images, 40 validation images, and 40 test images for each class. Image sizes range between 300×300 and 640×640, and the images contain at least 90% of the surface representing the category attribute. Figure 3.20 shows some pictures from this set.

Figure 3.20: Example images of the DTD dataset.

The third dataset is the Flickr Material Database (FMD). It is built with various everyday materials (e.g., glass, plastic, etc.). Each image is manually selected from Flickr (under Creative Commons license) to ensure lighting conditions, compositions, colors, texture, and material subtypes. FMD has become a benchmark for material recognition in the computer vision community [37]. Some images are shown in Figure 3.21.

Figure 3.21: Example images from the FMD dataset.

3.2.1.2 Data Preparation.

Since the images within the sets are initially sized differently, in the pre-processing stage, the images are resized to a size of 300×300 (a decision that depends on the NVIDIA GPU's computing power and the images' size because they range from 300×300 to 640×640) for the three sets, KT2B, FMD, and DTD. In the scaling functions of the OpenCV library is the `cv.resize()` function. The default scaling algorithm is Bilinear Interpolation [68, 69]. Also, when using a CNN architecture, the inputs must be processed. For that reason, when working with images, it is convenient to normalize the pixel values from 0 to 1. As a result, the model can quickly converge to a local minimum because inputs with large integer values can slow down the learning process.

3.2.1.3 Model Evaluation.

In this stage, a dataset is randomly generated and divided into three subsets, one for training with 70% of the images, another for validation with 15%, and the remaining 15% for the testing stage [4]. This new distribution of images for model evaluation during training and validation will be applied to the three datasets (KT2B, FMD, and DTD). Then, the training parameters are selected in order to evaluate the model during each epoch or learning iteration. In the search for kernel filters, which establish the characteristic features of each texture, the one with the best performance must be selected automatically. Therefore, this process allows adjusting and updating the model as the architecture is trained.

On the other hand, it is possible to use a performance metric when having classes with the same number of images. In this case, *Accuracy*, a very relevant performance metric in classification tasks, can be calculated. *Accuracy* (acc) is calculated as a percentage of images that the created model correctly labels. Regarding model performance, one way to determine error patterns in texture prediction or classification is by using the *multiple confusion matrix*. This is a table of N×N, which summarizes the level of success of the predictions of a classification model; that is, the correlation between the label and the classification of the model. In this case, N represents the number of classes, one axis of the confusion matrix is the label that the model predicted, and the other is the actual label.

3.2.1.4 Method for Feature Extraction.

Different feature extraction methods generally lead to different texture information elements. Therefore, taking up the main idea of the Wavelet CNN [21], it is decided to design an approximation of the architecture. The result is a hybrid system that combines the features generated by the CNN through the *filters* or *kernel*, the attributes or feature maps generated by hand with the Haar wavelet transform through multiresolution analysis at two decomposition levels.

The network design includes two separate processes (feature extraction using CNN and feature extraction using wavelet analysis) later merged in the model training. In the first process, the RGB image pre-processed in the data preparation stage enters as a tensor to the base VGG architecture to achieve specific patterns automatically. This will allow to classify the textures and discriminate one from another. The second process is performed additionally. In other words, before entering the spatial information into the CNN architecture, new synthetic data must be generated, the attributes (wavelet coefficients) or feature maps in the

spectral domain. It should be emphasized that the multiresolution analysis process in the decomposition stage. It allows generating feature maps that can be adapted to the convolutional blocks of the base CNN. Therefore, with the fusion of these processes in the feature extraction stage, it is possible to move to the next step of deep learning for classification. In Figure 3.22, the classification system with a spatial approach and the fusion of the spectral approach is shown.

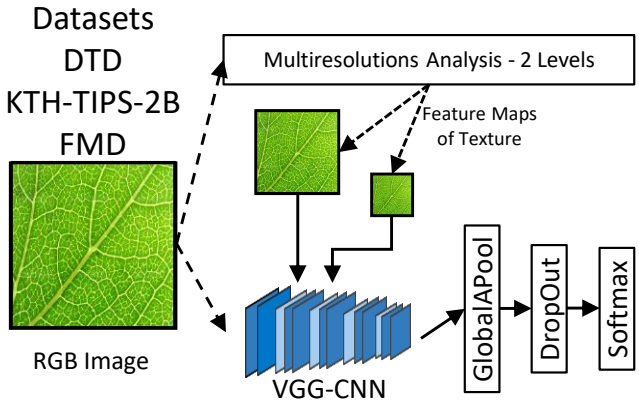


Figure 3.22: Architecture for a textured image classification system.

3.2.1.5 Experimental Setup.

The proposed classification system is illustrated in Figure 3.22. The architecture is designed according to the VGG16 network [36], it has five convolutional blocks with *kernels* of size 3×3 and a `padding (same)` so that the output has the same size as the input. In addition, each convolutional block contains two convolution methods `Conv2D`, the first with a *stride* of 1 and the second with a *stride* of 2 to enable the output to be half the size of the input. This allows the convolutional blocks to extract texture features in the spatial domain.

On the other hand, the VGG architecture and the multiresolution analysis (decomposition stage) have the same reduction feature. It is possible to concatenate each level of decomposition (information in the spectral domain) with the feature maps of each convolutional block. In order to determine the characteristic attributes of each texture, it was decided to use the Haar wavelet transform at two levels. This factor of 2 depends on the possibility of decreasing the image size. Furthermore, avoid using `padding` that affects the wavelet characteristics at the next level. Also, it is essential to mention that this process is applied for each of the RGB channels of the image, performing the individual decomposition per channel. In the end, the maps found are concatenated in a vector. The new information is concatenated into

a feature cube (spectral and spatial), and this must be transformed to change its representation using the `GlobalAveragePooling2D` method. This new vector, in turn, feeds the regularization method *DropOut* to avoid overfitting before passing through the last prediction layer, *Softmax* [73].

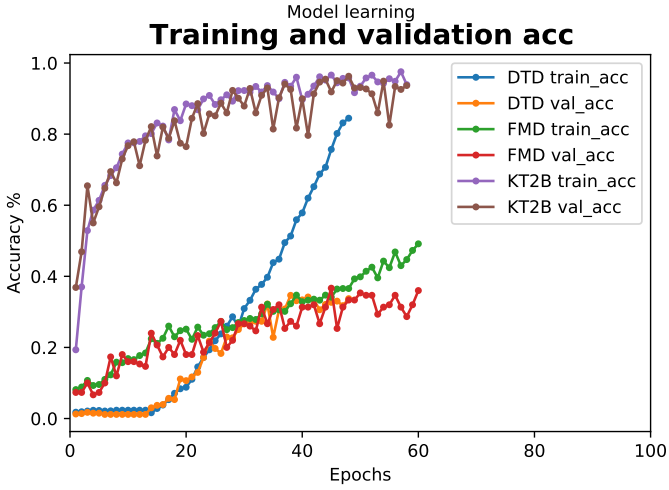
3.2.1.6 Implementation.

The system implementation is based on the Python language and the Keras API with Tensorflow as Backend [4]. In summary, the classification system has 5,441,866 trainable parameters or synaptic learning weights. Also, in selecting the hyperparameters, a learning rate value is defined as 0.001, a *minibatch* of 30, 500 training epochs for learning, four *Callbacks API* to improve model performance (*ModelChekpoint*, *EarlyStopping*, *CVLogger*, *ReduceLROnPlateau*), besides the Adam optimizer — a variant of Descent Gradient [65]. On the other hand, the OpenCV libraries are used for image processing due to their ease of use and adaptability in programming. In the case of the additional method, the Haar wavelet transform, the Pywt library is used [74].

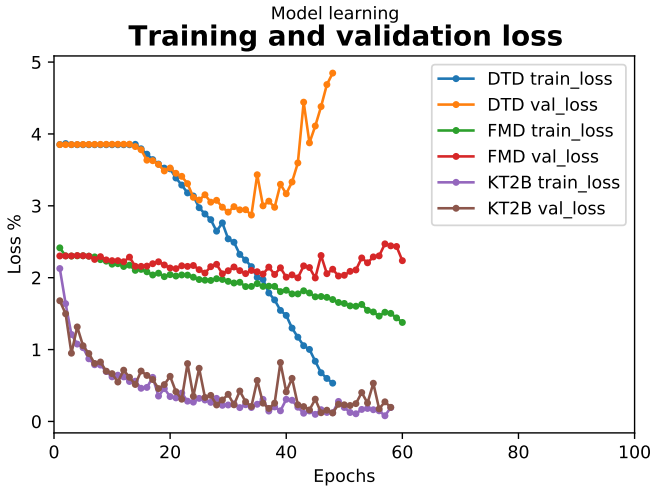
3.2.2 Experimental Results.

Three datasets (KT2B, DTD, and FMD) have validated the approach. Two (KT2B and DTD) are special cases of texture databases because they contain images captured under uncontrolled conditions.

The local maximum achieved for the *Accuracy* metric is shown in Figure 3.23a. For the DTD dataset, the maximum value achieved is around epoch 21, with a value of 39.74% in training (*DTD train_ acc*). And for validation (*DTD val_ acc*) of 32.21%. In the case a local maximum for the FMD dataset is given at epoch 45, the *Accuracy* is 34.14% for training (*FMD train_ acc*) and 36.67% for validation (*FMD val_ acc*). Finally, for the KT2B dataset the local maximum occurs at epoch 48, with a better performance, 95.80% *Accuracy* for training (*KT2B train_ acc*) and 96.29% (*KT2B val_ acc*) for validation. Figure 3.23b shows the loss of the model in each of the three datasets. Also, it is shown that there is a point of divergence between the two sets being trained (training and validation). This point matches the epoch number where the local maximum of *Accuracy* was found. The learning behavior for the three data sets is shown in Figure 3.23. The *Accuracy* achieved for the test set in KT2B is 96%, for FMD with 30%, and for DTD is 34%, as shown in Table 3.2. These results are significant, given that these are images that the model has never seen.



(a) Model accuracy.



(b) Model loss.

Figure 3.23: Evaluation of accuracy and loss metrics for training and validation sets.

Table 3.2: Learning performance in the three evaluation sets.

	Train [Acc]	Val [Acc]	Test [Acc]
DTD	0.3974	0.3221	0.3451
FMD	0.3414	0.3667	0.3000
KT2B	0.9580	0.9629	0.9678

In order to evaluate the classification performance of the model, it is determined to use the *multiple confusion matrix*. Note that there is a relationship between the test set and the *multiple confusion matrix* results. For DTD in Figure 3.24, it shows a blue diagonal difficult to detect, and the heat map is still distributed in some areas. This indicates that the developed model still finds similitude in most classes.

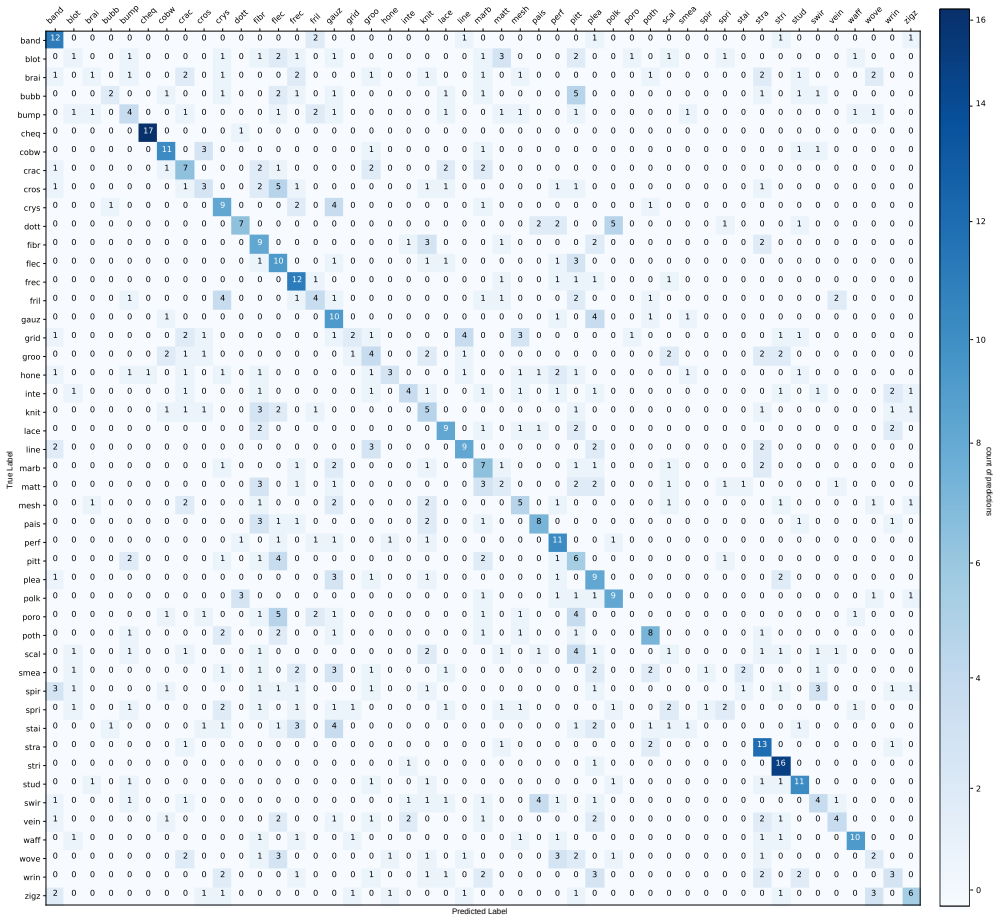


Figure 3.24: Confusion matrix for the DTD dataset.

In particular, the blue diagonal is also very difficult to detect for the set of FMD images, see Figure 3.25. There is no correct prediction in the center of the matrix. The diagonal is not completed with the true label or any other class. Also, it is found that most of the heat map is distributed to the right. On the other hand, for the KT2B dataset, see Figure 3.26, which shows a full blue color diagonal. This indicates a positive prediction trend concerning the original label of the KT2B

textures. Some images test are evaluated for each model; at the top, the true label and the prediction can be seen. In particular, it allows having a visual idea about the classification features and the correlation between classes. In Figures 3.27-3.29 show some test set images.

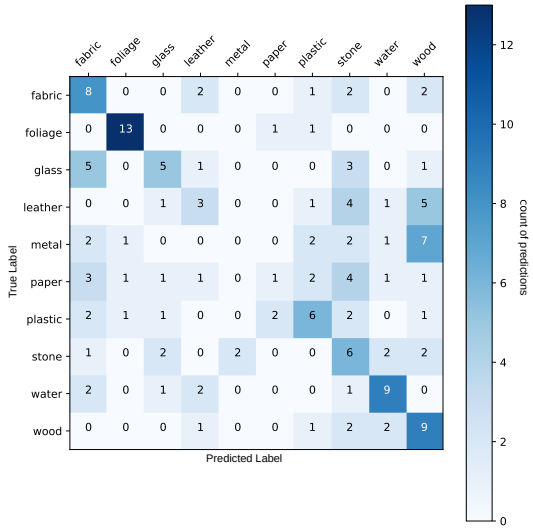


Figure 3.25: Confusion matrix for the FMD dataset.

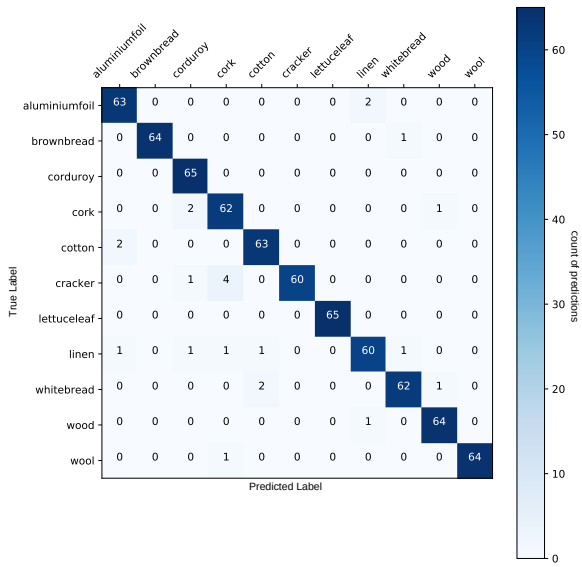
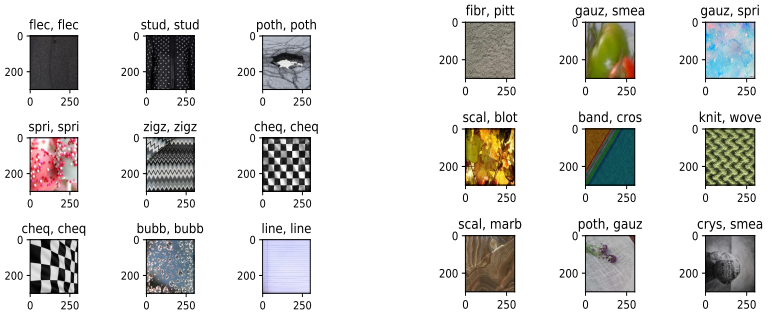
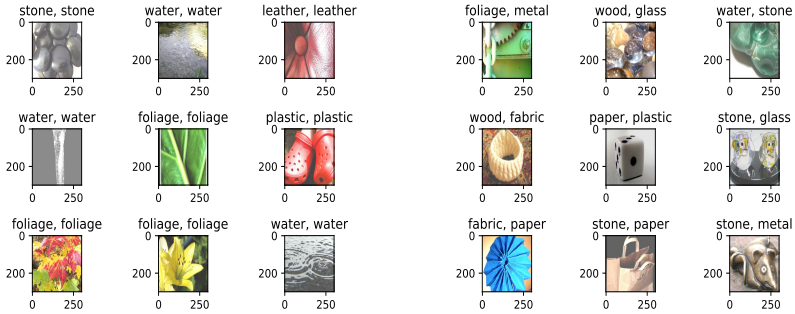


Figure 3.26: Confusion matrix for the KT2B dataset.



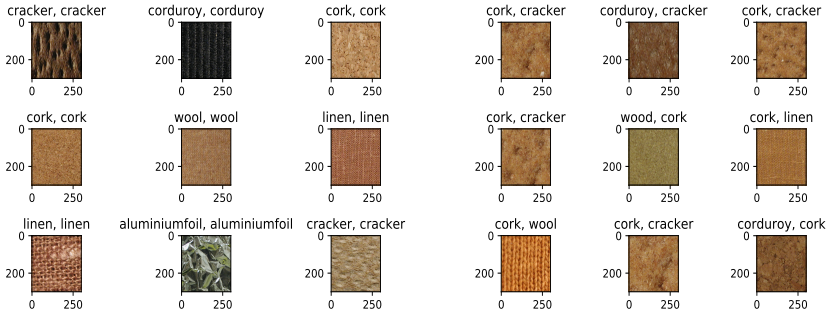
(a) 292 Correctly classified. **(b)** 554 Incorrectly classified.

Figure 3.27: Random texture classification (from 846 images) using the DTD prediction model.



(a) 45 Correctly classified. **(b)** 105 Incorrectly classified.

Figure 3.28: Random texture classification (from 150 images) using the FMD prediction model.



(a) 692 Correctly classified. **(b)** 23 Incorrectly classified.

Figure 3.29: Random texture classification (from 715 images) using the KT2B prediction model.

3.3 Texture Classification for Object Detection in Aerial Navigation using Transfer Learning and Wavelet-based Features.

The use of Micro Aerial Vehicles (MAVs) has increased in engineering and civil applications to explore environments without previous information. In particular, in autonomous navigation, a fundamental part is that of detecting and locating targets of our interest. For this reason, computer vision has become an essential analysis tool. This work focuses on object classification in aerial navigation tasks, where texture is involved as a physical property of the object. It is presented a classification model using transfer learning and wavelet-based features as an additional feature extraction method.

Therefore, it is decided to merge these methodologies (deep learning and wavelet features) as a solution for texture classification. The objective is that the MAV performs aerial navigation (inside the virtual environment) for the classification system to recognize the object, see Figure 3.30. This work focuses on preview information (in data collected by MAV) and structural recognition of the object (with a particular texture) within a region of interest in the image plane. The implementation of our system is developed with the fusion of two approaches. The first is in the spatial domain, using transfer learning. It is a baseline of the VGG16 architecture with the features of the ImageNet database [31, 38]. The second approach focuses on the spectral domain, applying the Haar wavelet transform in two dimensions to obtain features at different scales [34]. The VGG16 network has been selected for its fast performance and implementation with transfer learning and adaptability with wavelet analysis. Internally, the system is divided into two stages: the first corresponds to feature extraction and the second one to the classification stage.

3.3.1 Materials and Methods.

This system allows for the prediction or classifies the texture in images transmitted from the on-board camera of the drone, whose objects are in an outdoor scenario (in Gazebo) — a virtual simulation environment. This work proposes an approach based on transfer learning and wavelet features. There is interest in texture recognition, mainly to know one of the characteristics of the object. So, the image plane (640×360) is limited to a *Region of Interest-ROI* (300×300 pixels). In the

Figure 3.30: An aerial navigation texture classification system based on knowledge inference on the DTD database is designed. See at `https://youtu.be/d41kgBw7Y_c`.

ROI function is the code `frame[30:330, 170:470, :]`. The default ROI code extracts a portion of the image plane. As a result, the system will have the image in RGB and a grayscale version. This decision depends on feeding the CNN with the ROI image and applying the wavelet transform to a single channel. These two images are the inputs for our proposed classification system; see Figure 3.31.

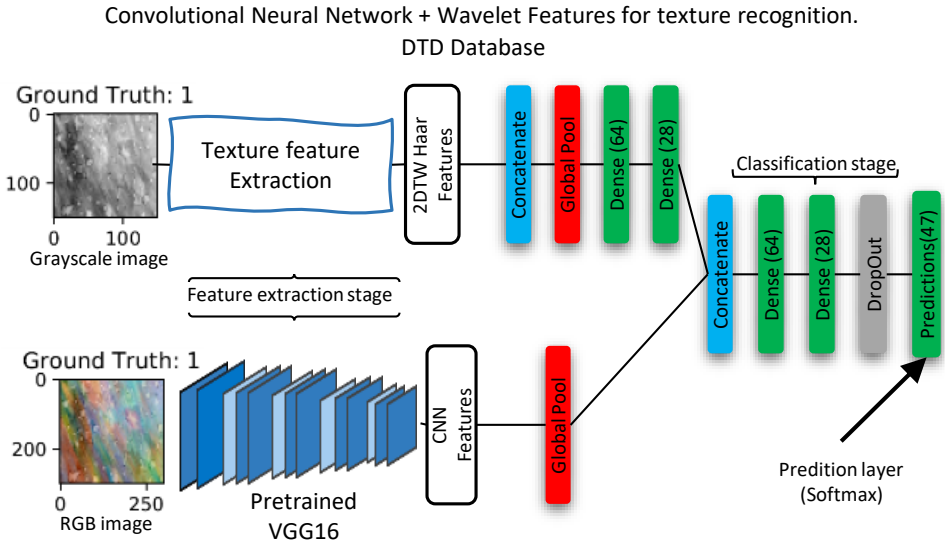


Figure 3.31: Texture classification system.

Describable Textures Dataset (DTD) was selected to be used. It contains 47 classes of 120 images in the wild. This means that the images have been acquired that in uncontrolled conditions [36]. This dataset includes ten divisions available with 40 training images, 40 validation images, and 40 test images for each class. Our experiment will create a new dataset, with the distribution of 70% for training, 15% for validation, and the remaining 15% for testing [4, 14]. Figure 3.20 shows some images from this set. One limitation of the dataset is the number of images per class, so it is decided to use the transfer learning method to improve the classification performance of our model. The synaptic weights are based on the ImageNet database, which will feed the feature extraction stage of the base architecture VGG16. Moreover, this approach was not decided to use KT2B and FMD because of the limitation in the number of classes. The model is intended to generalize to a variety of textured objects, such is the case of the DTD dataset.

Before training, the Haar wavelet transforms in two dimensions is applied to one level (a decision that depends on obtaining spectral information and decreasing the computational cost), see Figure 3.31. The factor of one represents the level of image decomposition. This new spectral information is essential for classification. Therefore, four sets (in the wavelet domain) are automatically generated to determine the characteristic attributes of each texture. This information can be combined with the spatial information of the VGG16 architecture. Also, it is essential to mention that this process is only applied to the image previously converted to grayscale, performing the decomposition for a single channel, see Figure 3.32 & 3.33.

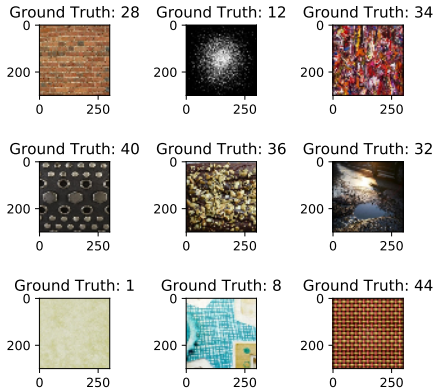
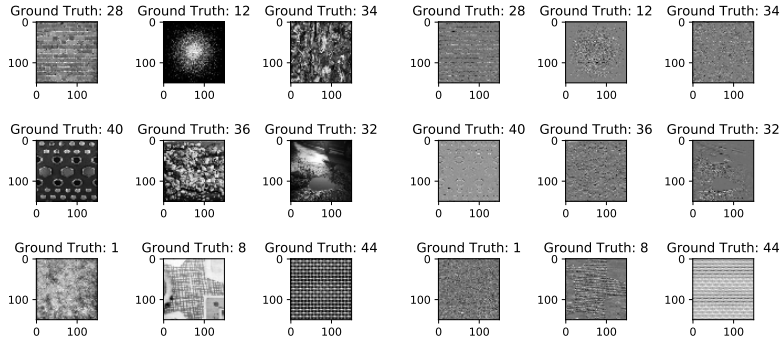
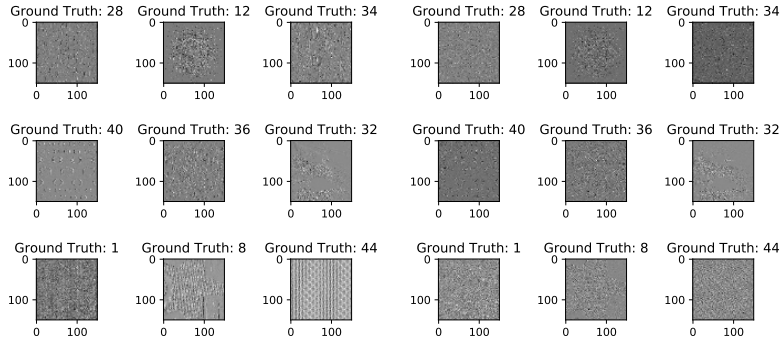


Figure 3.32: Images textures that have been decoded (Class) to train the classification model.

For the test stage in MAV, a scenario is designed in the Gazebo simulator. The virtual scenario is created with ten cubes with certain textures (Figure 3.30). These textures are selected due to the performance achieved in the model testing stage. Therefore, the chosen classes have performance above 70% *Accuracy* (Table 3.4).



(a) Approximation **(b)** Horizontal details



(c) Vertical details **(d)** Diagonal details

Figure 3.33: Approximation and details set of wavelet features.

3.3.2 Experimental Results.

The experiment to train our learning model was carried out with the Keras API with Tensorflow as Backend [3-17] [4]. Besides, the OpenCV libraries are used for image processing due to their ease of use and adaptability in programming. Also, it is used the Pywt library [74], from which the Haar wavelet transform was chosen as the feature extractor method. An aerial navigation experiment was performed using the ROS framework and Gazebo simulation environment to validate the classification system and its learning generalization. This section describes the results obtained in each experiment.

3.3.2.1 Model Training.

In the first instance, the VGG16 network was trained from scratch. The idea is to see if, with training from scratch, it is able to generalize its learning. Therefore, it is possible to use the transfer learning methodology. Table 3.3 shows the performance of the pre-trained network and the proposal with the wavelet feature fusion. It shows the *Accuracy* performance on the three sets to validate the model (training, validation, and test). In training the pre-trained network, slight overfitting is observed. The model will be adjusted to learn specific instances and unable to recognize new textures. One way to improve the performance of the model is to integrate the wavelet features. In this case, the elimination of overfitting and homogeneity between the three sets is achieved. Besides, the value of the test set is highlighted because these are images that the model has never seen. In summary, the classification system has 14,778,735 synaptic learning weights. 64,047 are trainable parameters, of which 16,832 correspond to wavelet features.

Table 3.3: Classification results for the pre-trained VGG16 network and our model indicated as accuracy (%).

	Training	**Validation**	**Test**
Pre-trained model	68.15	50.41	54.49
Our model	57.67	51.22	**53.19**

3.3.2.2 Texture Classification DTD.

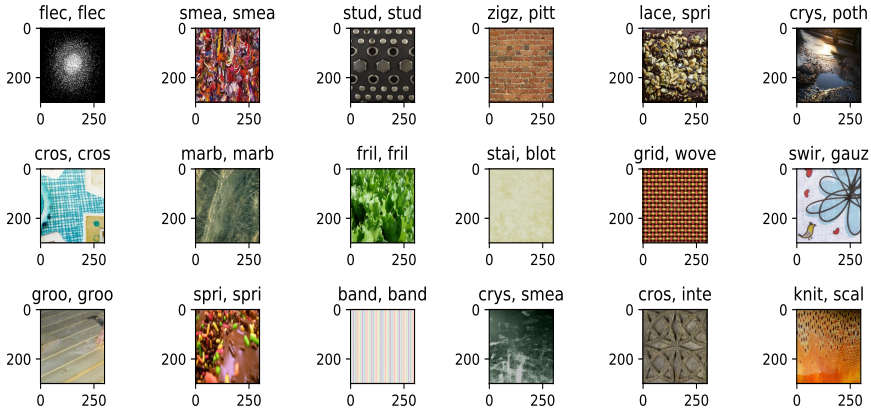
Other metrics evaluate the performance of the DTD dataset classes. The metrics such as *Precision*, *Recall*, and *F1-score* are given when applying the `classification_report` method, where it is necessary to involve the true labels and the prediction label of the model. Table 3.4 shows the classes that performed above 70% classification. Also, Table 3.5 shows three random classes that perform above 50% classification. This class selection analysis provides the basis for the design of the textured cubes in the Gazebo environment. On the other hand, we can observe the similarity and correlation between classes (about the test set) by performing the prediction. Figure 3.34 shows the true and prediction labels at the top of each texture.

Table 3.4: Classes (test set) that results with precision above 70%.

Class	Precision	Recall	F1-score	Test
bubb	0.73	0.61	0.67	18
cheq	1.00	0.78	0.88	18
fibr	0.73	0.61	0.67	18
fril	0.72	0.72	0.72	18
stri	0.77	0.94	0.85	18
stud	0.70	0.78	0.74	18
zigz	0.75	0.67	0.71	18

Table 3.5: Classes (test set) that results with precision above 50%. They are chosen from the easy human visual perception of the texture.

Class	Precision	Recall	F1-score	Test
hone	0.58	0.61	0.59	18
line	0.50	0.28	0.36	18
polk	0.65	0.61	0.63	18



(a) 450 Correctly classified **(b)** 396 Incorrectly classified

Figure 3.34: Classification of textures randomly (from a total of 846 images) using the DTD prediction model.

3.3.3 Texture Classification in Aerial Navigation.

The classification model is tested in navigation and aerial recognition, creating a virtual environment with the Gazebo simulator. They are controlling and sending information from the on-board camera of the drone using the ROS framework. In the world presented in Figure 3.30, the ten cubes with the selected textures are positioned in a row. Therefore, the position of the cubes allows the evaluation of the prediction model during aerial exploration. The idea of the model is that it generalizes its learning to textured objects. In total, 1000 image captures were performed in a navigation recognition for each class. The proposed texture sets bubbly, chequered, honey and striped, studded obtain a high correlation with their original label above 60% *Accuracy*. In contrast, the fibrous, polka-dotted, and zigzagged textures obtain a low performance, below 20% of the total captured images. Moreover, the frilly and lined textures show minimum correlation with their original label, see Figure 3.35.

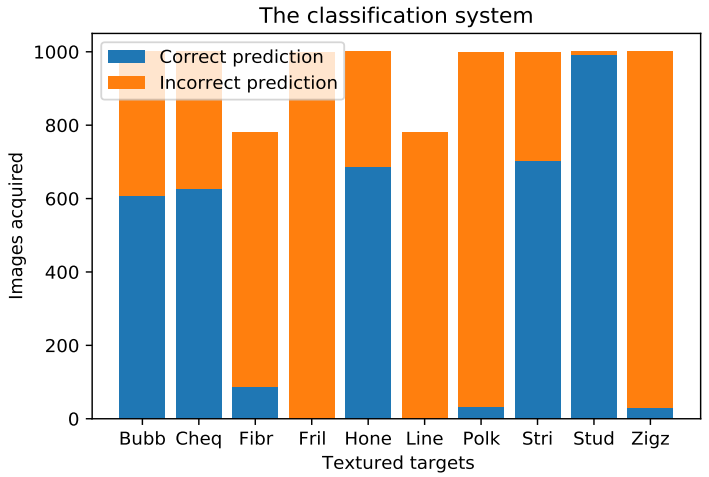
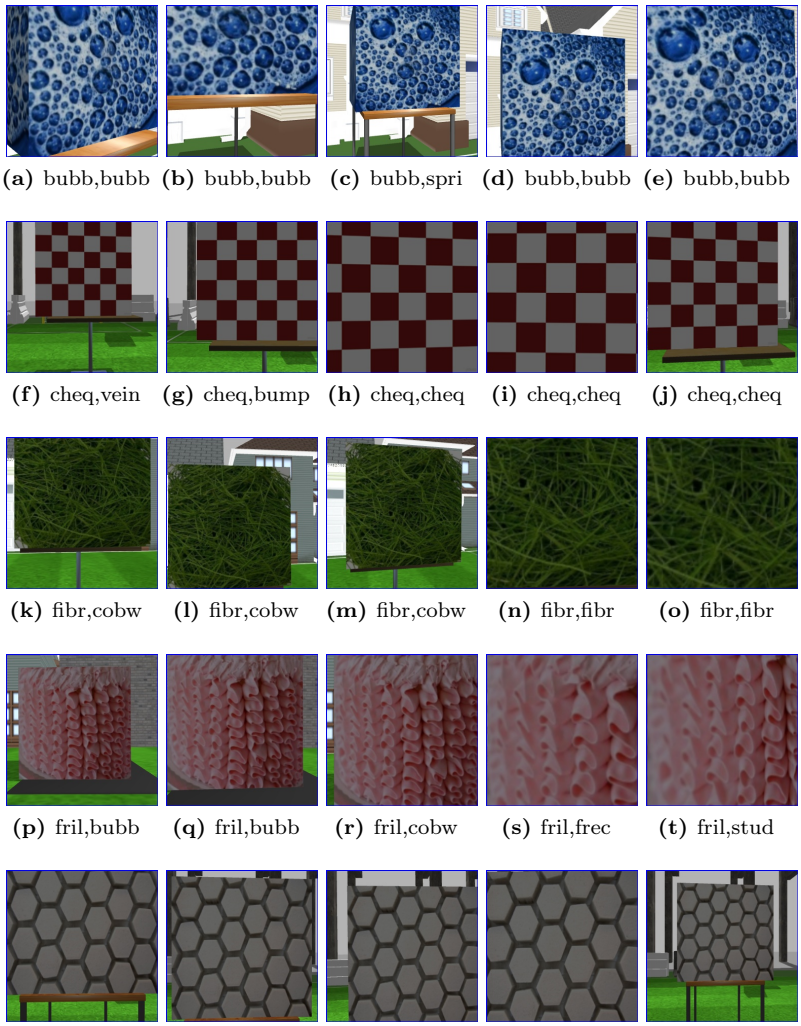


Figure 3.35: The number of images with textures obtained with the onboard camera while flying recognition.

Some images (figures 3.36 & 3.37) of the recognition set are shown, with its original label and its prediction label. However, it is observed that the five test images incorrectly predict the frilly, lined, polka-dotted, and zigzagged set. These five images relate to the whole recognition set, except for fibrous, polka-dotted, and zigzagged, which achieve at least 3% *Accuracy*. This result allows us to understand the generalization of learning between the model and textured objects.

The experimental development is positive because of the ten classes; the worst performing was frilly, with a high correlation with the bubbly, cobwebbed, freckled, and studded classes. In contrast, the fibrous class is highly correlated with the cobwebbed class. On the other hand, lined has a high correlation with the zigzagged, braided, striped, and swirly classes. In the case of the polka-dotted class, it performed poorly. However, it is observed that most of the predicted labels were of the dotted class.



(a) bubb,bubb **(b)** bubb,bubb **(c)** bubb,spri **(d)** bubb,bubb **(e)** bubb,bubb

(f) cheq,vein **(g)** cheq,bump **(h)** cheq,cheq **(i)** cheq,cheq **(j)** cheq,cheq

(k) fibr,cobw **(l)** fibr,cobw **(m)** fibr,cobw **(n)** fibr,fibr **(o)** fibr,fibr

(p) fril,bubb **(q)** fril,bubb **(r)** fril,cobw **(s)** fril,frec **(t)** fril,stud

(u) hone,hone **(v)** hone,hone **(w)** hone,hone **(x)** hone,hone **(y)** hone,hone

Figure 3.36: Image sequence acquired from the on-board camera of the drone. The classification system has a good inference on the texture in the first, second, and fifth rows.

Therefore, the two classes were observed to have almost the same characteristic patterns. In addition, the studded class obtained nearly 100% *Accuracy*. Bubbly, chequered, honey, and striped at least achieved 60% *Accuracy*. On the other hand, the striped class shows a low correlation with the cobwebbed class. Also, the zigzagged class has a high correlation with the meshed class and a slight correlation with the grid class.

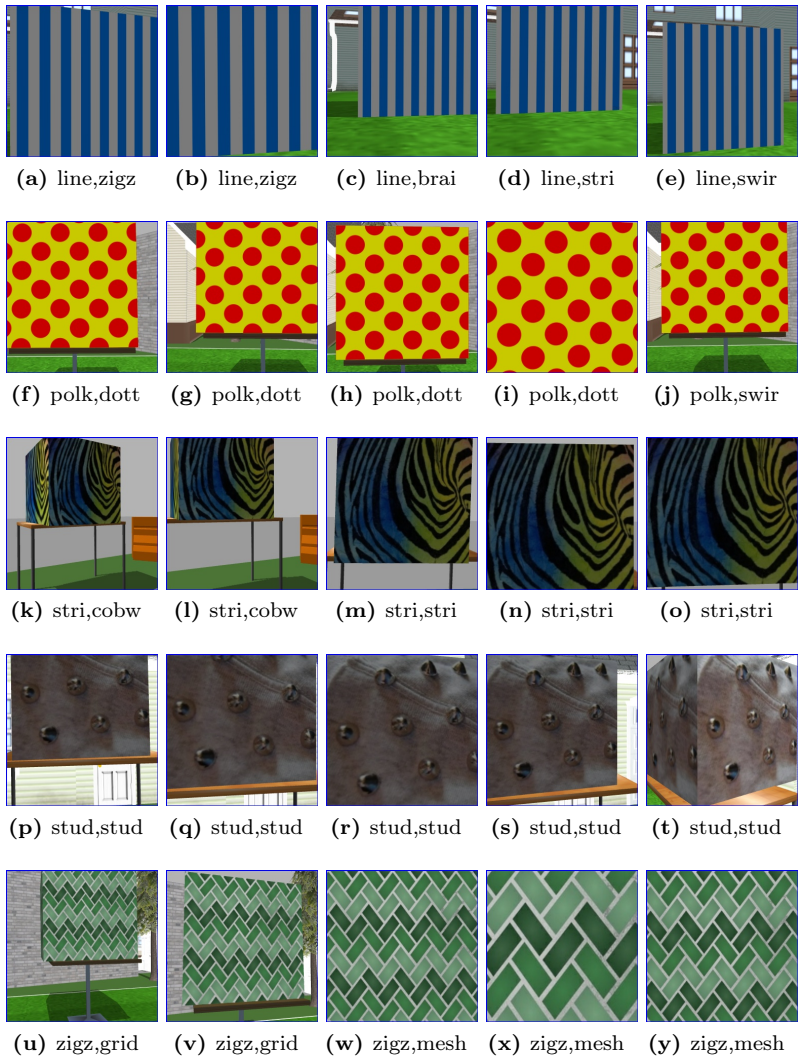


Figure 3.37: Image sequence acquired from the on-board camera of the drone. In the second, third, and fourth rows, the classification system gets good classification performance.

3.4 Texture and Materials Image Classification Based on Wavelet Pooling Layer in CNN.

Convolutional Neural Networks (CNNs) have recently been proposed as a solution to texture and material classification in computer vision. However, inside CNNs, the internal pooling layers often cause a loss of information, which is detrimental to learning architecture. Moreover, when considering images with repetitive and essential patterns, the loss of this information affects the performance of subsequent stages, such as feature extraction and analysis. In addition, it is known that regularization methods focus only on the convolutional layer. In contrast, the operations of the pooling layers have been left without an update [22]. In this approach, a classification system with a new pooling method called Discrete Wavelet Transform Pooling (DWTP) is proposed to solve this problem. This method is based on the image decomposition into sub-bands, in which the first level sub-band is considered its output. The objective is to obtain approximation and detail information. As a result, this information can be concatenated in different combinations. Hence, it is offered three configurations: DWTP = approximation and detail information; DWTaP = approximation information; and DWTdP = detail information. The goal is to preserve the most information for each texture and material.

The method is validated on three datasets: CIFAR-10, Describable Textures Dataset (DTD), and Flickr Material Database (FMD) [36, 37, 39]. The approach is different from traditional methods because it is not a subsampling methodology using neighboring regions, but wavelet pooling maintains its function as a reduction layer. Wavelets allow localization in scale (i.e., frequency) and space. In other words, wavelets can be used to analyze local and spatial transients in the data, such as edges or surfaces in an image [40]. Therefore, they can preserve the most relevant information about textures and materials, sometimes lost with traditional methods such as Max-Pooling (MaxP) and Ave-Pooling (AveP).

The following was presented in the first study, where a CNN architecture was designed for object detection with a repetitive pattern approach in aerial navigation. Where it argued that the characteristics at different frequencies, low and high, also affect the performance of the CNN during training, eliminate overfitting, and achieve higher efficiency in object detection. Based on the results from the previous work, it now presents a wavelet pooling approach to improve the learning of the classification model with the following contributions:

1. It is presented a CNN architecture with a combination of regularization methods (DropOut, Data Augmentation, and Batch Normalization) to evaluate the performance of each pooling method: MaxP, AveP, and wavelet

pooling (DWTP, DWTaP, and DWTdP). The objective is to have a reference for the learning behavior.

2. It is shown that the method eliminates the overfitting created by pooling methods while reducing features using an approach based on level-based decomposition. It is more compact than pooling by using neighboring regions.

3. It is demonstrated that a correct inference of texture or material can be obtained if we determine the type of pooling used during learning. It has conducted several experiments, but now it can choose the best pooling method depending on the dataset. The experiments indicate that this is also useful for future object detection applications, focusing on physical features such as texture.

3.4.1 Materials and Methods.

The design of an effective model for texture and material classification considers several issues: CNN architecture, dataset, regularization methods, model accuracy, and information pooling. The DWTP method mainly focuses on improving the model's classification performance. Moreover, the method reduces the artifacts resulting from dimension reduction in feature maps. The approach preserves the significant features that traditional methods cannot retain. To evaluate the approach (DWTP) and to have the effect of each pooling method concerning the dataset, we outline the main steps below:

1. It is decided to involve digital images containing mainly textures and materials for the CNN training. Textures and materials are key features for evaluating the pooling method against losing information with repetitive patterns.

2. Each dataset being evaluated is divided into training, validation, and test. A higher distribution percentage for the training set and the remaining percentages for the validation and test sets are similar. This is a good practice in state-of-the-art CNNs [4].

3. An approximate version of the VGG16 architecture is used in the CNN design but with only three convolutional blocks [38]. In addition, a classification block is proposed for our research case. The training hyperparameters are described in Table 3.6.

4. The configuration for pooling inside each convolutional block of the CNN (Block.CX) permits a reduction in the feature map. This initial configuration depends on the selection of the pooling method. Therefore, it has at disposal

AveP and MaxP, the proposed DWTP method, and the complementary versions DWTaP and DWTdP.

5. The evaluation stage includes the analysis of the classifier with the *Accuracy* metric because it allows us to evaluate the performance of the model and its learning behavior.

6. Finally, regularization methods are used to improve the performance of the model.

The main contribution is to perform pooling (as a layer) inside the CNN using a level-based decomposition approach. Hence, the proposed approach (DWTP) concatenates the sub-images x_{LL}, x_{LH}, x_{HL} and x_{HH}, given Equation (2.12). From this approach, it is obtained two configurations. The first configuration (DWTaP) uses only the first level approximation sub-band x_{LL}, and the second approach (DWTdP) uses all the first level detail sub-bands. The traditional methods (AveP–MaxP) are implemented with the Keras and TensorFlow methods. The diagram of the proposed methodology is presented in Figure 3.4.

3.4.1.1 Network Training and Parameter Setting.

The algorithms are implemented and developed using the Python language and Keras API with Tensorflow as Backend. Moreover, it is an open-source project, and its manner of programming is sequential through blocks [4]. The hardware specifications of the training device are an Intel® Core™ i7 processor with an NVIDIA GeForce RTX™ 2080 graphics card, 12 GB of RAM, and Ubuntu 18.04 64-bit operating system.

The base architecture is the VGG network, and it is one of the first deep models with good results in a large-scale visual recognition challenge (ILSVRC-2014) with 92.7% top-5 accuracy [38]. This architecture is designed to facilitate the creation of a *classification model*—three convolutional blocks with their pooling layer and one classification stage. The process is as follows: using the base VGG architecture, combined with the pre-processed CIFAR-10, DTD, and FMD datasets, through supervised learning. Before training the CNN, the *Loss function* and the *optimizer* need to be specified. These parameters determine how the network weights should be updated during training. The training parameters for the proposed models have listed in Table 3.6.

A complete analysis is performed with each of the proposed pooling methods: MaxP, AveP, DWTP, DWTaP, and DWTdP. In addition, it is combined with the regularization methods DropOut [73,75,76], Data Augmentation [28,77], and Batch Normalization [78, 79]. In this manner, a learning model is obtained, and it can

predict the objects, textures, and materials in the dataset (test) images with better *Accuracy*.

Table 3.6: Training parameters of the proposed model.

Hyperparameters	
Learning rate	0.001
Minibatch	30, CIFAR-10 = 64
Loss function	'categorical_crossentropy'
Metrics	'acc','loss'
Epochs	500
Callbacks API	4
ModelCheckpoint	Monitor = 'val_loss', save_best_only = True, mode='min'
EarlyStopping	Monitor = 'val_acc', patience = 15, mode = 'max'
CVLogger	'model_history.csv', append = True
ReduceLROnPlateau	Monitor = 'val_los', factor=0.2, patience=10, min_lr = 0.001
Optimizer	SGD—Adam

3.4.1.2 Benchmark Dataset.

In classification tasks, the model must be evaluated on a dataset. It has performed our experiments on three datasets. The first dataset is CIFAR-10 [39], the second one is the Describable Textures Dataset (DTD) [36], and the last one is Flickr Material Database (FMD) [37]. CIFAR-10 consists of 60,000 images of 32×32 pixels of ten different objects. DTD contains 47 classes of 120 images in the wild. This dataset is developed in different uncontrolled conditions. Initially, it includes 40 training images, 40 validation images, and 40 test images for each class. Finally, FMD is built with standard materials. It has ten classes of 100 images, and each image is hand-picked from Flickr (under Creative Commons license) to ensure various lighting conditions, compositions, colors, texture, and material subtypes.

A good practice is to split our dataset using the Hold-Out Cross-Validation sampling technique [4,14]. In the case of CIFAR-10, the test set is initially with approximately 16.66%, and the training set is divided into two subsets with the same distribution of images: training 80% and validation 20%. For DTD and FMD, the distribution is different because the dataset is small. The test set contains 15% of the data. Therefore, the rest is divided into the training subset with 82% and the validation subset with 18% of the data.

The images have dimensions of 224×224 pixels, except for the CIFAR-10 dataset, which has dimensions of 32×32 pixels. Following convention, it is helpful to normalize the pixel values to a range of 0 to 1 for our model to converge quickly because the inputs with large integer values can slow down the learning process. The number of images per class is shown in Table 3.7. As observed, the last two datasets have a few images, but one advantage is that they have a balance between the number of images per class.

Table 3.7: The number of images per class.

Dataset	Classes	Images per Class	Training	Validation	Test
CIFAR-10	10	10,000	40,000	10,000	10,000
DTD	47	120	3,931	863	846
FMD	10	100	700	150	150

3.4.2 Experimental Results.

The different classification models created allow us to analyze the contribution of wavelet pooling; in this case, the model can analyze images with objects, textures, and materials. It can also observe the learning curve of the proposed pooling methods. Furthermore, it is incorporated regularization methods for image classification to improve the model's learning capability. The experiments obtained using the three proposed regularization techniques are shown in Figures A.1 and A.2 in Appendix A.1, based on the VGG architecture and the pooling method. In this manner, a complete analysis of the performance of the classifier is provided.

In order to perform efficiency testing of each pooling method on each dataset, it is used an initial configuration where each pooling layer inside the architecture has only one pooling method at a time. All pooling methods use a 2×2 window to perform the comparison with the proposed method.

3.4.2.1 Image Classification CIFAR-10.

The first dataset used is CIFAR-10, with a set of 60,000 images. Table 3.8 shows that the proposed method outperforms traditional methods. In this sense, the DWTaP combination uses only the approximation information. In addition, it retained the number of parameters to be trained. Figure 3.38 shows the learning curves of the pooling methods for CIFAR-10. In this case, it is observed that

MaxP and DWTaP resist overfitting; moreover, it shows a slower tendency to learn in both sets. AveP maintains a consistent learning progression in both sets, but *Accuracy* does not improve after epoch 50. In DWTP, it shows the smoothest drop-in learning. It also achieves the best *Accuracy* performance for the training set. DWTdP shows a rapid decrease during learning, which does not resist overfitting after epoch 70.

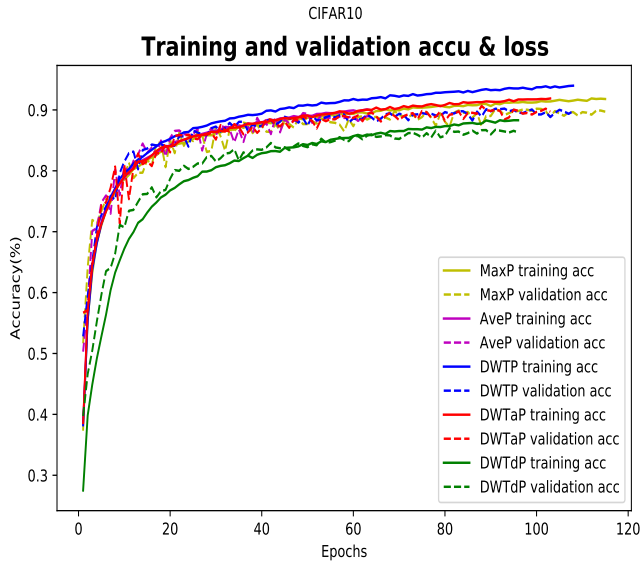
Table 3.8: Performance of pooling methods on CIFAR-10.

Method	Trainable Params	Loss	Acc	Val_ loss	Val_ acc	Test_ loss	Test_ acc
MaxP	545,206	0.2741	0.9069	0.3058	0.8990	0.3365	0.8913
AveP	545,206	0.3220	0.8906	0.3296	0.8932	0.3493	0.8850
DWTP	1,558,966	0.1958	0.9330	0.3181	0.9020	0.3461	0.8946
DWTaP	545,206	0.2568	0.9126	0.2970	0.9067	0.3208	0.8970
DWTdP	1,221,046	0.3678	0.8735	0.4040	0.8701	0.4207	0.8672

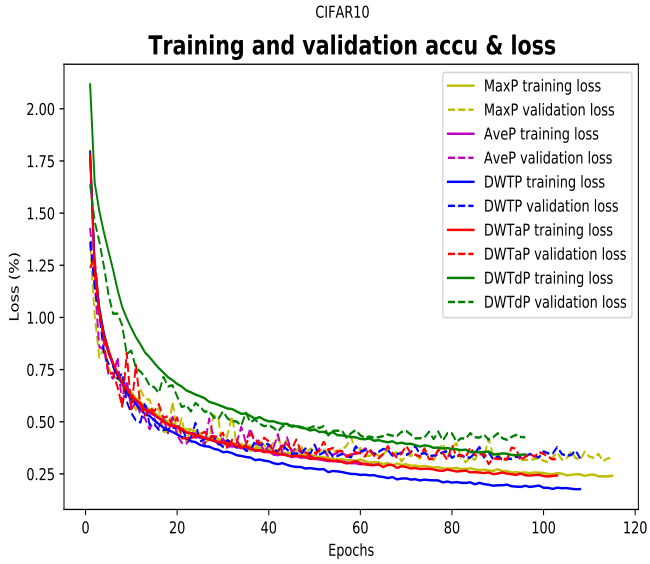
The correlation of each class with their actual and predicted label for each model is shown in Figure A.3 in Appendix A.2, which shows the *multiple confusion matrix*. Moreover, the classification report with the evaluation metrics for CIFAR-10 is shown in Table A.1 of Appendix A.3.

3.4.2.2 Image Classification with Textures DTD.

The second dataset used is DTD, with 47 classes of different textures. It has only 120 images for each category, which may cause overfitting in the model. Thus, the proposed method is also a solution when you have a small dataset. In this case, two experiments are performed by varying the training optimizer. First, it used SGD as the optimizer [63]. Table 3.9 shows that the proposed DWTP method using its DWTaP configuration outperforms all the methods. In addition, it retains similitude in all three sets: training (37.40%), validation (31.17%), and test (34.16%). Figure 3.39 shows the learning curves of the pooling methods for DTD. MaxP shows a smooth learning decay and similar behavior between both sets in this case, but the learning rate does not improve after epoch 71. AveP and DWTP maintain a consistent learning progression, and their validation sets progress at a similar rate but does not resist overfitting. DWTaP shows identical behavior to AveP and DWTP. The model resists overfitting, achieving the best *Accuracy* performance in both sets. DWTdP practically does not resist overfitting; in this case, the validation set shows an increase in the error of the cost function. The DWTaP model obtained with this configuration is shown in Figure A.4 of Appendix A.2, which shows the correlation of each class with its actual and predicted label. Based on this result, it is decided to use the following optimizer to improve classification performances.



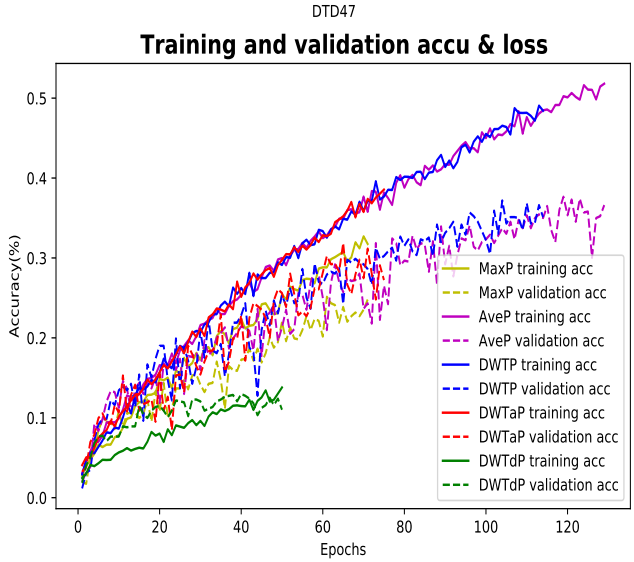
(a) Accuracy



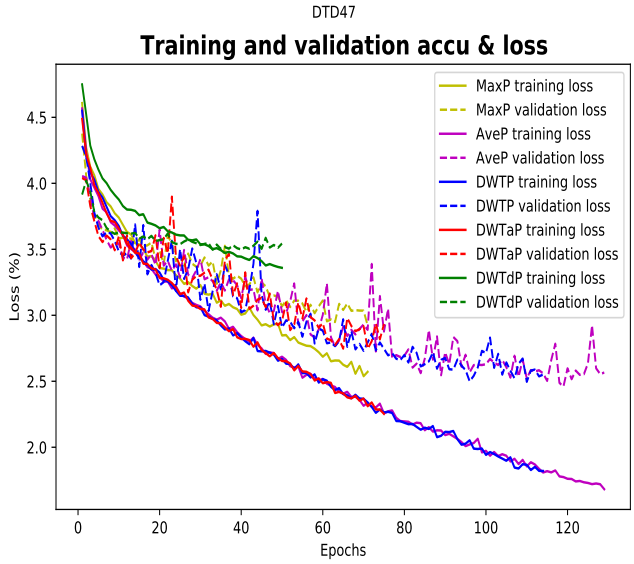
(b) Loss

Figure 3.38: Learning behavior on CIFAR-10 training and validation sets.

.



(a) Accuracy



(b) Loss

Figure 3.39: Learning behavior on DTD training and validation sets—SGD optimizer.

.

Table 3.9: Performance of pooling methods on DTD—SGD optimizer.

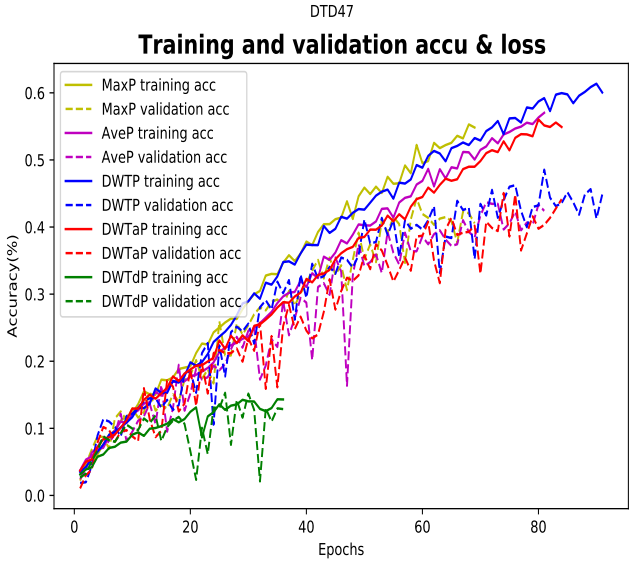
Method	Trainable Params	Loss	Acc	Val_ loss	Val_ acc	Test_ loss	Test_ acc
MaxP	12,344,831	2.5715	0.3176	2.9217	0.2480	2.8203	0.2742
AveP	12,344,831	1.7849	0.5024	2.4606	0.3766	2.3937	0.3865
DWTP	48,748,031	1.8842	0.4817	2.4958	0.3685	2.4415	0.3924
DWTaP	12,344,831	2.3214	0.3740	2.7459	0.3117	2.6390	0.3416
DWTdP	36,613,631	3.5035	0.1069	3.4968	0.1136	3.4817	0.1288

The second experiment uses an Adam optimizer — an extension of stochastic gradient descent [65]. Table 3.10 shows that the proposed DWTaP method and MaxP exhibit the best classification performance on all three data sets. In this case, it considers a change in the optimizer that resulted in an essential factor for learning MaxP. Figure 3.40 shows the learning curves of the pooling methods for DTD. MaxP shows a smooth learning decay and similar behavior between the two sets. It also resists overfitting, managing to have good *Accuracy* performance. AveP and DWTP maintain a consistent learning progression, and their validation sets progress at a similar rate but does not resist overfitting. The learning rate of DWTaP resists overfitting in both sets, achieving one of the best *Accuracy* performances. DWTdP shows a slow learning behavior; thus, the learning rate does not improve after epoch 28.

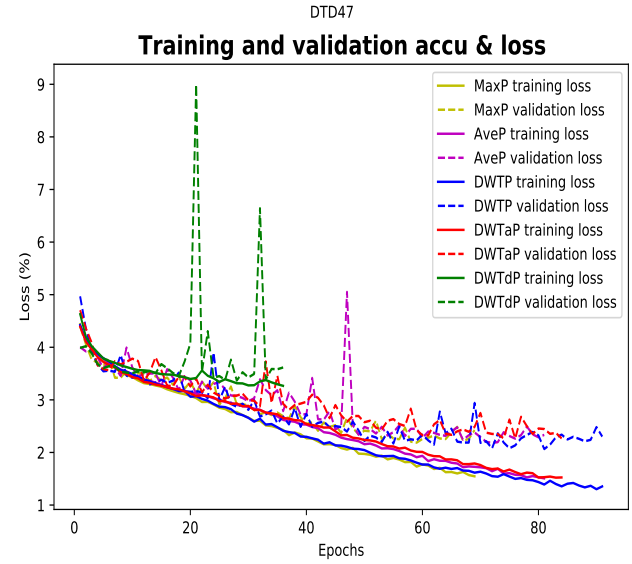
Table 3.10: Performance of pooling methods on DTD—Adam optimizer.

Method	Trainable Params	Loss	Acc	Val_ loss	Val_ acc	Test_ loss	Test_ acc
MaxP	12,344,831	1.7423	0.5225	2.1863	0.4426	2.1376	0.4350
AveP	12,344,831	1.6607	0.5324	2.1816	0.4345	2.1934	0.4184
DWTP	48,748,031	1.4055	0.5922	2.0647	0.4855	2.0195	0.4799
DWTaP	12,344,831	1.6408	0.5329	2.2657	0.4484	2.2878	0.4302
DWTdP	36,613,631	3.3241	0.1356	3.3666	0.1425	3.3205	0.1536

The DWTaP and MaxP learning models obtained with this configuration are shown in Figure A.5 of Appendix A.2, which summarizes the level of success of the classification model predictions. Moreover, the classification reports obtained with both configurations (SGD and Adam) for DTD are shown in Appendix A.3 and Tables A.2 and A.3.



(a) Accuracy



(b) Loss

Figure 3.40: Learning behavior on DTD training and validation sets—Adam optimizer.

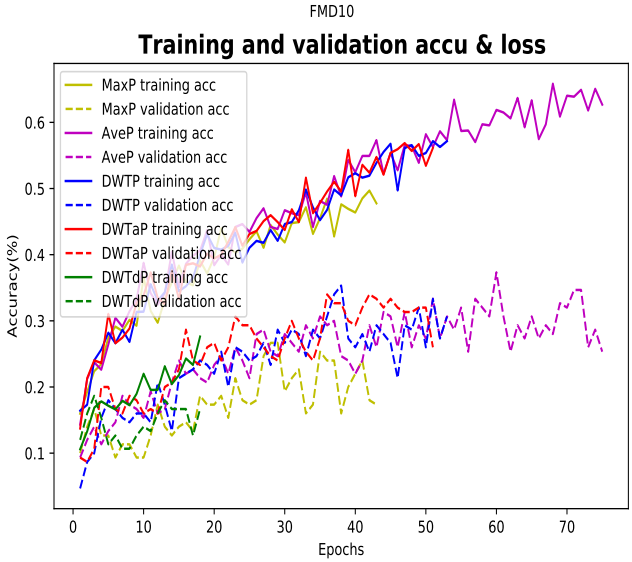
3.4.2.3 Image Classification with Materials FMD.

The third dataset used is FMD, with ten classes of different materials. Moreover, it is a small dataset since it only has 100 images per class. Likewise, it performed two experiments: the first with the SGD optimizer and the second with the Adam optimizer [63, 65]. Table 3.11 shows that the proposed DWTP method using its DWTdP configuration outperforms all methods. In addition, the model retains a similitude in the three sets: training, validation, and test. Figure 3.41 shows the learning curves of the pooling methods for FMD. In this case, MaxP presents overfitting rather quickly. AveP maintains a smooth descent for the training set, but the validation set does not avoid overfitting. DWTP maintains a consistent learning progression for its two sets, but they do not resist overfitting. The learning rate of DWTaP does not resist overfitting in both sets, having similar behavior as DWTP. Finally, DWTdP shows a consistent progression in its first learning epochs, and its ensembles progress at a similar rate, thus achieving the best performance for this dataset. The DWTdP model obtained with this configuration is shown in Figure A.6 of Appendix A.2, which shows the correlation of each class with its actual and predicted label.

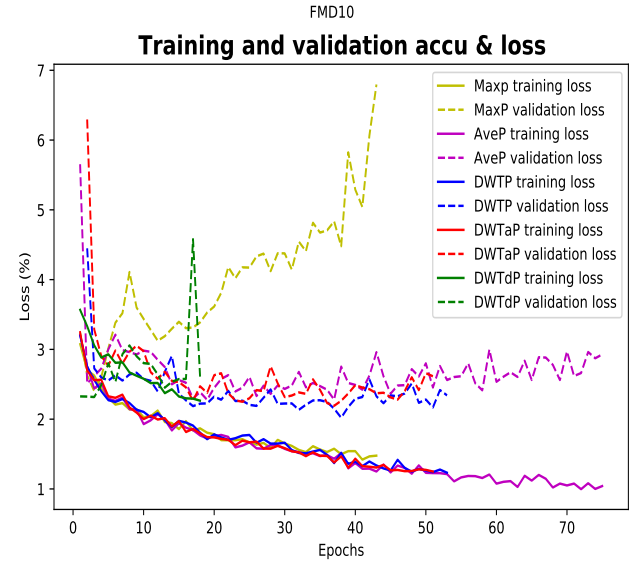
Table 3.11: Performance of pooling methods on FMD—SGD optimizer.

Method	**Trainable Params**	**Loss**	**Acc**	**Val_ loss**	**Val_ acc**	**Test_ loss**	**Test_ acc**
MaxP	12,341,686	2.6234	0.2239	2.4013	0.1667	2.4426	0.1333
AveP	12,341,686	1.7555	0.4369	2.2773	0.2067	2.3420	0.2467
DWTP	48,744,886	1.5113	0.4896	2.0208	0.3533	2.1488	0.2867
DWTaP	12,341,686	1.3802	0.5101	2.1916	0.3267	2.3301	0.3066
DWTdP	36,610,486	3.0464	0.1687	2.3172	0.1867	2.4176	0.1400

The change of the optimizer, in this case, was beneficial for AveP learning. Table 3.12 shows that the proposed DWTdP method and AveP exhibit the best classification performance on all three datasets. Figure 3.42 shows the learning curves of the pooling methods for FMD. In this case, MaxP shows a smooth learning descent and similar behavior between the two sets, but after epoch 22, it does not resist overfitting. AveP achieves the best performance at epoch 17, avoiding overfitting in the following epochs. DWTP and DWTaP maintain a consistent learning progression, and their validation sets progress at a similar rate but does not resist overfitting. DWTdP shows a slow learning trend in the early epochs, but after epoch 15, the learning rate improves, and the sets evolve at a similar rate, achieving good *Accuracy* performance.



(a) Accuracy



(b) Loss

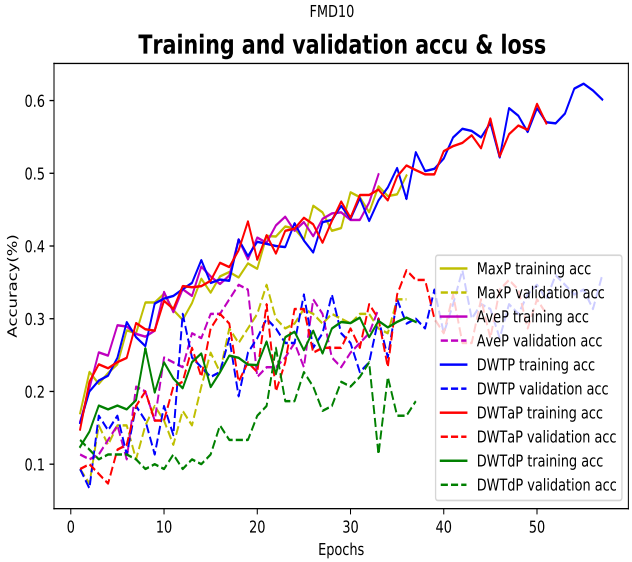
Figure 3.41: Learning behavior on FMD training and validation sets—SGD optimizer.

.

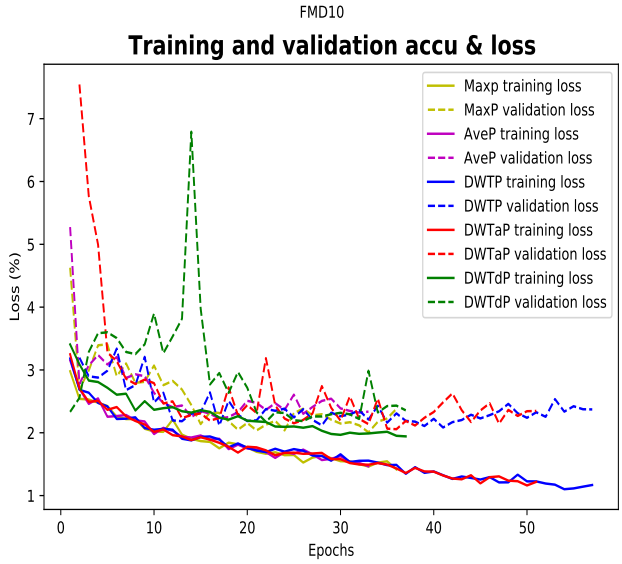
Table 3.12: Performance of pooling methods on FMD—Adam optimizer.

Method	Trainable Params	Loss	Acc	Val_ loss	Val_ acc	Test_ loss	Test_ acc
MaxP	12,341,686	1.4713	0.4821	2.0068	0.2867	2.1222	0.2867
AveP	12,341,686	1.9043	0.3594	2.2249	0.3267	2.0981	0.3000
DWTP	48,744,886	1.3116	0.5493	2.0832	0.3200	2.0667	0.3133
DWTaP	12,341,686	1.4319	0.5108	2.0566	0.3667	2.0717	0.2866
DWTdP	36,610,486	2.1660	0.2239	2.1728	0.2600	2.2071	0.2199

The DWTdP and AveP learning models obtained with this configuration are shown in Figure A.7 of Appendix A.2, which summarizes the level of success of the classification model predictions. Moreover, the classification reports obtained with both configurations (SGD and Adam) for FMD are shown in Tables A.4 and A.5 of Appendix A.3.



(a) Accuracy



(b) Loss

Figure 3.42: Learning behavior on FMD training and validation sets—Adam optimizer.

3.5 Discussion

This section presents the discussion of results and observations for each approach.

First approach: CNNs have impacted classification, detection, and localization tasks in area navigation. Two virtual worlds are illustrated in Figures 3.15 and 3.17. These have allowed the generation of two new datasets. This further information allows for validating the use of the wavelet transform. In addition, the created dataset is allowed to become a benchmark for aerial object recognition in the deep learning community. Furthermore, Table 3.1 contains the results of comparing both experiments on the test set, where the *Accuracy* and *Loss* metrics are evaluated. The results show that the proposals have very similar learning. However, this value differs from the learning behavior in the training and validation sets, see Figures 3.7-3.10. Therefore, the model in the spatial domain has been overfitted. Moreover, Table 3.13 shows that the detection model has an excellent performance in prediction time. This value is lower than the time the image is published in the topic into the ROS framework. In particular, this time value depends on the network, depth, image size, number of neurons in the last layer, and the hardware where it is implemented.

Table 3.13: Experimental results of running our application on the ROS framework and Gazebo simulator.

Topic	Value [s]
Prediction function	0.01608
Image_raw	0.02587

The main observations are: (a) Considering scaled information in the wavelet domain at the training stage eliminates overfitting; (b) using the information in the wavelet domain for CNN training is beneficial to reducing the computational cost; and (c) although some experimental tests have good detection performance, the analysis is affected by the classification latency of the model. However, using an NVIDIA graphics card is feasible to optimize the application performance.

Second approach: CNN can be a universal feature extractor. However, it is unclear whether it can process spectral information (information for texture analysis) in practice. It is shown in Table 3.2 that the test set in KT2B texture datasets is 96%, DTD is 34%, and FMD with 30%. These results are significant, given that these are images that the model has never seen. Besides, although the metrics in training and validation are given at different times, similar behavior is shown with the test data. So, there is a learning generalization between the three training sets.

Table 3.14 shows a summary of the achieved performance of the classification system, as well as a comparison against AlexNet [31], Texture CNN [19], and Wavelet CNN [21]. Furthermore, the FMD dataset is proposed to validate the learning model. Therefore, based on the results, it can be seen that the model offers a generalization of learning to other datasets.

Table 3.14: Classification results and comparison with other state-of-the-art architectures in terms of accuracy (%).

	AlexNet	**T-CNN**	**Wavelet CNN**	**Proposal**
DTD	22.7	27.8	35.6	**34.5**
KT2B	48.3	49.6	63.7	**96.7**
FMD	–	–	–	**30.0**

In general, the performance of the classifier is evaluated by taking into account the fusion of the information. The main observations are: (a) the proposed model generalizes its learning to other datasets; and (b) extracting features using the wavelet transform, which feeds spectral information to the CNN.

Third approach: Sometimes there is not enough information to train CNNs, and sometimes the datasets are tiny. Despite this, transfer learning is a methodology that can help to improve the learning capacity of the classification model. Table 3.3 shows the analysis of the experiment; in this case, it is observed that the network trained from scratch shows overfitting, as opposed to using the pre-trained network. Also, it is essential to mention that the wavelet features provide homogeneity in the performance of the classifier. Moreover, Table 3.15 summarizes the achieved performance of our classification system, as well as a comparison to AlexNet (trained from scratch) [31], T-CNN [19], and Wavelet CNN [21]. In particular, it is observed that through the transfer learning and wavelet features, the performance of the classifier is improved when having a small dataset.

Table 3.15: Classification results and comparison with other state-of-the-art pre-trained architectures with ImageNet, in terms of accuracy (%).

	AlexNet	**T-CNN**	**Wavelet CNN**	**New model**
DTD	22.7	55.8	59.8	**53.19**

Fourth approach: Even though CNNs have established their position in image analysis and the different elements considered to improve classification performance and are well known in the literature, only a few experiments have been conducted by considering the pooling layers. In Figures 3.38–3.39 and 3.41, it illustrates the learning behavior of each model and for each pooling method. Figures clearly show

that the DWTP versions of the model behavior in both training sets are uniformly distributed. The learning curve remains stable and shows a similar generalization between the three training sets. When the optimizer change is proposed (Figures 3.40 and 3.42), the results are very similar to the DWTP versions. Moreover, it achieves increased classification performance for both the proposed version and the traditional methods.

Furthermore, Table 3.16 contains the results of comparing our proposals with other methods proposed by Fujieda *et al.* [21] and Andrearczyk *et al.* [19], where the *Accuracy* rate for the models trained from scratch on the DTD dataset is evaluated. The bold values shown in Table 3.16 indicate that the results are quite comparable with those of the other methods. The results show that the proposals are computationally lightweight. Generally, it can observe the algorithm's efficiency for CIFAR-10 in Table 3.8. This dataset can be compared in the literature because it is one of the most important in the deep learning area. As for FMD, it is mentioned that there are algorithms with a performance above that obtained; however, it differs from the central concept in combining both approaches and considering the wavelet pooling method.

Table 3.16: Performance evaluation and comparison with other methods indicated as accuracy (%)—DTD dataset.

Topic	**T-CNN**	**Wavelet CNN**	**Test1**	**Test2**	
			DWTaP	**MaxP**	**DWTaP**
Trainable params (millions)	23.4	14.1	12.3	12.3	12.3
DTD (%)	27.8	35.6	34.16	**43.5**	**43.02**

Besides, Tables 3.9–3.12 show that the *Loss* metric achieves a high index in the training sets compared to Table 3.8; this learning behavior is because the sets being evaluated are different. In this case, CIFAR-10 accounts for more than 1000 images per class, unlike for the sets with small data such as DTD and FMD. Therefore, the size of the dataset is one more parameter to consider for the contribution of the research, where overfitting is prevented, and it maintains a similitude in the *Accuracy* of the model. In this context, the impact of considering DWTP and its different configurations inside CNN learning is analyzed through the different experiments. The main observations are: (a) To consider a DWTP configuration in the learning stage that presents a learning uniformity; (b) the use of a DWTP configuration to reduce the number of features is desirable to preserve relevant information; and (c) although some tests upon optimizer change have a good response towards other methods. The DWTP method also increases its classification performance. However, note that this approach depends on the dataset's type.

4

Conclusions.

In this chapter, the conclusions of the research project, the perspective of experiments, and future work are described.

Contents

4.1 Conclusions.

This research has presented some approaches to the extensive use and variety of wavelet transform applications, which, together with deep learning techniques, can optimize its performance and contribute to solving tasks for detecting objects in digital images. There are several specific contributions to this research work.

First approach:

- In order to classify the information received by the drone camera and detect if the object is located within the image plane, it was possible to establish two datasets in the spatial domain (original image) and in the wavelet domain (image preprocessed with the two-Dimensional Discrete Wavelet Transform), which makes it possible to analyze the information and apply deep learning on the characteristic patterns based on the training of a Convolutional Neural Network. In the case of wavelet information, this reduces the number of parameters trained within the network and maintains the relevant characteristics in each captured image. This allows having a distinction and a high performance in classification by the binary classifier (model trained with CNN). The classifier is successful with both training sets. Still, using the set in the wavelet domain significantly improves the detection performance, as opposed to having images in the spatial domain as input to the ConvNet architecture. In addition, the wavelet dataset was useful in mitigating overfitting by generalizing the learning during the training stage.
- Information in the wavelet domain is known to improve the learning of the Convolutional Neural Network because the learning model achieved high binary classification performance. The use of this new method of textured object detection has been adapted in the field of aerial navigation. The images acquired from the on-board camera of the drone are classified frame by frame. The acquired information is transformed to the wavelet domain in conjunction with the ROS system and the learning model. This three-level transform exhibits important approximation and detail characteristics. These characteristics maintain an energy distribution, with the approximation dataset retaining most of the properties of the original image.
- To conclude, the application has a high *Accuracy* score, where the prediction output shows the two possible combinations in the scene; textured object (Texture) or not textured object (NotTexture). It is essential to mention that the system predicts almost twice the transmission of the camera frames.

Second approach:
In this work, spectral analysis can be incorporated into the CNN architecture, according to some architectures reported in the literature.

- It was possible to establish the reconstruction of the convolutional layers, the way to generalize the learning, and the reduction of the feature map of a classification system based on deep learning.
- From the spatial information and the two feature maps in the spectral domain, it was possible to feed the three-input architecture - one model. In particular, the fusion of the additionally created feature maps does not limit the learning to spectral features.
- The learning model has improved texture classification *Accuracy*, using a smaller number of parameters to train concerning the existing models.
- The training learning behavior analysis shows that textured features generated additionally may be helpful for CNN architectures, mainly when using small datasets.

Third approach:
Localization and object detection tasks using visual information are challenging, especially when objects exhibit repetitive texture. However, these tasks open up the opportunity for various applications using Micro Aerial Vehicles equipped with on-board cameras for object detection and recognition, e.g., for package pickup, place recognition, landing zone detection, and many more.

- It was possible to apply spectral analysis in combination with deep neural networks to improve the detection and recognition stage. In particular, in this proposal, it was proposed to use the spectral feature maps (additionally created) with the prior learning of the CNN.
- It has been demonstrated that the model applied achieves the elimination of overfitting and higher *Accuracy* in texture classification with a minimum increase in the number of parameters to be trained.
- Tests performed on the simulation shows some interesting results. The prediction model shows the creation of a generalized perception of the texture attached to the objects. Furthermore, despite having a low classification rate, the model is shown to classify most of the test classes correctly.

Fourth approach:

- One of the ideas was to implement wavelet pooling, a method capable of preserving helpful information to improve the performance of texture and material classification in images. Wavelet pooling is introduced inside the proposed VGG architecture as a layer. This layer performs the same function

as the traditional methods; however, the difference is that instead of using a subsampling technique on neighborhood regions, this technique is based on the multilevel decomposition of the input image using wavelet analysis. As a result, four new subsets of features contribute to model learning: approximation, vertical details, diagonal details, and horizontal details.

- Using the wavelet pooling method was shown to achieve acceptable classification performance. Moreover, wavelet pooling performs matching and outperforms some traditional methods used in CNN learning.
- The proposed method outperforms all other pooling sets, e.g., for the CIFAR-10 dataset, it achieves 89.70% on the test set. The DTD dataset shares a similar performance when changing the optimizer with 43%. In the case of the FMD set, the performance achieved was 22% in the detailed version and 30% with the Ave method, possessing similarities in its three training sets. Integrating DropOut, Data Augmentation, and Batch Normalization also positively react to the proposed methods, improving the classification performance.
- The proposed methodology in its decomposition stage can result in a better reduction in image features. In addition, sub-bands at different levels can be considered in learning and could result in better *Accuracy*. The results show that some methods perform better than others depending on the dataset, hyperparameter configurations, and the design of the CNN architecture.
- On the other hand, CNN is characterized by the random aspect in the election of filters of the convolution layers. Therefore, as a further investigation, we can add stability to the selected filters inside the pooling layer.

4.2 Future work.

The results obtained in the first approach of this work are promising. Thus establishing a new opportunity in the evaluation and implementation of feature extraction methods, taking into account new wavelet features to extend object detection, improve texture characterization, and especially couple it to a much more complex autonomous navigation system. While in the second approach of this work can be deepened by selecting other texture features and CNN architectures in pattern recognition in image restoration, classification tasks, and object detection applications. Also, based on this work and the third approach, tests can be performed in real-world scenarios. The fourth approach to the use of wavelet

pooling will allow in the future to test other texture features and change the wavelet base to analyze which base works best for pooling.

Moreover, aerial robotics can apply the proposed architecture and pooling method in pattern recognition, classification tasks, and object detection. Therefore, this is ideal for designing an object classification system for aerial navigation, where the main feature is the analysis of repetitive patterns such as textures. Furthermore, it will investigate possible methods to improve the architecture in order to reduce computational costs while preserving classification performances.

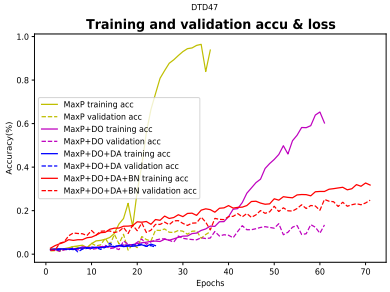
A

Appendix.

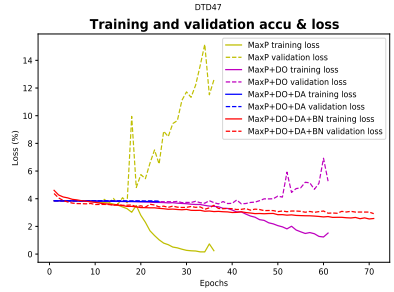
Contents

A.1 Training Process Using Regularization Techniques and Pooling.

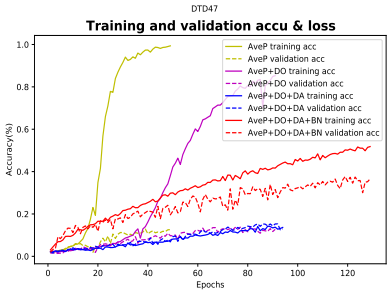
A.1.1 DTD Dataset.



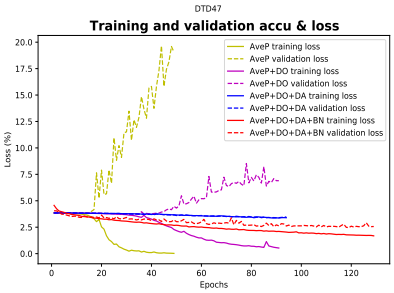
(a) MaxPooling-Acc



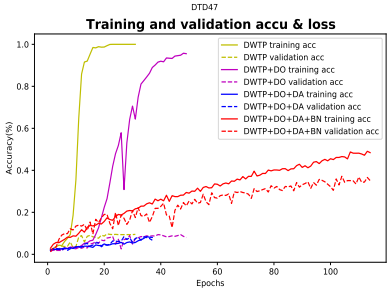
(b) MaxPooling-Loss



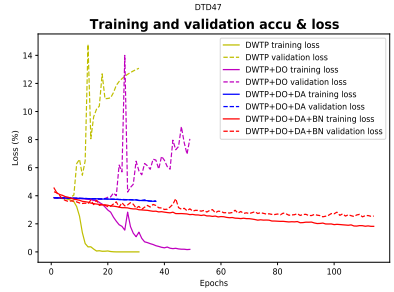
(c) AvePooling-Acc



(d) AvePooling-Loss

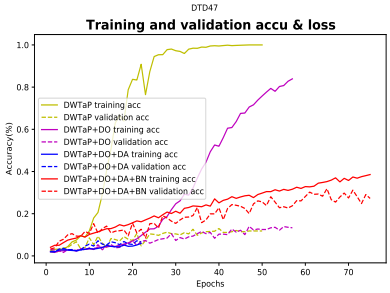


(e) DWTPooling-Acc

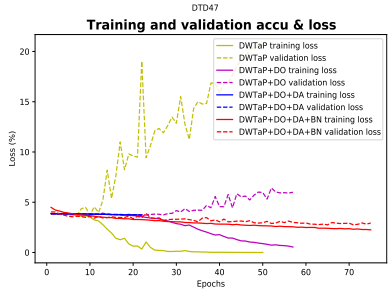


(f) DWTPooling-Loss

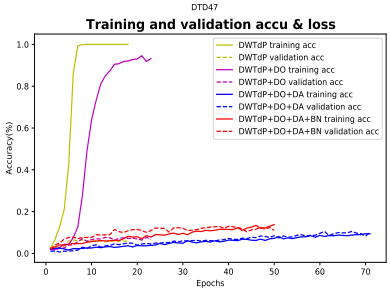
Figure A.1: *Cont.*



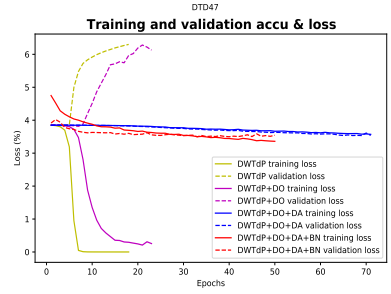
(g) DWTaPooling-Acc



(h) DWTaPooling-Loss



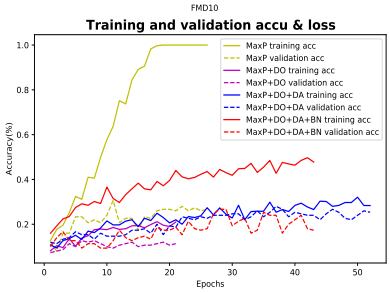
(i) DWTdPooling-Acc



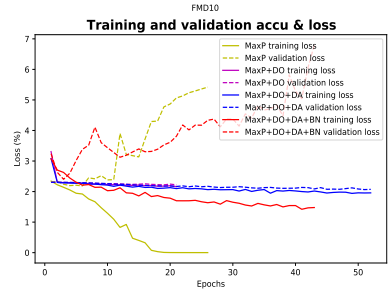
(j) DWTdPooling-Loss

Figure A.1: Learning behavior for baseline architecture + pooling, increasing DropOut, Data Augmentation, and Batch Normalization—DTD dataset.

A.1.2 FMD Dataset.

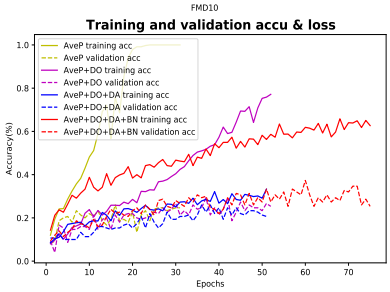


(a) MaxPooling-Acc

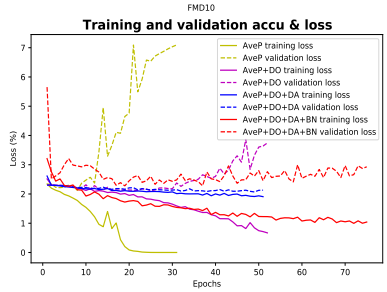


(b) MaxPooling-Loss

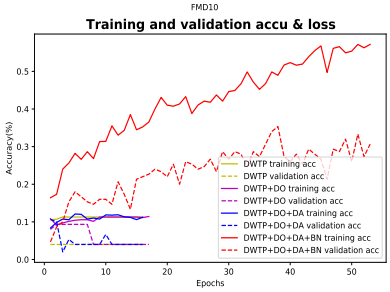
Figure A.2: *Cont.*



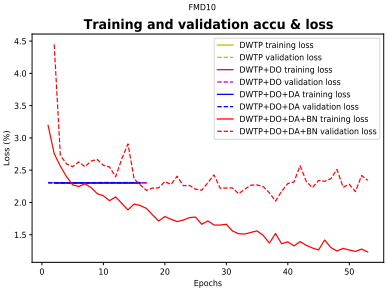
(c) AvePooling-Acc



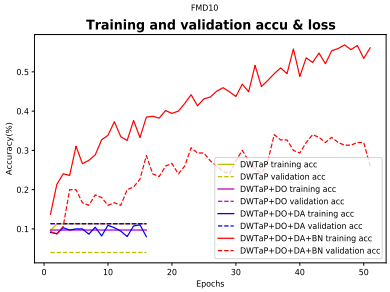
(d) AvePooling-Loss



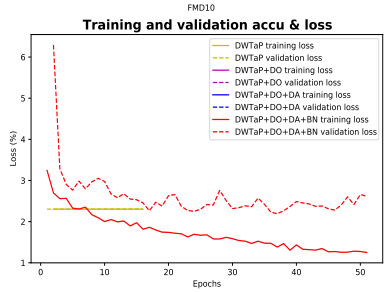
(e) DWTPooling-Acc



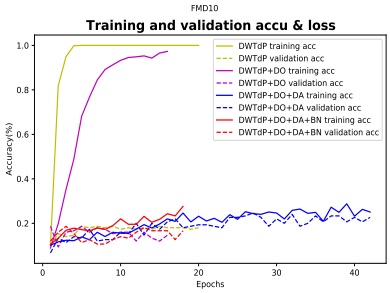
(f) DWTPooling-Loss



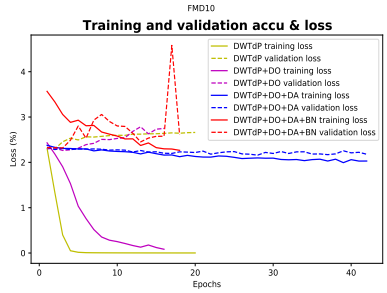
(g) DWTaPooling-Acc



(h) DWTaPooling-Loss



(i) DWTSPooling-Acc



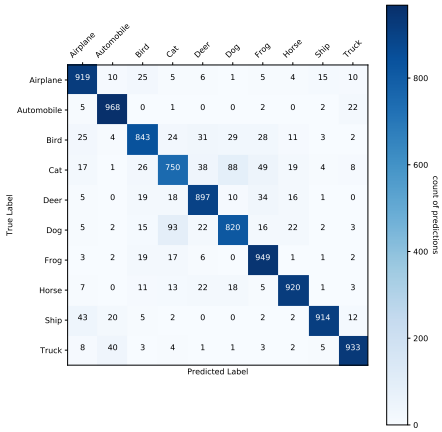
(j) DWTdPooling-Loss

Figure A.2: Learning behavior for baseline architecture + pooling, increasing DropOut, Data Augmentation, and Batch Normalization—FMD dataset.

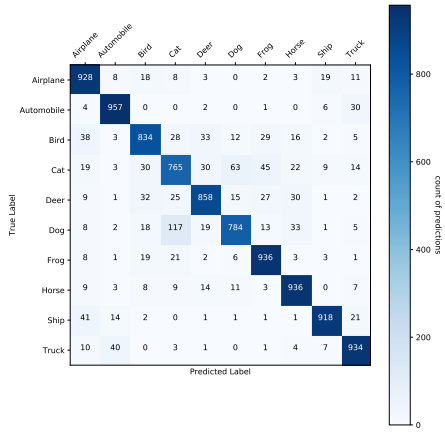
A.2 Multiple Confusion Matrix.

The multiple confusion matrix is an N×N table that summarizes the level of success in the predictions of a classification model: that is, the correlation between the label and the classification of the model.

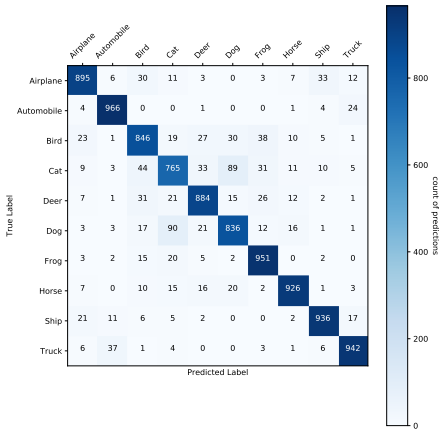
A.2.1 CIFAR-10 Dataset



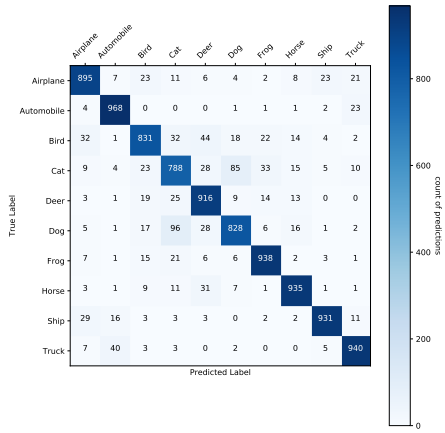
(a) Max



(b) Ave

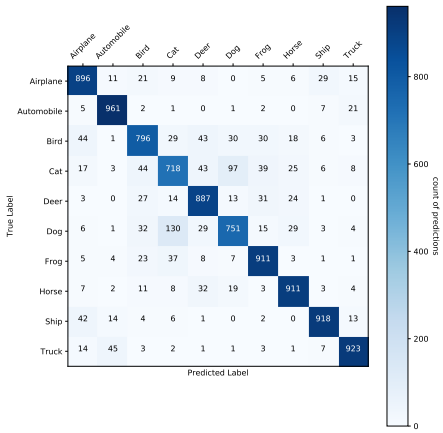


(c) DWT



(d) DWTa

Figure A.3: *Cont.*



(e) DWTd

Figure A.3: In this case, each confusion matrix correlates with the five models obtained for the CIFAR-10 dataset.

A.2.2 DTD Dataset.

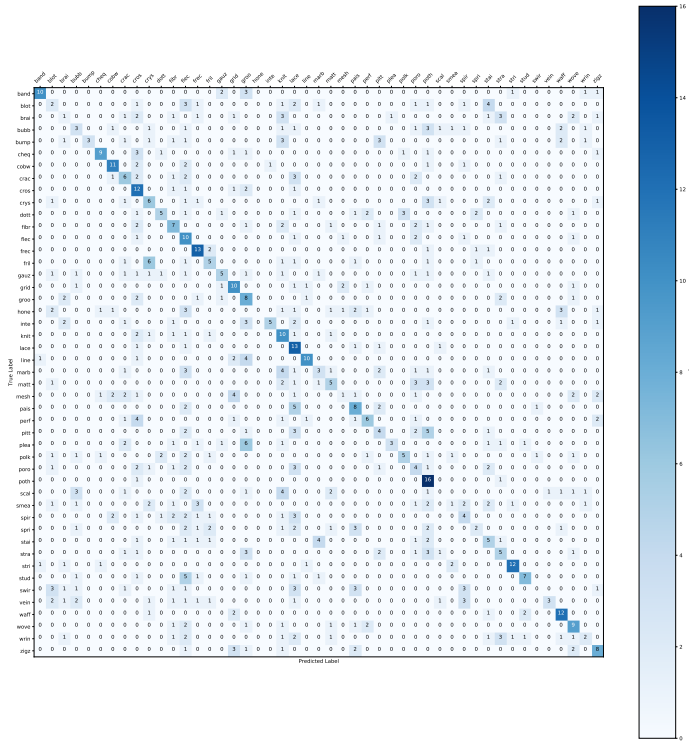
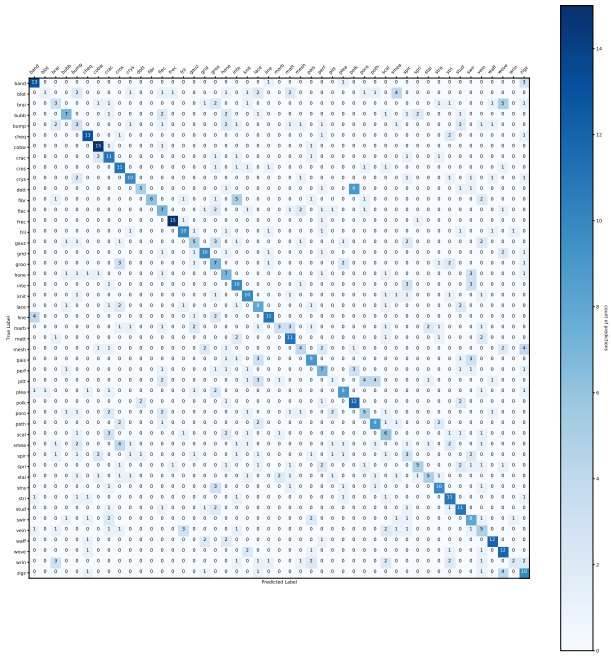
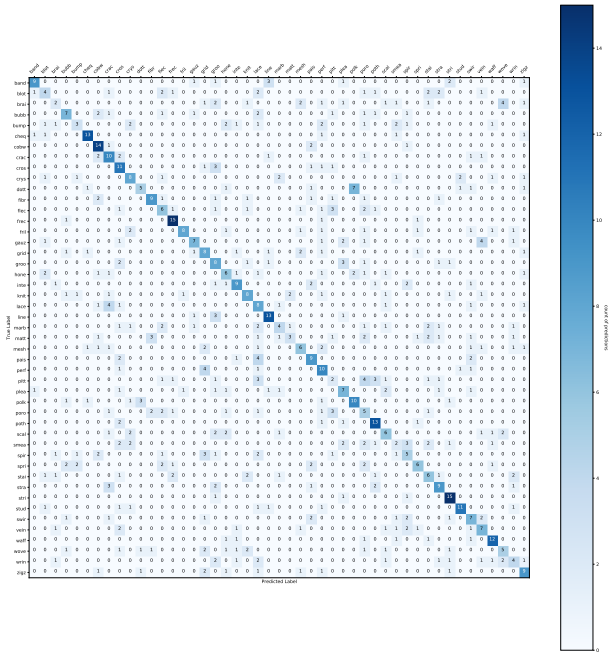


Figure A.4: Experiment 1 with SGD Optimizer—the confusion matrix correlates with the best model (DWTaP) obtained for the DTD dataset.



(a) MaxP



(b) DWTaP.

Figure A.5: Experiment 2 with Adam Optimizer—the confusion matrix correlates with the two best models (MaxP and DWTaP) for the DTD dataset.

A.2.3 FMD Dataset.

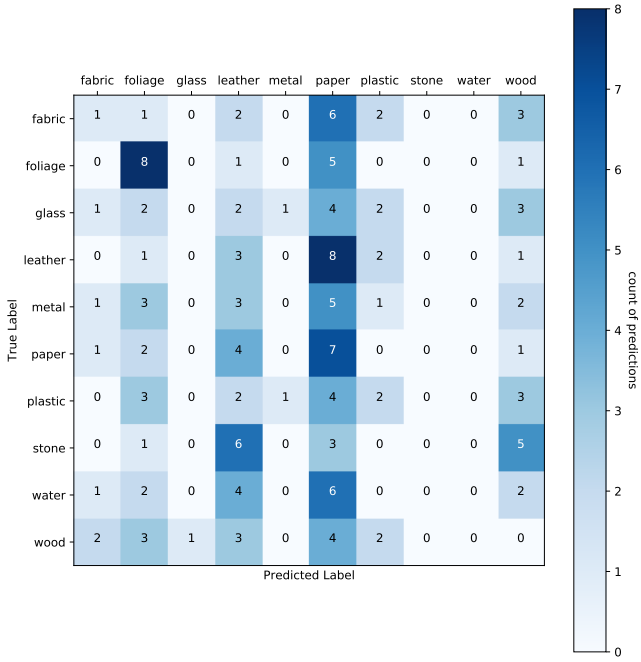
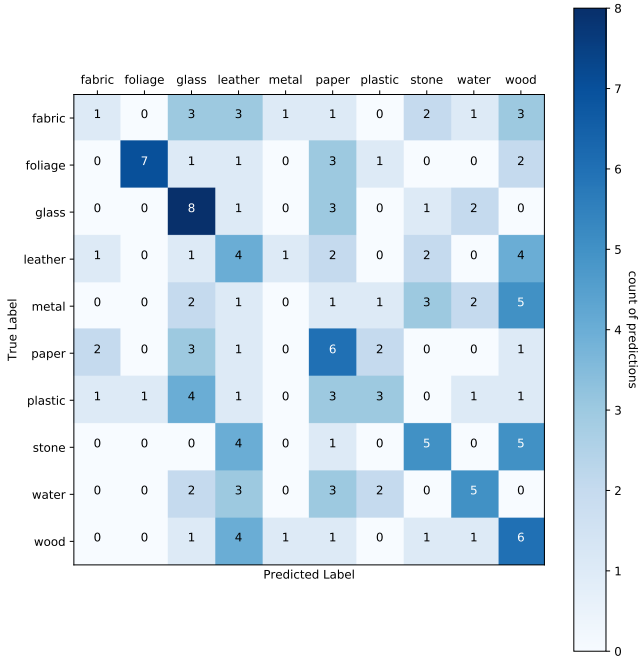
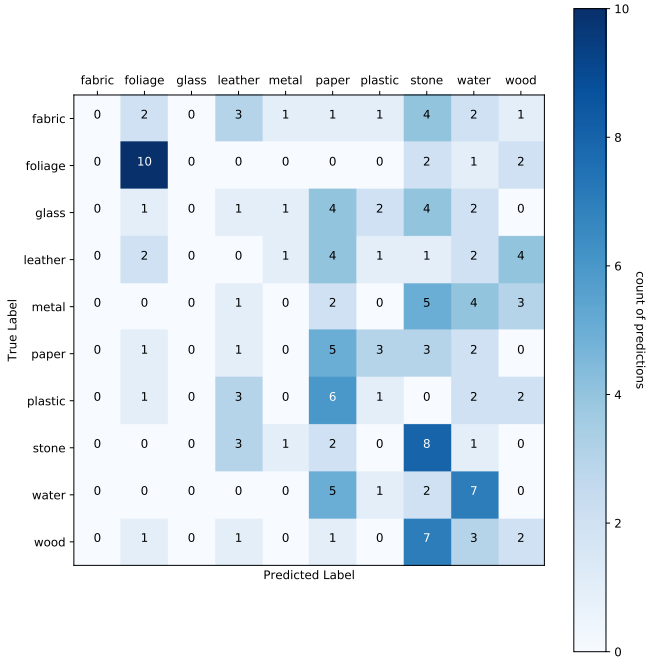


Figure A.6: Experiment 1 with SGD Optimizer—The confusion matrix correlates with the best model (DWTdP) obtained for the FMD dataset.



(a) AveP.



(b) DWTdP.

Figure A.7: Experiment 2 with Adam Optimizer—the confusion matrix correlates with the two best models (AveP and DWTdP) for the DTD dataset.

A.3 Classification Report with Evaluation Metrics.

A.3.1 CIFAR-10 Dataset.

Table A.1: Classification report for CIFAR-10 dataset. In this case, each pooling method is evaluated considering DropOut, Data Augmentation, and Batch Normalization.

Method	**MaxP**			**AveP**			**DWTP**			**DWTaP**			**DWTdP**			
Class	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**Test**
airplane	0.89	0.92	0.90	0.86	0.93	0.89	0.92	0.90	0.90	0.90	0.90	0.90	0.86	0.90	0.88	1000
automobile	0.92	0.97	0.95	0.93	0.96	0.94	0.94	0.97	0.95	0.93	0.97	0.95	0.92	0.96	0.94	1000
bird	0.87	0.84	0.86	0.87	0.83	0.85	0.85	0.85	0.85	0.88	0.83	0.86	0.83	0.80	0.81	1000
cat	0.81	0.75	0.78	0.78	0.77	0.77	0.81	0.77	0.78	0.80	0.79	0.79	0.75	0.72	0.73	1000
deer	0.88	0.90	0.89	0.89	0.86	0.87	0.89	0.88	0.89	0.86	0.92	0.89	0.84	0.89	0.86	1000
dog	0.85	0.82	0.83	0.88	0.78	0.83	0.84	0.84	0.84	0.86	0.83	0.84	0.82	0.75	0.78	1000
frog	0.87	0.95	0.91	0.88	0.78	0.83	0.89	0.95	0.92	0.92	0.94	0.93	0.88	0.91	0.89	1000
horse	0.92	0.92	0.92	0.88	0.94	0.91	0.94	0.93	0.93	0.93	0.94	0.93	0.90	0.91	0.90	1000
ship	0.96	0.91	0.94	0.95	0.92	0.93	0.94	0.94	0.94	0.95	0.93	0.94	0.94	0.92	0.93	1000
truck	0.94	0.93	0.94	0.91	0.93	0.92	0.94	0.94	0.94	0.93	0.94	0.93	0.93	0.92	0.93	1000
acc			**0.89**			**0.89**			**0.89**			**0.90**			**0.87**	10000

A.3.2 DTD Dataset.

Table A.2: Experiment 1 with SGD Optimizer—classification report for the DTD dataset.

Method	MaxP			AveP			DWTP			DWTaP			DWTdP			
Class	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	Test
band	0.67	0.44	0.53	0.75	0.50	0.60	0.71	0.56	0.63	0.83	0.56	0.67	0.12	0.06	0.08	18
blot	0.00	0.00	0.00	0.33	0.06	0.10	0.08	0.06	0.06	0.13	0.11	0.12	0.00	0.00	0.00	18
brai	0.17	0.06	0.08	0.13	0.11	0.12	0.29	0.11	0.16	0.10	0.06	0.07	0.11	0.06	0.07	18
bubb	0.17	0.06	0.08	0.64	0.39	0.48	0.30	0.33	0.32	0.20	0.17	0.18	0.10	0.11	0.11	18
bump	0.73	0.44	0.55	0.43	0.17	0.24	0.00	0.00	0.00	1.00	0.17	0.29	0.00	0.00	0.00	18
cheq	0.48	0.67	0.56	0.75	0.50	0.60	0.65	0.61	0.63	0.69	0.50	0.58	0.33	0.33	0.33	18
cobw	0.48	0.67	0.56	0.68	0.72	0.70	0.52	0.72	0.60	0.61	0.61	0.61	0.12	0.06	0.08	18
crac	0.45	0.28	0.34	0.33	0.44	0.38	0.33	0.44	0.38	0.29	0.33	0.31	0.11	0.11	0.11	18
cros	0.16	0.44	0.24	0.36	0.56	0.43	0.24	0.50	0.33	0.27	0.67	0.38	0.00	0.00	0.00	18
crys	0.35	0.33	0.34	0.60	0.50	0.55	0.53	0.44	0.48	0.29	0.33	0.31	0.11	0.11	0.11	18
dott	0.80	0.22	0.35	0.15	0.11	0.13	0.42	0.28	0.33	0.50	0.28	0.36	0.00	0.00	0.00	18
fibr	0.17	0.28	0.21	0.41	0.39	0.40	0.35	0.44	0.39	0.30	0.39	0.34	0.12	0.33	0.18	18
flec	0.15	0.44	0.22	0.13	0.39	0.19	0.18	0.17	0.17	0.17	0.56	0.26	0.10	0.17	0.13	18
frec	0.43	0.56	0.49	0.64	0.78	0.70	0.93	0.78	0.85	0.50	0.72	0.59	0.21	0.67	0.32	18
fril	0.21	0.22	0.22	0.53	0.44	0.48	0.60	0.50	0.55	0.36	0.28	0.31	0.00	0.00	0.00	18
gauz	0.38	0.17	0.23	0.32	0.33	0.32	0.39	0.39	0.39	0.45	0.28	0.34	0.10	0.22	0.14	18
grid	0.14	0.06	0.08	0.36	0.44	0.40	0.40	0.56	0.47	0.40	0.56	0.47	0.00	0.00	0.00	18
groo	0.18	0.33	0.23	0.17	0.39	0.24	0.32	0.61	0.42	0.22	0.44	0.29	0.00	0.00	0.00	18
hone	0.50	0.06	0.10	0.50	0.17	0.25	0.42	0.28	0.33	0.00	0.00	0.00	0.00	0.00	0.00	18
inte	0.36	0.28	0.31	0.40	0.44	0.42	0.38	0.44	0.41	0.83	0.28	0.42	0.17	0.06	0.08	18
knit	0.13	0.50	0.21	0.43	0.56	0.49	0.57	0.44	0.50	0.25	0.56	0.34	0.00	0.00	0.00	18
lace	0.14	0.28	0.18	0.29	0.33	0.31	0.22	0.44	0.30	0.24	0.72	0.36	0.07	0.17	0.10	18
line	0.41	0.72	0.52	0.52	0.67	0.59	0.53	0.56	0.54	0.71	0.56	0.63	0.11	0.06	0.07	18
marb	0.21	0.33	0.26	0.43	0.33	0.38	0.23	0.28	0.25	0.25	0.17	0.20	0.06	0.11	0.07	18
matt	0.29	0.44	0.35	0.35	0.39	0.37	0.40	0.33	0.36	0.36	0.28	0.31	0.07	0.22	0.11	18
mesh	0.50	0.06	0.10	0.50	0.17	0.25	0.50	0.44	0.47	0.20	0.06	0.09	0.17	0.06	0.08	18
pais	0.19	0.22	0.21	0.34	0.72	0.46	0.31	0.56	0.40	0.33	0.44	0.38	0.50	0.06	0.10	18
perf	0.40	0.22	0.29	0.46	0.33	0.39	0.35	0.44	0.39	0.46	0.33	0.39	0.50	0.06	0.10	18
pitt	0.24	0.33	0.28	0.00	0.00	0.00	0.25	0.22	0.24	0.24	0.22	0.23	0.00	0.00	0.00	18
plea	0.60	0.17	0.26	0.30	0.33	0.32	0.37	0.56	0.44	0.75	0.17	0.27	0.00	0.00	0.00	18
polk	0.50	0.17	0.25	0.42	0.56	0.48	0.62	0.56	0.59	0.56	0.28	0.37	0.00	0.00	0.00	18
poro	0.05	0.06	0.05	0.50	0.28	0.36	0.13	0.11	0.12	0.17	0.22	0.19	0.07	0.17	0.10	18
poth	0.26	0.44	0.33	0.38	0.67	0.48	0.45	0.72	0.55	0.31	0.89	0.46	0.07	0.39	0.12	18
scal	0.00	0.00	0.00	0.33	0.11	0.17	0.22	0.11	0.15	0.00	0.00	0.00	0.12	0.06	0.08	18
smea	0.00	0.00	0.00	0.22	0.11	0.15	0.14	0.11	0.12	0.20	0.06	0.09	0.00	0.00	0.00	18
spir	0.50	0.11	0.18	0.20	0.11	0.14	0.30	0.17	0.21	0.25	0.22	0.24	0.36	0.22	0.28	18
spri	0.75	0.17	0.27	0.50	0.33	0.40	0.50	0.33	0.40	0.33	0.11	0.17	1.00	0.06	0.11	18
stai	0.10	0.17	0.12	0.17	0.06	0.08	0.25	0.17	0.20	0.22	0.28	0.24	0.22	0.33	0.27	18
stra	0.31	0.50	0.38	0.38	0.67	0.48	0.40	0.56	0.47	0.24	0.28	0.26	0.00	0.00	0.00	18
stri	0.81	0.72	0.76	0.55	0.67	0.60	0.73	0.61	0.67	0.75	0.67	0.71	0.25	0.28	0.26	18
stud	0.64	0.50	0.56	0.60	0.67	0.63	0.55	0.61	0.58	0.64	0.39	0.48	0.20	0.50	0.28	18
swir	0.50	0.11	0.18	0.29	0.22	0.25	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	18
vein	0.25	0.28	0.26	0.43	0.33	0.38	0.46	0.33	0.39	0.75	0.17	0.27	0.14	0.28	0.19	18
waff	0.55	0.67	0.60	0.79	0.61	0.69	0.52	0.78	0.62	0.52	0.67	0.59	0.25	0.56	0.34	18
wove	0.25	0.22	0.24	0.40	0.56	0.47	0.45	0.50	0.47	0.39	0.50	0.44	0.14	0.22	0.17	18
wrin	0.00	0.00	0.00	0.30	0.17	0.21	0.20	0.06	0.09	0.29	0.11	0.16	0.00	0.00	0.00	18
zigz	0.18	0.11	0.14	0.30	0.39	0.34	0.27	0.22	0.24	0.42	0.44	0.43	0.00	0.00	0.00	18
acc			**0.27**			**0.39**			**0.39**			**0.34**			**0.13**	846

Table A.3: Experiment 2 with Adam Optimizer—classification report for the DTD dataset.

Method	MaxP			AveP			DWTP			DWTaP			DWTdP			
Class	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	Test
band	0.65	0.72	0.68	0.71	0.67	0.69	0.82	0.78	0.80	0.75	0.50	0.60	0.21	0.33	0.26	18
blot	0.50	0.06	0.10	0.10	0.06	0.07	0.22	0.11	0.15	0.33	0.22	0.27	0.00	0.00	0.00	18
brai	0.23	0.17	0.19	0.00	0.00	0.00	0.16	0.17	0.16	0.25	0.11	0.15	0.00	0.00	0.00	18
bubb	0.54	0.39	0.45	0.43	0.17	0.24	0.40	0.44	0.42	0.47	0.39	0.42	0.00	0.00	0.00	18
bump	0.18	0.17	0.17	0.18	0.11	0.14	0.25	0.06	0.09	0.38	0.17	0.23	0.00	0.00	0.00	18
cheq	0.65	0.72	0.68	0.60	0.67	0.63	0.93	0.72	0.81	0.76	0.72	0.74	0.42	0.28	0.33	18
cobw	0.65	0.83	0.73	0.67	0.89	0.76	0.61	0.94	0.74	0.52	0.78	0.62	0.29	0.11	0.16	18
crac	0.39	0.61	0.48	0.55	0.61	0.58	0.77	0.56	0.65	0.38	0.56	0.45	0.33	0.11	0.17	18
cros	0.39	0.61	0.48	0.23	0.50	0.31	0.40	0.44	0.42	0.34	0.61	0.44	0.00	0.00	0.00	18
crys	0.59	0.56	0.57	0.47	0.50	0.49	0.44	0.78	0.56	0.42	0.44	0.43	0.14	0.28	0.19	18
dott	0.62	0.28	0.38	0.45	0.28	0.34	0.36	0.22	0.28	0.50	0.28	0.36	0.00	0.00	0.00	18
fibr	1.00	0.33	0.50	0.53	0.50	0.51	0.65	0.61	0.63	0.60	0.50	0.55	0.09	0.17	0.12	18
flec	0.32	0.39	0.35	0.15	0.28	0.20	0.30	0.39	0.34	0.32	0.33	0.32	0.21	0.22	0.22	18
frec	0.88	0.83	0.86	0.74	0.78	0.76	0.88	0.83	0.86	0.68	0.83	0.75	0.17	0.44	0.25	18
fril	0.59	0.56	0.57	0.58	0.39	0.47	0.53	0.56	0.54	0.80	0.44	0.57	0.08	0.06	0.06	18
gauz	0.38	0.28	0.32	0.39	0.39	0.39	0.28	0.28	0.28	0.58	0.39	0.47	0.12	0.28	0.17	18
grid	0.56	0.56	0.56	0.56	0.56	0.56	0.41	0.67	0.51	0.33	0.44	0.38	0.00	0.00	0.00	18
groo	0.25	0.39	0.30	0.32	0.50	0.39	0.37	0.39	0.38	0.27	0.44	0.33	0.00	0.00	0.00	18
hone	0.29	0.39	0.33	0.47	0.44	0.46	0.24	0.28	0.26	0.33	0.33	0.33	0.00	0.00	0.00	18
inte	0.36	0.56	0.43	0.41	0.39	0.40	0.50	0.67	0.57	0.53	0.50	0.51	0.25	0.06	0.09	18
knit	0.45	0.56	0.50	0.36	0.67	0.47	0.50	0.50	0.50	0.50	0.44	0.47	0.17	0.28	0.21	18
lace	0.33	0.44	0.38	0.37	0.39	0.38	0.50	0.56	0.53	0.24	0.44	0.31	0.12	0.39	0.19	18
line	0.65	0.61	0.63	0.57	0.72	0.63	0.72	0.72	0.72	0.62	0.72	0.67	0.44	0.39	0.41	18
marb	0.43	0.17	0.24	0.26	0.28	0.27	0.43	0.33	0.38	0.36	0.22	0.28	0.10	0.17	0.12	18
matt	0.48	0.61	0.54	0.41	0.39	0.40	0.57	0.44	0.50	0.50	0.17	0.25	0.08	0.28	0.12	18
mesh	0.33	0.22	0.27	0.43	0.33	0.38	0.55	0.33	0.41	0.43	0.33	0.38	0.20	0.11	0.14	18
pais	0.45	0.50	0.47	0.41	0.72	0.52	0.48	0.67	0.56	0.43	0.50	0.46	0.29	0.22	0.25	18
perf	0.33	0.39	0.36	0.50	0.50	0.50	0.35	0.44	0.39	0.34	0.56	0.43	0.00	0.00	0.00	18
pitt	0.14	0.06	0.08	0.25	0.28	0.26	0.13	0.11	0.12	0.15	0.11	0.13	0.17	0.06	0.08	18
plea	0.53	0.50	0.51	0.44	0.39	0.41	0.32	0.44	0.37	0.37	0.39	0.38	0.25	0.11	0.15	18
polk	0.48	0.67	0.56	0.50	0.61	0.55	0.44	0.44	0.44	0.53	0.56	0.54	0.00	0.00	0.00	18
poro	0.36	0.28	0.31	0.17	0.06	0.08	0.29	0.28	0.29	0.17	0.28	0.21	0.06	0.11	0.08	18
poth	0.53	0.50	0.51	0.45	0.56	0.50	0.65	0.61	0.63	0.54	0.72	0.62	0.11	0.50	0.18	18
scal	0.29	0.33	0.31	0.33	0.22	0.27	0.46	0.67	0.55	0.35	0.33	0.34	0.00	0.00	0.00	18
smea	0.00	0.00	0.00	0.08	0.06	0.06	0.12	0.06	0.08	0.18	0.11	0.14	0.00	0.00	0.00	18
spir	0.19	0.17	0.18	0.23	0.17	0.19	0.50	0.28	0.36	0.28	0.28	0.28	0.33	0.11	0.17	18
spri	0.56	0.28	0.37	0.57	0.22	0.32	0.56	0.50	0.53	0.50	0.33	0.40	0.00	0.00	0.00	18
stai	0.56	0.28	0.37	0.32	0.33	0.32	0.57	0.44	0.50	0.33	0.33	0.33	0.17	0.22	0.19	18
stra	0.56	0.56	0.56	0.38	0.44	0.41	0.59	0.56	0.57	0.47	0.50	0.49	0.00	0.00	0.00	18
stri	0.44	0.61	0.51	0.60	0.67	0.63	1.00	0.67	0.80	0.60	0.83	0.70	0.26	0.44	0.33	18
stud	0.44	0.61	0.51	0.60	0.67	0.63	0.61	0.61	0.61	0.73	0.61	0.67	0.38	0.33	0.35	18
swir	0.33	0.44	0.38	0.43	0.33	0.38	0.41	0.39	0.40	0.35	0.39	0.37	0.00	0.00	0.00	18
vein	0.22	0.28	0.24	0.30	0.33	0.32	0.45	0.28	0.34	0.37	0.39	0.38	0.12	0.28	0.17	18
waff	0.67	0.67	0.67	0.71	0.56	0.63	0.60	0.67	0.63	0.63	0.67	0.65	0.19	0.56	0.29	18
wove	0.43	0.67	0.52	0.35	0.44	0.39	0.45	0.72	0.55	0.38	0.28	0.32	0.24	0.28	0.26	18
wrin	0.50	0.11	0.18	0.40	0.11	0.17	0.33	0.44	0.38	0.31	0.22	0.26	0.00	0.00	0.00	18
zigz	0.36	0.56	0.43	0.45	0.56	0.50	0.60	0.50	0.55	0.47	0.50	0.49	0.04	0.06	0.05	18
acc			**0.43**			**0.42**			**0.48**			**0.43**			**0.15**	846

A.3.3 FMD Dataset.

Table A.4: Experiment 1 with SGD Optimizer—classification report for the FMD dataset.

Method	MaxP			AveP			DWTP			DWTaP			DWTdP			
Class	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**Test**
fabric	0.00	0.00	0.00	0.07	0.07	0.07	0.00	0.00	0.00	0.29	0.13	0.18	0.14	0.07	0.09	15
foliage	0.00	0.00	0.00	1.00	0.33	0.50	1.00	0.73	0.85	0.73	0.73	0.73	0.31	0.53	0.39	15
glass	0.00	0.00	0.00	0.33	0.07	0.11	0.12	0.07	0.09	0.38	0.20	0.26	0.00	0.00	0.00	15
leather	0.12	0.80	0.21	0.17	0.33	0.23	0.17	0.20	0.18	0.17	0.47	0.25	0.10	0.20	0.13	15
metal	0.00	0.00	0.00	0.11	0.20	0.14	0.19	0.20	0.19	0.17	0.13	0.15	0.00	0.00	0.00	15
paper	0.43	0.40	0.41	0.15	0.13	0.14	0.30	0.20	0.24	0.00	0.00	0.00	0.13	0.47	0.21	15
plastic	0.00	0.00	0.00	1.00	0.13	0.17	0.43	0.20	0.27	0.33	0.13	0.19	0.18	0.13	0.15	15
stone	0.12	0.07	0.09	0.22	0.13	0.17	0.22	0.40	0.29	0.30	0.20	0.24	0.00	0.00	0.00	15
water	0.04	0.07	0.05	0.60	0.60	0.60	0.33	0.47	0.39	0.47	0.53	0.50	0.00	0.00	0.00	15
wood	0.00	0.00	0.00	0.21	0.47	0.29	0.25	0.40	0.31	0.25	0.53	0.34	0.00	0.00	0.00	15
acc			**0.13**			**0.25**			**0.29**			**0.31**			**0.14**	150

Table A.5: Experiment 2 with Adam Optimizer—classification report for the FMD dataset.

Method	MaxP			AveP			DWTP			DWTaP			DWTdP			
Class	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**	**Test**
fabric	0.06	0.07	0.06	0.20	0.07	0.10	0.06	0.07	0.06	0.11	0.13	0.12	0.00	0.00	0.00	15
foliage	0.82	0.60	0.69	0.88	0.47	0.61	0.92	0.73	0.81	1.00	0.80	0.89	0.56	0.67	0.61	15
glass	0.50	0.07	0.12	0.32	0.53	0.40	0.29	0.13	0.18	0.20	0.20	0.20	0.00	0.00	0.00	15
leather	0.12	0.13	0.12	0.17	0.27	0.21	0.13	0.13	0.13	0.21	0.33	0.26	0.00	0.00	0.00	15
metal	0.14	0.13	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.25	0.07	0.11	0.00	0.00	0.00	15
paper	0.25	0.13	0.17	0.25	0.40	0.31	0.31	0.33	0.32	0.17	0.07	0.10	0.17	0.33	0.22	15
plastic	0.25	0.07	0.11	0.33	0.20	0.25	0.33	0.20	0.25	0.36	0.27	0.31	0.11	0.07	0.08	15
stone	0.33	0.60	0.43	0.36	0.33	0.34	0.30	0.60	0.40	0.17	0.27	0.21	0.22	0.53	0.31	15
water	0.35	0.80	0.49	0.42	0.33	0.37	0.47	0.53	0.50	0.50	0.40	0.44	0.27	0.47	0.34	15
wood	0.25	0.27	0.26	0.22	0.40	0.29	0.35	0.40	0.38	0.22	0.33	0.26	0.14	0.13	0.14	15
acc			**0.29**			**0.30**			**0.31**			**0.29**			**0.22**	150

B

Appendix.

In this appendix, some codes have been used in the proposed approaches. In addition, the link to Github, where we can find the complete experiments and codes.

Contents

B.1 Algorithm 1: Binary Classification.

Algorithm design for CNN training. The first training code developed with Keras and TensorFlow is shown below. It is a binary classifier that allows classifying into two unique classes. See at `https://github.com/JanManuell/ThesisPhD-1ProposedApproach.git`.

Listing B.1: CNN - Train code.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Aug  6 11:39:17 2019
Proceso completo para clasificaci\'on y detecci\'on de texturas en un ambiente simulado.
Entrenamiento de una RedNeuronal (Codigo-copia)
Al final se gr\'afica el proceso de entrenamiento y validaci\'on.
@author: jm
"""
import os, shutil
from keras import layers
from keras import models
from keras import optimizers
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
from keras import backend as K

img_width, img_height = 64, 64

#DIRECTORIO DE IMAGENES DE TEXTURAS y NO TEXTURAS
original_dataset_dir = '/home/jm/Escritorio/CNNWavelet_Texture/Datasets/Data5/tex_a'
original_dataset_dir1 = '/home/jm/Escritorio/CNNWavelet_Texture/Datasets/Data5/scene_a'
#DIRECTORIO PARA DATASET CON IMAGENES DE ENTRENAMIENTO,VALIDACIoN Y TEST
base_dir = '/home/jm/Escritorio/CNNWavelet_Texture/Datasets/Testing5_a'
os.mkdir(base_dir)

train_dir = os.path.join(base_dir, 'train')
os.mkdir(train_dir)
validation_dir = os.path.join(base_dir, 'validation')
os.mkdir(validation_dir)
test_dir = os.path.join(base_dir, 'test')
os.mkdir(test_dir)
#DIRECTORIO CON IMG ENTRENAMIENTO TEXTURAS
train_texture_dir = os.path.join(train_dir, 'textura')
os.mkdir(train_texture_dir)
#DIRECTORIO CON IMG ENTRENAMIENTO NO TEXTURAS
train_nottex_dir = os.path.join(train_dir, 'nottext')
os.mkdir(train_nottex_dir)
#DIRECTORIO CON IMG VALIDACION TEXTURAS
validation_texture_dir = os.path.join(validation_dir, 'textura')
os.mkdir(validation_texture_dir)
#DIRECTORIO CON IMG VALIDACION NO TEXTURAS
validation_nottex_dir = os.path.join(validation_dir, 'nottext')
os.mkdir(validation_nottex_dir)
#DIRECTORIO CON IMG TEST TEXTURAS
test_texture_dir = os.path.join(test_dir, 'textura')
```

```
os.mkdir(test_texture_dir)
#DIRECTORIO CON IMG TEST TEXTURAS
test_nottex_dir = os.path.join(test_dir, 'nottext')
os.mkdir(test_nottex_dir)
#COPIA LAS PRIMERAS 700 IMG TEXTURAS A TRAIN_TEXTURE_DIR
#fnames = ['ImgAtest{}.jpg'.format(i) for i in range(700)]
fnames = ['test-{}.jpg'.format(i) for i in range(700)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(train_texture_dir, fname)
    shutil.copyfile(src, dst)
#COPIA LAS SIGUIENTES 150 IMG TEXTURAS A VALIDATION_TEXTURE_DIR
fnames = ['test-{}.jpg'.format(i) for i in range(700, 850)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(validation_texture_dir, fname)
    shutil.copyfile(src, dst)
#COPIA LAS SIGUIENTES 150 IMG TEXTURAS A TEST_TEXTURE_DIR
fnames = ['test-{}.jpg'.format(i) for i in range(850, 954)]
for fname in fnames:
    src = os.path.join(original_dataset_dir, fname)
    dst = os.path.join(test_texture_dir, fname)
    shutil.copyfile(src, dst)
#COPIA LAS PRIMERAS 700 IMG NO TEXTURAS A TRAIN_NOTTEXTURE_DIR
fnames = ['test-{}.jpg'.format(i) for i in range(700)]
for fname in fnames:
    src = os.path.join(original_dataset_dir1, fname)
    dst = os.path.join(train_nottex_dir, fname)
    shutil.copyfile(src, dst)
#COPIA LAS SIGUIENTES 150 IMG NO TEXTURAS A VALIDATION_NOTTEXTURE_DIR
fnames = ['test-{}.jpg'.format(i) for i in range(700, 850)]
for fname in fnames:
    src = os.path.join(original_dataset_dir1, fname)
    dst = os.path.join(validation_nottex_dir, fname)
    shutil.copyfile(src, dst)
#COPIA LAS SIGUIENTES 150 IMG NO TEXTURAS A TEST_NOTTEXTURE_DIR
fnames = ['test-{}.jpg'.format(i) for i in range(850, 954)]
for fname in fnames:
    src = os.path.join(original_dataset_dir1, fname)
    dst = os.path.join(test_nottex_dir, fname)
    shutil.copyfile(src, dst)

#Checking format of Image:
if K.image_data_format() == 'channels_first':
  input_shape = (3, img_width, img_height)
else:
  input_shape = (img_width, img_height, 3)

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',input_shape=input_shape))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
#model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
#Configuring the model for training
model.compile(loss='binary_crossentropy',
```

```
109                 optimizer=optimizers.RMSprop(lr=1e-4),
110                 metrics=['acc'])
111 #Using ImageDataGenerator to read images from directories
112 #Rescales all images by 1/255
113 train_datagen = ImageDataGenerator(rescale=1./255)
114 validation_datagen = ImageDataGenerator(rescale=1./255)
115 test_datagen = ImageDataGenerator(rescale=1./255)
116
117 train_generator = train_datagen.flow_from_directory(
118         train_dir,
119         target_size=(64, 64),
120         batch_size=20,
121         class_mode='binary')#Because you use binary_crossentropy loss, you need
      binary labels.
122 validation_generator = validation_datagen.flow_from_directory(
123         validation_dir,
124         target_size=(64, 64),
125         batch_size=20,
126         class_mode='binary')
127 test_generator = test_datagen.flow_from_directory(
128         test_dir,
129         target_size=(64, 64),
130         batch_size=20,
131         class_mode='binary')
132
133 #Fitting the model using a batch generator
134 history = model.fit_generator(
135       train_generator,
136       steps_per_epoch=100,
137       epochs=10,
138       validation_data=validation_generator,
139       validation_steps=50)
140
141 #Saving the model
142 model.save('model5_a.h5')
143
144 test_loss, test_acc = model.evaluate_generator(test_generator, steps=50)
145 print('test acc:', test_acc)
146 print('test loss:', test_loss)
147
148 #Displaying curves of loss and accuracy during training
149 acc = history.history['acc']
150 val_acc = history.history['val_acc']
151 loss = history.history['loss']
152 val_loss = history.history['val_loss']
153
154 epochs = range(1, len(acc) + 1)
155 plt.plot(epochs, acc, '-g', label='Training acc')
156 plt.plot(epochs, val_acc, 'b', label='Validation acc')
157 plt.title('Training and validation accuracy')
158 plt.xlabel('Epochs')
159 plt.ylabel('Accuracy')
160 plt.legend()
161
162 plt.figure()
163
164 plt.plot(epochs, loss, '-g', label='Training loss')
165 plt.plot(epochs, val_loss, 'b', label='Validation loss')
166 plt.title('Training and validation loss')
167 plt.xlabel('Epochs')
168 plt.ylabel('Accuracy')
169 plt.legend()
170 plt.show()
```

B.2 Algorithm 2: Modeltlvgg16_3Input1Model.

Architecture design with three inputs for multiclass classification. It is based on the fusion of information in the spatial and wavelet domain. The architecture has a functional basis with multiple inputs with random connections between layers. See at `https://github.com/JanManuell/ThesisPhD-2ProposedApproach.git`.

Listing B.2: Architecture design - code.

```
"""### **Arquitectura Multiple Input Model**: Wavelet CNN"""
print("[INFO] Dise-o de la arquitectura...")
# define two sets of inputs
inputA = Input(shape=(300,300,3))
inputB = Input(shape=(150,150,3))
inputC = Input(shape=(75,75,3))

l1 = MaxPooling2D(pool_size=(2, 2), strides=(2,2), name='block1_Pool')(inputA)
#l3 = MaxPooling2D(pool_size=(2, 2), strides=(2,2), name='block3_Pool')(inputA)

# Input original image starts
conv1 = Conv2D(64, (3, 3), strides=1, padding='same', activation='relu', name='
    block1_conv1')(inputA)
conv2 = Conv2D(64, (3, 3), strides=2, padding='same', activation='relu', name='
    block1_conv2')(conv1)
# level one decomposition starts
conv1_1 = Conv2D(64, (3, 3), strides=1, padding='same', activation='relu', name
    ='block1_conv3')(inputB)
# concate level one and level two decomposition
concat1 = concatenate([conv2, conv1_1, l1, inputB])
conv3 = Conv2D(128, (3, 3), strides=1, padding='same', activation='relu', name=
    'block2_conv1')(concat1)
conv4 = Conv2D(128, (3, 3), strides=2, padding='same', activation='relu', name=
    'block2_conv2')(conv3)
# level two decomposition starts
conv2_1 = Conv2D(64, (3, 3), strides=1, padding='same', activation='relu', name
    ='block2_conv3')(inputC)
conv3_1 = Conv2D(128, (3, 3), strides=1, padding='same', activation='relu',
    name='block2_conv4')(conv2_1)

l2 = MaxPooling2D(pool_size=(2, 2), strides=(2,2), name='block2_Pool')(concat1)

# concate level two and level three decomposition
concat2 = concatenate([conv4, conv3_1, l2, inputC])
conv6 = Conv2D(256, (3, 3), strides=1, padding='same', activation='relu', name=
    'block3_conv1')(concat2)
conv7 = Conv2D(256, (3, 3), strides=2, padding='same', activation='relu', name=
    'block3_conv2')(conv6)
conv8 = Conv2D(512, (3, 3), strides=1, padding='same', activation='relu', name=
    'block4_conv1')(conv7)
conv9 = Conv2D(512, (3, 3), strides=2, padding='same', activation='relu', name=
    'block4_conv2')(conv8)

avep1 = GlobalAveragePooling2D(name='block4_Pool')(conv9)
output = Dense(nClasses, activation='softmax', name='predictions')(avep1)

# create model with two inputs
model = Model([inputA,inputB,inputC], output)
```

```
model.summary()

[INFO] Dise-o de la arquitectura...
Model: "model_1"
__________________________________________________________________________________
Layer (type)                    Output Shape         Param #  Connected to
==================================================================================
input_1 (InputLayer)            (None, 300, 300, 3)  0
__________________________________________________________________________________
block1_conv1 (Conv2D)           (None, 300, 300, 64) 1792     input_1[0][0]
__________________________________________________________________________________
input_2 (InputLayer)            (None, 150, 150, 3)  0
__________________________________________________________________________________
block1_conv2 (Conv2D)           (None, 150, 150, 64) 36928    block1_conv1[0][0]
__________________________________________________________________________________
block1_conv3 (Conv2D)           (None, 150, 150, 64) 1792     input_2[0][0]
__________________________________________________________________________________
block1_Pool (MaxPooling2D)      (None, 150, 150, 3)  0        input_1[0][0]
__________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 150, 150, 134 0        block1_conv2[0][0]
                                                              block1_conv3[0][0]
                                                              block1_Pool[0][0]
                                                              input_2[0][0]
__________________________________________________________________________________

input_3 (InputLayer)            (None, 75, 75, 3)    0
__________________________________________________________________________________

block2_conv1 (Conv2D)           (None, 150, 150, 128 154496  concatenate_1[0][0]
__________________________________________________________________________________

block2_conv3 (Conv2D)           (None, 75, 75, 64)   1792     input_3[0][0]
__________________________________________________________________________________
block2_conv2 (Conv2D)           (None, 75, 75, 128)  147584   block2_conv1[0][0]
__________________________________________________________________________________
block2_conv4 (Conv2D)           (None, 75, 75, 128)  73856    block2_conv3[0][0]
__________________________________________________________________________________

block2_Pool (MaxPooling2D)      (None, 75, 75, 134)  0       concatenate_1[0][0]
__________________________________________________________________________________

concatenate_2 (Concatenate)     (None, 75, 75, 393)  0        block2_conv2[0][0]
                                                              block2_conv4[0][0]
                                                              block2_Pool[0][0]
                                                              input_3[0][0]
__________________________________________________________________________________

block3_conv1 (Conv2D)           (None, 75, 75, 256)  905728  concatenate_2[0][0]
__________________________________________________________________________________
block3_conv2 (Conv2D)           (None, 38, 38, 256)  590080   block3_conv1[0][0]
__________________________________________________________________________________
block4_conv1 (Conv2D)           (None, 38, 38, 512)  1180160  block3_conv2[0][0]
__________________________________________________________________________________
block4_conv2 (Conv2D)           (None, 19, 19, 512)  2359808  block4_conv1[0][0]
__________________________________________________________________________________
block4_Pool (GlobalAveragePooli(None, 512)           0        block4_conv2[0][0]
__________________________________________________________________________________
predictions (Dense)             (None, 47)           24111    block4_Pool[0][0]
==================================================================================
Total params: 5,478,127
Trainable params: 5,478,127
Non-trainable params: 0
__________________________________________________________________________________
```

B.3 Algorithm 3: PyWavelets.

Creation of datasets (test, train, and valid) and application of the Discrete Wavelet Transform in grayscale. The full version can be found at the link `https://github.com/JanManuell/ThesisPhD-3ProposedApproach.git`.

Listing B.3: PYWT - Wavelet Transforms in Python

```
#Metodologia 2DWT
"""
G. R. Lee, R.Gommers, F. Wasilewski, K. Wohlfahrt, A. OLeary (2019).
PyWavelets: A Python package for wavelet analysis.
Journal of Open Source Software, 4(36), 1237,
https://doi.org/10.21105/joss.01237
"""
import pywt # Wavelet transform of image
import pywt.data
import cv2
from sklearn import preprocessing
min_max_scaler = preprocessing.MinMaxScaler()

n_imagenes = 846
n_imagenes1 = 3931
n_imagenes2 = 863
alto = 150
ancho = 150
#alto1 = 75
#ancho1 = 75
canales = 1
#canalesx = 1
testA_x = np.zeros((n_imagenes, ancho, alto, canales), dtype = np.float32)
trainA_x = np.zeros((n_imagenes1, ancho, alto, canales), dtype = np.float32)
validA_x = np.zeros((n_imagenes2, ancho, alto, canales), dtype = np.float32)

testH_x = np.zeros((n_imagenes, ancho, alto, canales), dtype = np.float32)
trainH_x = np.zeros((n_imagenes1, ancho, alto, canales), dtype = np.float32)
validH_x = np.zeros((n_imagenes2, ancho, alto, canales), dtype = np.float32)

testV_x = np.zeros((n_imagenes, ancho, alto, canales), dtype = np.float32)
trainV_x = np.zeros((n_imagenes1, ancho, alto, canales), dtype = np.float32)
validV_x = np.zeros((n_imagenes2, ancho, alto, canales), dtype = np.float32)

testD_x = np.zeros((n_imagenes, ancho, alto, canales), dtype = np.float32)
trainD_x = np.zeros((n_imagenes1, ancho, alto, canales), dtype = np.float32)
validD_x = np.zeros((n_imagenes2, ancho, alto, canales), dtype = np.float32)

size=len(test_x)
print("Cantidad de imagenes:",size)

#Se descomprime cada imagen de la lista creada.
for i in range(0, size):
    img1 = test_x[i] #Imagenes
    x = cv2.cvtColor(img1,cv2.COLOR_RGB2GRAY)

    max_lev = 1       # how many levels of decomposition to draw

    for level in range(0, max_lev + 1):
```

```
        # compute the 2D DWT
        c = pywt.wavedec2(x, 'haar', mode='periodization', level=level)
    # Reconstruction
    #N=1, Img aproximacipn canales BGR.
    IMGba=((c[0]))
    imga = np.expand_dims(IMGba, axis = 2)
    #Save Data
    testA_x[i] = imga
```

B.4 Algorithm 4: Wavelet Pooling Layer.

Creation of the wavelet layer using Keras.Layers methods. The script shows the full DWTP version. The approximation and detail versions can be found inside the link, respectively. See at `https://github.com/JanManuell/ThesisPhD-4ProposedApproach.git`.

Listing B.4: DWTP - Wavelet Pooling in Keras Layers

```
import keras.backend as K
from keras.layers import Layer

def dwt(x, data_format='channels_last'):

    """
    DWT (Discrete Wavelet Transform) function implementation according to
    "Multi-level Wavelet Convolutional Neural Networks"
    by Pengju Liu, Hongzhi Zhang, Wei Lian, Wangmeng Zuo
    https://arxiv.org/abs/1907.03128
    """

    if data_format == 'channels_last':
        x1 = x[:, 0::2, 0::2, :] #x(2i-1, 2j-1)
        x2 = x[:, 1::2, 0::2, :] #x(2i, 2j-1)
        x3 = x[:, 0::2, 1::2, :] #x(2i-1, 2j)
        x4 = x[:, 1::2, 1::2, :] #x(2i, 2j)

    elif data_format == 'channels_first':
        x1 = x[:, :, 0::2, 0::2] #x(2i-1, 2j-1)
        x2 = x[:, :, 1::2, 0::2] #x(2i, 2j-1)
        x3 = x[:, :, 0::2, 1::2] #x(2i-1, 2j)
        x4 = x[:, :, 1::2, 1::2] #x(2i, 2j)

    x_LL = x1 + x2 + x3 + x4
    x_LH = -x1 - x3 + x2 + x4
    x_HL = -x1 + x3 - x2 + x4
    x_HH = x1 - x3 - x2 + x4

    if data_format == 'channels_last':
        return K.concatenate([x_LL,x_LH,x_HL,x_HH],axis=-1)
    elif data_format == 'channels_first':
        return K.concatenate([x_LL,x_LH,x_HL,x_HH],axis=1)
class DWT_Pooling(Layer):
    """
    Custom Layer performing DWT pooling operation described in :
```

```
37
38      "Multi-level Wavelet Convolutional Neural Networks"
39      by Pengju Liu, Hongzhi Zhang, Wei Lian, Wangmeng Zuo
40      https://arxiv.org/abs/1907.03128
41
42      # Arguments :
43          data_format (String): 'channels_first' or 'channels_last'
44
45      # Input shape :
46          If data_format='channels_last':
47              4D tensor of shape: (batch_size, rows, cols, channels)
48          If data_format='channels_first':
49              4D tensor of shape: (batch_size, channels, rows, cols)
50
51      # Output shape
52          If data_format='channels_last':
53              4D tensor of shape: (batch_size, rows/2, cols/2, channels*4)
54          If data_format='channels_first':
55              4D tensor of shape: (batch_size, channels*4, rows/2, cols/2)
56      """
57
58      def __init__(self, data_format=None,**kwargs):
59          super(DWT_Pooling, self).__init__(**kwargs)
60          self.data_format = K.normalize_data_format(data_format)
61
62      def build(self, input_shape):
63          super(DWT_Pooling, self).build(input_shape)
64
65      def call(self, x):
66          return dwt(x, self.data_format)
67
68      def compute_output_shape(self, input_shape):
69          if self.data_format == 'channels_first':
70              return (input_shape[0], input_shape[1]*4, input_shape[2]//2,
        input_shape[3]//2)
71          elif self.data_format == 'channels_last':
72              return (input_shape[0], input_shape[1]//2, input_shape[2]//2,
        input_shape[3]*4)
```

C

Appendix

C.1 List of terms and abbreviations

CNN/ConvNet Convolutional Neural Network.
IWT Inverse Wavelet Transform.
MAV Micro Aerial Vehicle.
UAV Unmanned Aerial Vehicle.
SSD Single Shot MultiBox Detector.
GPU Graphics Processing Unit.
YOLO You Only Look Once.
ROS The Robot Operating System.
OpenCV Open Source Computer Vision Library.
VGG Very Deep Convolutional Network.
DTD Describable Texture Database.
FMD Flickr Material Database.
KT2B KTH-TIPS-2B.
DWTP Discrete Wavelet Transform Pooling.
DWTaP Discrete Wavelet Transform Approximation Pooling.
DWTdP Discrete Wavelet Transform Detail Pooling.
AveP Ave-Pooling.
MaxP Max-Pooling.
2D-WT Two-Dimensional Wavelet Transform.
2D-DWT Two-Dimensional Discrete Wavelet Transform.
DWT Discrete Wavelet Transform.
M-MA Mallat-Multiresolution Analysis.
2D-FWT Fast Two-Dimensional Wavelet Transform.
ANNs Artificial Neural Networks.
ReLU Rectified Linear Unit.
SGD Stochastic Gradient Descent optimizer.
FCN Fully Connected Layers.
RGB Red Green Blue.
CPU Central Processing Unit.
BSD Berkeley Software Distribution.
API Application Programming Interface.
ROI Region of Interest.

Bibliography

[1] C. Vargas-Olmos, *Procesamiento de señales y solución de problemas con la transformada wavelet*. Ph.d. dissertation, Instituto de Investigación en Comunicación Óptica, UASLP, San Luis Potosí, México, 2017.

[2] C. C. Aggarwal *et al.*, "Neural networks and deep learning," *Springer*, vol. 10, pp. 978–3, 2018.

[3] M. Rivera, "Reeuso de redes preentrenadas (cimat)." [Online]. Available: http://mrivera/cursos/aprendizaje_profundo/preentrenadas/preentrenadas.html#transferencia-de-conocimiento-de-redes-preentrenadas-a-nuevos-problemas.

[4] F. Chollet, *Deep learning with Python*. Shelter Island, NY: Manning Publications Co, 2018.

[5] R. Group, "What is gradient descent?." [Online], 2021. Available: https://www.robofied.com/.

[6] A. A. Cabrera-Ponce and J. Martínez-Carranza, "Onboard cnn-based processing for target detection and autonomous landing for mavs," in *Mexican Conference on Pattern Recognition*, pp. 195–208, Springer, 2020.

[7] C. Vargas-Olmos, "Procesamiento de imágenes con métodos de ondeleta," m.s. thesis, Instituto de Investigación en Comunicación Óptica, UASLP, San Luis Potosí, México, jun 2010.

[8] Y. Bazi and F. Melgani, "Convolutional svm networks for object detection in uav imagery," *Ieee transactions on geoscience and remote sensing*, vol. 56, no. 6, pp. 3107–3118, 2018.

[9] N. S. Vassilieva, "Content-based image retrieval methods," *Programming and Computer Software*, vol. 35, no. 3, pp. 158–180, 2009.

[10] L. O. Rojas Pérez and J. Martínez Carranza, "Autonomous drone racing with an opponent: a first approach," *Computación y Sistemas*, vol. 24, no. 3, pp. 1271–1279, 2020.

[11] V. Alcalá-Rmz, V. Maeda-Gutiérrez, L. A. Zanella-Calzada, A. Valladares-Salgado, J. M. Celaya-Padilla, and C. E. Galván-Tejada, "Convolutional neural network for classification of diabetic retinopathy grade," in *Mexican International Conference on Artificial Intelligence*, pp. 104–118, Springer, 2020.

[12] J. M. Tapia-Téllez and H. J. Escalante, "Data augmentation with transformers for text classification," in *Mexican International Conference on Artificial Intelligence*, pp. 247–259, Springer, 2020.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, vol. 1. MIT Press: Cambridge, MA, USA, 2017.

[15] T. Williams, R. Li, *et al.*, "An ensemble of convolutional neural networks using wavelets for image classification," *Journal of Software Engineering and Applications*, vol. 11, no. 02, p. 69, 2018.

[16] P. Liu, H. Zhang, W. Lian, and W. Zuo, "Multi-level wavelet convolutional neural networks," *IEEE Access*, vol. 7, pp. 74973–74985, 2019.

[17] J. Piao, Y. Chen, and H. Shin, "A new deep learning based multi-spectral image fusion method," *Entropy*, vol. 21, no. 6, p. 570, 2019.

[18] D. De Silva, S. Fernando, I. T. S. Piyatilake, and A. Karunarathne, "Wavelet based edge feature enhancement for convolutional neural networks," in *Eleventh International Conference on Machine Vision (ICMV 2018)*, vol. 11041, pp. 751–760, SPIE, 2019.

[19] V. Andrearczyk and P. F. Whelan, "Using filter banks in convolutional neural networks for texture classification," *Pattern Recognition Letters*, vol. 84, pp. 63–69, 2016.

[20] S. Fujieda, K. Takayama, and T. Hachisuka, "Wavelet convolutional neural networks for texture classification," *arXiv preprint arXiv:1707.07394*, 2017.

[21] S. Fujieda, K. Takayama, and T. Hachisuka, "Wavelet convolutional neural networks," *arXiv preprint arXiv:1805.08620*, 2018.

[22] T. Williams and R. Li, "Wavelet pooling for convolutional neural networks," in *International Conference on Learning Representations*, 2018.

[23] C. Ben Chaabane, D. Mellouli, T. M. Hamdani, A. M. Alimi, and A. Abraham, "Wavelet convolutional neural networks for handwritten digits recognition," in *International Conference on Hybrid Intelligent Systems*, pp. 305–310, Springer, 2017.

[24] H. Gholamalinejad and H. Khosravi, "Vehicle classification using a real-time convolutional structure based on dwt pooling layer and se blocks," *Expert Systems with Applications*, vol. 183, p. 115420, 2021.

[25] A. Ferrà, E. Aguilar, and P. Radeva, "Multiple wavelet pooling for cnns," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pp. 0–0, 2018.

[26] A. de Souza Brito, M. B. Vieira, M. L. S. C. de Andrade, R. Q. Feitosa, and G. A. Giraldi, "Combining max-pooling and wavelet pooling strategies for semantic image segmentation," *Expert Systems with Applications*, vol. 183, p. 115403, 2021.

[27] Q. Li and L. Shen, "3d waveunet: 3d wavelet integrated encoder-decoder network for neuron segmentation," *arXiv preprint arXiv:2106.00259*, 2021.

[28] A. Alijamaat, A. NikravanShalmani, and P. Bayat, "Multiple sclerosis lesion segmentation from brain mri using u-net based on wavelet pooling," *International journal of computer assisted radiology and surgery*, vol. 16, no. 9, pp. 1459–1467, 2021.

[29] Y. Pi, N. D. Nath, and A. H. Behzadan, "Convolutional neural networks for object detection in aerial imagery for disaster response and recovery," *Advanced Engineering Informatics*, vol. 43, p. 101009, 2020.

[30] S. Dionisio-Ortega, L. O. Rojas-Perez, J. Martinez-Carranza, and I. Cruz-Vega, "A deep learning approach towards autonomous flight in forest environments," in *2018 International Conference on Electronics, Communications and Computers (CONIELECOMP)*, pp. 139–144, IEEE, 2018.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[32] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[33] S. Jung, S. Hwang, H. Shin, and D. H. Shim, "Perception, guidance, and navigation for indoor autonomous drone racing using deep learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2539–2544, 2018.

[34] S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 11, no. 7, pp. 674–693, 1989.

[35] P. Mallikarjuna, A. T. Targhi, M. Fritz, E. Hayman, B. Caputo, and J.-O. Eklundh, "The kth-tips2 database," *Computational Vision and Active Perception Laboratory, Stockholm, Sweden*, vol. 11, 2006.

[36] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing textures in the wild," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3606–3613, 2014.

[37] L. Sharan, R. Rosenholtz, and E. H. Adelson, "Accuracy and speed of material categorization in real-world images," *Journal of Vision*, vol. 14, no. 10, 2014.

[38] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[39] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[40] S. G. Mallat, "Multifrequency channel decompositions of images and wavelet models," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, no. 12, pp. 2091–2110, 1989.

[41] R. A. Gopinath, H. Guo, C. S. Burrus, and L. S. Burrus, "Introduction to wavelets and wavelet transforms: a primer," 1997.

[42] J. S. Walker, *A primer on wavelets and their scientific applications*. Broken Sound Parkway NW: Chapman and hall/CRC, 2nd ed. ed., 2008.

[43] S. Mallat *et al.*, "A wavelet tour of signal processing: the sparce way," *AP Professional, Third Edition, London*, 2009.

[44] T. Nguyen, *Wavelets and filter banks*. Wellesley-Cambridge Press, 1996.

[45] A. Harr, "Theorie der orthogonalen funktionensysteme," *Math Anal*, vol. 69, 1910.

[46] A. Rosebrock, *Deep learning for computer vision with python: Starter bundle*. PyImageSearch, 2017.

[47] W. S. McCulloch and W. Pitts, *A Logical Calculus of the Ideas Immanent in Nervous Activity*, p. 15–27. Cambridge, MA, USA: MIT Press, 1988.

[48] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[49] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," tech. rep., Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[50] M. Minsky and S. Papert, "Perceptrons cambridge," *MA: MIT Press. zbMATH*, 1969.

[51] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[52] T. Williams and R. Li, "Advanced image classification using wavelets and convolutional neural networks," in *2016 15th IEEE international conference on machine learning and applications (ICMLA)*, pp. 233–239, IEEE, 2016.

[53] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," *Advances in neural information processing systems*, vol. 2, 1989.

[54] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[55] M. Ranzato, Y.-L. Boureau, Y. Cun, *et al.*, "Sparse feature learning for deep belief networks," *Advances in neural information processing systems*, vol. 20, 2007.

[56] M. A. Nielsen, *Neural networks and deep learning*, vol. 25. Determination press San Francisco, CA, USA, 2015.

[57] C.-Y. Lee, P. W. Gallagher, and Z. Tu, "Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree," in *Artificial intelligence and statistics*, pp. 464–472, PMLR, 2016.

[58] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, *et al.*, "On rectified linear units for speech processing," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 3517–3521, IEEE, 2013.

[59] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

[60] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[61] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[62] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[63] S. University, "Adaptive adaline neuron using chemical memistors," 1960. Cited on page 106.

[64] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, no. 8, p. 2, 2012.

[65] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[66] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning," *Coursera, video lectures*, vol. 264, no. 1, pp. 2146–2153, 2012.

[67] F. Lin, T. Hou, Q. Jin, and A. You, "Improved yolo based detection algorithm for floating debris in waterway," *Entropy*, vol. 23, no. 9, p. 1111, 2021.

[68] K. T. Gribbon and D. G. Bailey, "A novel approach to real-time bilinear interpolation," in *Proceedings. DELTA 2004. Second IEEE international workshop on electronic design, test and applications*, pp. 126–131, IEEE, 2004.

[69] V. Patel and K. Mistree, "A review on different image interpolation techniques for image enhancement," *International Journal of Emerging Technology and Advanced Engineering*, vol. 3, no. 12, pp. 129–133, 2013.

[70] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3, pp. 2149–2154, IEEE, 2004.

[71] L. Joseph, *Robot operating system (ros) for absolute beginners: Robotics Programming Made Easy*. India: Apress, 2018.

[72] Y. LeCun *et al.*, "Generalization and network design strategies," *Connectionism in perspective*, vol. 19, no. 143-155, p. 18, 1989.

[73] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[74] G. Lee, R. Gommers, F. Waselewski, K. Wohlfahrt, and A. O'Leary, "Pywavelets: A python package for wavelet analysis," *Journal of Open Source Software*, vol. 4, no. 36, p. 1237, 2019.

[75] L. Brigato and L. Iocchi, "A close look at deep learning with small data," in *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 2490–2497, IEEE, 2021.

[76] N. Srivastava, "Improving neural networks with dropout," *University of Toronto*, vol. 182, no. 566, p. 7, 2013.

[77] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," in *2018 international interdisciplinary PhD workshop (IIPhDW)*, pp. 117–122, IEEE, 2018.

[78] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, PMLR, 2015.

[79] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.