CLASSIFICATION OF UNDERWATER PIPELINE EVENTS USING DEEP CONVOLUTIONAL NEURAL NETWORKS

Felipe Rembold Petraglia

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: José Gabriel Rodríguez Carneiro Gomes

Rio de Janeiro
Setembro de 2017

CLASSIFICATION OF UNDERWATER PIPELINE EVENTS USING DEEP CONVOLUTIONAL NEURAL NETWORKS

Felipe Rembold Petraglia

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUACÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA

Examinada por:

[illegible]

Prof. José Gabriel Rodríguez Carneiro Gomes, Ph. D.

Diego Barreto Haddad

Prof. Diego Barreto Haddad, D. Sc.

[illegible]

Prof. Julio Cesar Boscher Torres, D. Sc.

RIO DE JANEIRO, RJ - BRASIL

SETEMBRO DE 2017

Petraglia, Felipe Rembold

Classification of Underwater Pipeline Events using Deep Convolutional Neural Networks/Felipe Rembold Petraglia. – Rio de Janeiro: UFRJ/COPPE, 2017.

XII, 56 p.: il.; 29, 7cm.

Orientador: José Gabriel Rodríguez Carneiro Gomes

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2017.

Referências Bibliográficas: p. 54-56.

1. Introduction. 2. Artificial Neural Networks. 3. Deep Neural Networks. 4. Experiments and Databases. 5. Results. 6. Conclusions. I. Gomes, José Gabriel Rodríguez Carneiro. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título

Àqueles que estiveram próximos a mim durante o período do mestrado.

Agradecimentos

Gostaria de agradecer ao professor José Gabriel Rodríguez Carneiro Gomes, meu orientador, pela paciência e atenção ao longo deste trabalho.

Agradeço também aos meus pais, Mariane e Antonio Petraglia, pelos conselhos e pela experiência compartilhada ao longo da minha trajetória acadêmica. Demonstro também minha gratidão ao meu irmão, Pedro Gabriel, pela companhia e pelo convívio harmonioso no cotidiano.

Sou extremamente grato aos professores cujas disciplinas cursei ao longo do curso de mestrado. O conhecimento que adquiri em suas aulas foi de fundamental importância para a minha formação como engenheiro. Agradeço também aos funcionários da Universidade Federal do Rio de Janeiro (UFRJ), pois suas atuações são decisivas para o sucesso dessa instituição de excelência.

Por fim, registro minha gratidão aos meus colegas de classe, em especial àqueles pertencentes ao grupo de amigos que estiveram mais próximos de mim durante a graduação e o mestrado. Tanto os momentos de estudo e de trabalho em equipe quanto os de descontração foram extremamente proveitosos em sua companhia, e estimularam de forma significativa a minha participação na UFRJ.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

CLASSIFICAÇÃO DE EVENTOS EM DUTOS SUBMARINOS UTILIZANDO REDES NEURAIS CONVOLUCIONAIS

Felipe Rembold Petraglia

Setembro/2017

Orientador: José Gabriel Rodríguez Carneiro Gomes

Programa: Engenharia Elétrica

A inspeção automática de dutos submarinos tem sido uma tarefa de crescente importância para a detecção de diferentes tipos de eventos, dos quais destacam-se armadura exposta, presença de algas, flanges e manta. Tais inspeções podem se beneficiar de técnicas de aprendizado de máquinas para classificar acuradamente essas ocorrências. Neste trabalho, apresenta-se um algoritmo de redes neurais convolucionais para classificação de eventos em dutos submarinos. A arquitetura e os parâmetros da rede neural que resultam em desempenho de classificação ótimo são selecionados. A técnica de rede neural convolucional, em comparação ao algoritmo do *perceptron* precedido por extração de *features wavelet*, apresenta desempenho superior para diferentes classes de eventos, alcançando em média acurácia de classificação de 93.2%, enquanto o desempenho alcançado pelo *perceptron* é de 91.2%. Além dos resultados obtidos no conjunto de teste, são analisadas as curvas de acurácia e de entropia cruzada obtidas para o conjunto de validação ao longo do treinamento, de modo a comparar os desempenhos de cada método e para cada classe de eventos. São também fornecidas visualizações das saídas das camadas intermediárias da rede convolucional. Essas visualizações são interpretadas e associadas aos resultados obtidos.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

CLASSIFICATION OF UNDERWATER PIPELINE EVENTS USING DEEP CONVOLUTIONAL NEURAL NETWORKS

Felipe Rembold Petraglia

September/2017

Advisor: José Gabriel Rodríguez Carneiro Gomes

Department: Electrical Engineering

Automatic inspection of underwater pipelines has been a task of growing importance for the detection of four different types of events: inner coating exposure, presence of algae, flanges and concrete blankets. Such inspections might benefit of machine learning techniques in order to accurately classify such occurrences. In this work, we present a deep convolutional neural network algorithm for the classification of underwater pipeline events. The neural network architecture and parameters that result in optimal classifier performance are selected. The convolutional neural network technique outperforms the perceptron algorithm preceded by wavelet feature extraction for different event classes, reaching on average 93.2% classification accuracy, while the accuracy achieved by the perceptron is 91.2%. Besides the results obtained in the test set, accuracy and cross entropy curves obtained in the validation set during training are analyzed, so that the performances of each method and for each event class are compared. Visualizations of the convolutional neural network intermediate layer outputs are also provided. These visualizations are interpreted and associated to the results obtained.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The intensification of subsea oil and gas field exploitation has turned the inspection of underwater pipelines into a progressively demanding task. Usually conducted with the use of remotely operated underwater vehicles (ROVs), which employ sensors and cameras and are controlled through radio or cable connections [1], visual inspection by humans is a tedious endeavor, particularly in the cases of long inspections, low image quality and search for multiple targets [2].

In contrast to ROVs, autonomous underwater vehicles (AUVs) are able to automatically detect and track underwater pipelines. In this regard, event classification methods based on machine learning can be used in order to automatically inspect the pipelines.

Classic neural network techniques, such as the multilayer perceptron (MLP), are strongly dependent on feature extraction methods, which are often manually carried out. Recently, deep learning algorithms have been able to iteratively extract their own features from original data. Such techniques usually consist of very complex models, composed of a large amount of elements and a variety of regularization techniques, such as dropout, batch-normalization and max-pooling.

These networks present convenient training properties, allowing the model to learn information from a large number of data samples before its accuracy achieves saturation. On the other hand, when a relatively small number of data samples is used in the training stage, overfitting harms the model classification precision, causing its learning curve to saturate after a small amount of training epochs.

This thesis focuses on the application of one of these recent techniques, namely convolutional neural network (CNN), to event classification. A method consisting of a perceptron preceded by a wavelet-based feature extractor is also described, and the results obtained using the deep CNN architecture and the perceptron are compared.

In order to provide information for the understanding of the neural network decisions, visualizations of the outputs of the trained CNN intermediate layers are exhibited and interpreted. Convolutional layer weights are also plotted.

1.2 Objectives

This work main objective is to develop a deep convolutional neural network architecture capable of achieving high classification accuracy for pipeline events without the need for manually selected feature extraction techniques. Whereas a variety of approaches proposing deep neural network architectures for image classification have been recently published, benchmark on models particularly for underwater pipeline event classification is relatively scarse. In this sense, this work provides a reference for deep neural network models for this type of application.

In the experiments performed, images from underwater pipelines and their surrounding environments were utilized, in order to validate the method for future applications in real-time pipeline event detection. In order to take advantage of the complex deep neural network architecture, a large number of data samples were collected and utilized.

1.3 Thesis Structure

In Chapter 2, basic neural network concepts are presented. Perceptron and multilayer perceptron (MLP) are described, as well as the backpropagation learning technique. Activation functions and regularization techniques utilized in these methods are presented, and feature extraction methods that process data to generate neural network inputs are explained.

In Chapter 3, deep neural networks are characterized. CNN, an architecture which is vastly used in image processing to explore local properties, is described. Recent regularization techniques, such as max-pooling, dropout and batchnormalization, are explained. Optimization methods utilized in this work are presented.

In Chapter 4, the experiments conducted in this work are described, as well as the generated and utilized database. The four event classes considered are detailed, and the utilized CNN and MLP architectures are specified. The windows extracted from pipeline images, used as inputs to the neural network algorithms, are characterized.

In Chapter 5, experimental results are presented. Accuracy and loss curves obtained by the CNN and by the perceptron for the four event classes are exhibited and compared. In order to assist the understanding of the CNN functioning, visualizations of intermediate layer outputs are provided and analyzed.

Finally, conclusions from this thesis and suggestions for future work are presented in Chapter 6.

Chapter 2

Artificial Neural Networks

Artificial neural networks (ANNs) [3] are computational models based on the aproximate functioning of biological human neurons. These models are statistical methods for estimating complex functions, usually utilized in data classification, prediction or regression. The units within the network are organized in different layers, so that neurons from adjacent layers are connected through weights. The connection weights can enhance or inhibit the element activation, which is determined according to a specific nonlinear function, called activation function.

The first neural network generation, proposed in 1957, was the perceptron [4], one of the simplest models for supervised learning. This algorithm consists in a binary classifier based on a linear prediction function combining a set of weights with the feature vector, used as input to the system. As can be seen in Figure 2.1, the perceptron maps an input $\boldsymbol{x}$, a real valued vector, to a binary output value $f(\boldsymbol{x})$:

$$f(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{w}^T\boldsymbol{x} + b > 0 \\ 0, & \text{otherwise} \end{cases}, \tag{2.1}$$

where $\boldsymbol{w}$ is the perceptron weight vector and b is the bias coefficient.

As Figure 2.2 shows, the perceptron algorithm separates the input hyperspace by a hyperplane, $\boldsymbol{w}^T\boldsymbol{x} + b = 0$. In this sense, each side of the hyperspace corresponds to a different class. Therefore, the single layer perceptron is only capable of learning linearly separable data patterns. In order to learn more complex patterns, additional layers with nonlinear activation functions are required.

Based on the simple perceptron structure, the MLP [5] was posteriorly proposed as a feedforward neural network composed of three or more layers of nodes in a directed graph, with each layer being fully-connected to the following one, as shown in Figure 2.3. An input and an output layer with one or more hidden layers compose the architecture.

Each element value is determined by a function of the weighted sum of the previ-

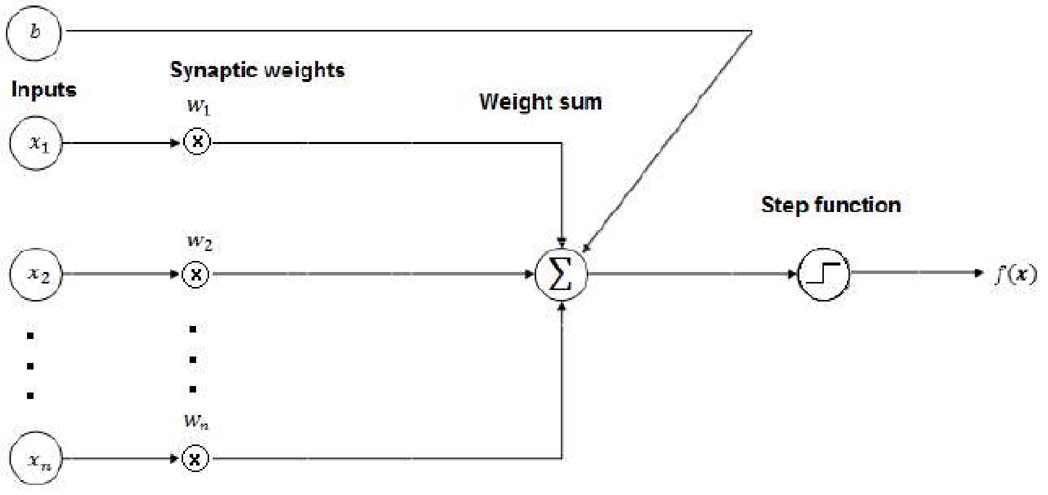

Figure 2.1: Perceptron mathematical model.

ous layer element values. Except for the output layer, which is activated by the step function, the elements present continuosly differentiable activation functions, most commonly hyperbolic tangent and logistic functions, enabling the use of gradient-based optimization methods. Therefore, the input signal is propagated through the intermediate layers, determining the output classification result.

2.1 Backpropagation

In the training phase, in order to set the optimal weight values for a supervised learning task, backpropagation [6] is used in conjunction with an optimization method, such as gradient descent. After the input vector $\boldsymbol{x}$ is propagated layer by layer and reaches the output layer, the network output y is compared to the desired output d. This comparison is performed according to a loss function $J(\boldsymbol{w}, b)$ (where the $\boldsymbol{w}$ vector contains all network parameters and b is the bias coefficient), usually set as the mean squared error over N samples in a batch:

$$J(\boldsymbol{w}, b) = \frac{1}{2N} \sum_{n=1}^{N} |d(n) - y(n)|^2. \tag{2.2}$$

The gradient of the loss function with respect to the weight vector $\boldsymbol{w}$, connecting the last hidden layer to the output layer, is then calculated. According to the gradient descent method, after being multiplied by a learning rate η, the gradient is subtracted from $\boldsymbol{w}$ in each update iteration, in order to minimize the loss function:

$$\Delta \boldsymbol{w} = -\eta \nabla J(\boldsymbol{w}) = -\eta \frac{\partial J}{\partial \boldsymbol{w}}. \tag{2.3}$$

The partial derivative of the error with respect to the weight vector is calculated

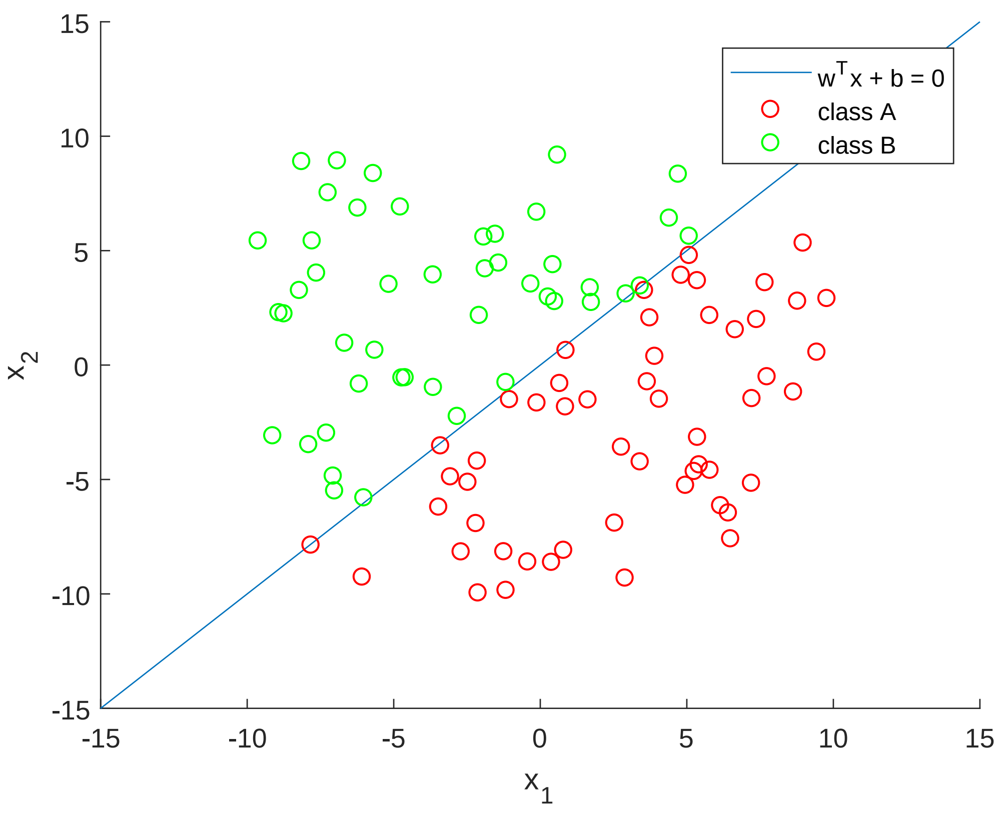

Figure 2.2: Simple perceptron classification example, with $\boldsymbol{w} = [1 \quad -1]^T$ and $b = 0$.

according to the chain rule:

$$\frac{\partial J}{\partial \boldsymbol{w}} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial \boldsymbol{w}} = \frac{1}{N}\sum_{n=1}^{N}(y(n) - d(n))\frac{\partial y(n)}{\partial \boldsymbol{w}}. \tag{2.4}$$

Since

$$y(n) = \phi(\boldsymbol{w}^T\boldsymbol{o}(n) + b), \tag{2.5}$$

where ϕ is the activation function and $\boldsymbol{o}(n)$ is the previous layer output vector, the partial derivative of y with respect to $\boldsymbol{w}$ is calculated as:

$$\frac{\partial y(n)}{\partial \boldsymbol{w}} = \phi'(\boldsymbol{w}^T\boldsymbol{o}(n) + b)\boldsymbol{o}(n), \tag{2.6}$$

where ϕ' is the derivative of the activation function cited above. A commonly used activation function, which was used in this work, is the logistic function:

$$\phi(z) = \frac{1}{1 + e^{-z}}. \tag{2.7}$$

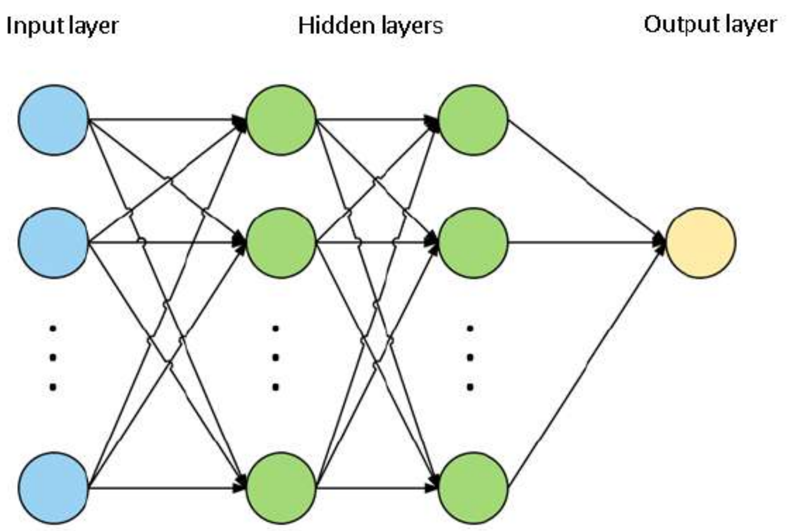

Figure 2.3: Multilayer perceptron structure.

In this case,

$$\phi^{'}(z) = \frac{\partial \phi(z)}{\partial z} = \phi(z)(1 - \phi(z)). \tag{2.8}$$

Therefore, we conclude from (2.6) that

$$\frac{\partial y(n)}{\partial \boldsymbol{w}} = \phi(\boldsymbol{w}^T \boldsymbol{o}(n) + b)(1 - \phi(\boldsymbol{w}^T \boldsymbol{o}(n) + b))\boldsymbol{o}(n). \tag{2.9}$$

From (2.3) and (2.4), the weight vector update is computed as

$$\Delta \boldsymbol{w} = -\eta \frac{1}{N} \sum_{n=1}^{N} (y(n) - d(n))\phi(\boldsymbol{w}^T \boldsymbol{o}(n) + b)(1 - \phi(\boldsymbol{w}^T \boldsymbol{o}(n) + b))\boldsymbol{o}(n). \tag{2.10}$$

Similarly, the bias coefficient update is calculated based on gradient descent:

$$\Delta b = -\eta \nabla J(b) = -\eta \frac{\partial J}{\partial b} = -\eta \frac{1}{N} \sum_{n=1}^{N} (y(n) - d(n)) \frac{\partial y(n)}{\partial b}. \tag{2.11}$$

Since

$$\frac{\partial y(n)}{\partial b} = \phi^{'}(\boldsymbol{w}^T \boldsymbol{o}(n) + b), \tag{2.12}$$

the coefficient update is given by

$$\Delta b = -\eta \frac{1}{N} \sum_{n=1}^{N} (y(n) - d(n))\phi(\boldsymbol{w}^T \boldsymbol{o}(n) + b)(1 - \phi(\boldsymbol{w}^T \boldsymbol{o}(n) + b)). \tag{2.13}$$

For the previous layer weights, the chain rule is again utilized, and the weight gradients are given by the partial derivatives of the loss with respect to the previous layer outputs, multiplied by the partial derivatives of the previous layer outputs with respect to the weights of the layer.

Let $\boldsymbol{W}$ be the previous layer weight matrix, $\boldsymbol{c}$ be its bias coefficient vector and $\boldsymbol{o}_2$ be its input vector. Hence:

$$\boldsymbol{o}(n) = \phi(\boldsymbol{W}^T \boldsymbol{o}_2(n) + \boldsymbol{c}). \tag{2.14}$$

For the i-th column of $\boldsymbol{W}$, the gradient is computed according to

$$\Delta \boldsymbol{W}_i = -\eta \nabla J(\boldsymbol{W}_i) = -\eta \frac{\partial J}{\partial o_i} \frac{\partial o_i}{\partial \boldsymbol{W}_i}, \tag{2.15}$$

where o_i is the i-th element of $\boldsymbol{o}$ and $\boldsymbol{W}_i$ is the i-th column of $\boldsymbol{W}$. Analogously to (2.6), we conclude that

$$\frac{\partial y(n)}{\partial o_i} = \phi^{'}(\boldsymbol{w}^T \boldsymbol{o}(n) + b) w_i, \tag{2.16}$$

where w_i is the i-th element of $\boldsymbol{w}$. From (2.14), we calculate

$$\frac{\partial o_i(n)}{\partial \boldsymbol{W}_i} = \phi^{'}(\boldsymbol{W}_i^T \boldsymbol{o}_2(n) + c_i) \boldsymbol{o}_2(n), \tag{2.17}$$

where c_i is the i-th element of $\boldsymbol{c}$. From (2.15), the weight matrix update is then calculated:

$$\begin{aligned} \Delta \boldsymbol{W} = -\eta \frac{1}{N} \sum_{n=1}^{N} &(y(n) - d(n)) \phi(\boldsymbol{w}^T \boldsymbol{o}(n) + b)(1 - \phi(\boldsymbol{w}^T \boldsymbol{o}(n) + b)) \\ &\cdot \boldsymbol{o}_2(n) \phi^{'}(\boldsymbol{W}^T \boldsymbol{o}_2(n) + \boldsymbol{c}) \text{diag}[\boldsymbol{w}], \end{aligned} \tag{2.18}$$

where diag[$\boldsymbol{v}$] is the diagonal matrix containing the elements of $\boldsymbol{v}$ in its main diagonal.

Similarly to (2.17), the partial derivative of the previous layer output with respect to the bias vector is obtained as

$$\frac{\partial o_i(n)}{\partial c_i} = \phi^{'}(\boldsymbol{W}_i^T \boldsymbol{o}_2(n) + c_i). \tag{2.19}$$

Hence, the bias vector update is

$$\begin{aligned} \Delta \boldsymbol{c} = -\eta \frac{1}{N} \sum_{n=1}^{N} &(y(n) - d(n)) \phi(\boldsymbol{w}^T \boldsymbol{o}(n) + b)(1 - \phi(\boldsymbol{w}^T \boldsymbol{o}(n) + b)) \\ &\cdot \text{diag}[\boldsymbol{w}] \phi^{'}(\boldsymbol{W}^T \boldsymbol{o}_2(n) + \boldsymbol{c}). \end{aligned} \tag{2.20}$$

2.2 Activation Functions

Over the years, neuron models with different activation functions have been proposed. In this section, the most commonly used activation functions will be described.

2.2.1 Logistic Function

Logistic function is described by

$$S(t) = \frac{1}{1 + e^{-t}}. \tag{2.21}$$

As can be seen in Figure 2.4, its output approaches 0 as t approaches $-\infty$, and 1 as t approaches ∞. Applied to the weighted combination of the input signal, this function introduces nonlinearity in the neural network, and limits the signal to the interval $[0, 1]$. This is an advantage comparatively to linear activation functions, which are usually associated with unstable training behavior, since neuron inputs along favored paths tend to increase unlimitedly, as these functions are not normalizable.

Logistic function can be seen as a smoother variant of the classic threshold neuron, which is another important advantage in machine learning tasks. Since it is differentiable, this model might be trained using backpropagation.

For $-2.5 < t < 2.5$, $S(t)$ presents an approximately linear behavior, which is why this interval is called linear regime. As saturation begins, the growth slows, and, at maturity, growth stops.

A major disadvantage of this activation function is that it is not centered at zero, which might restrict the weights either to positive or to negative values. Another disadvantage is that its saturation might attenuate the error gradients in case the weight values are too high or too low.

2.2.2 Softmax

Softmax is a generalization of the logistic function for multiclass classification methods. This function limits a K-dimensional vector $\boldsymbol{t}$ of arbitrary real values to a K-dimensional vector $\boldsymbol{\sigma}(\boldsymbol{t})$ of real values in the range $(0, 1)$ that add up to 1. The function is given by

$$\sigma_j(\boldsymbol{t}) = \frac{e^{t_j}}{\sum_{k=1}^{K} e^{t_k}}, \tag{2.22}$$

where each entry σ_j in $\boldsymbol{\sigma}$ and t_j in $\boldsymbol{t}$ corresponds to the j-th class. The classification result is therefore the class corresponding to the maximum value of $\boldsymbol{\sigma}(\boldsymbol{t})$.

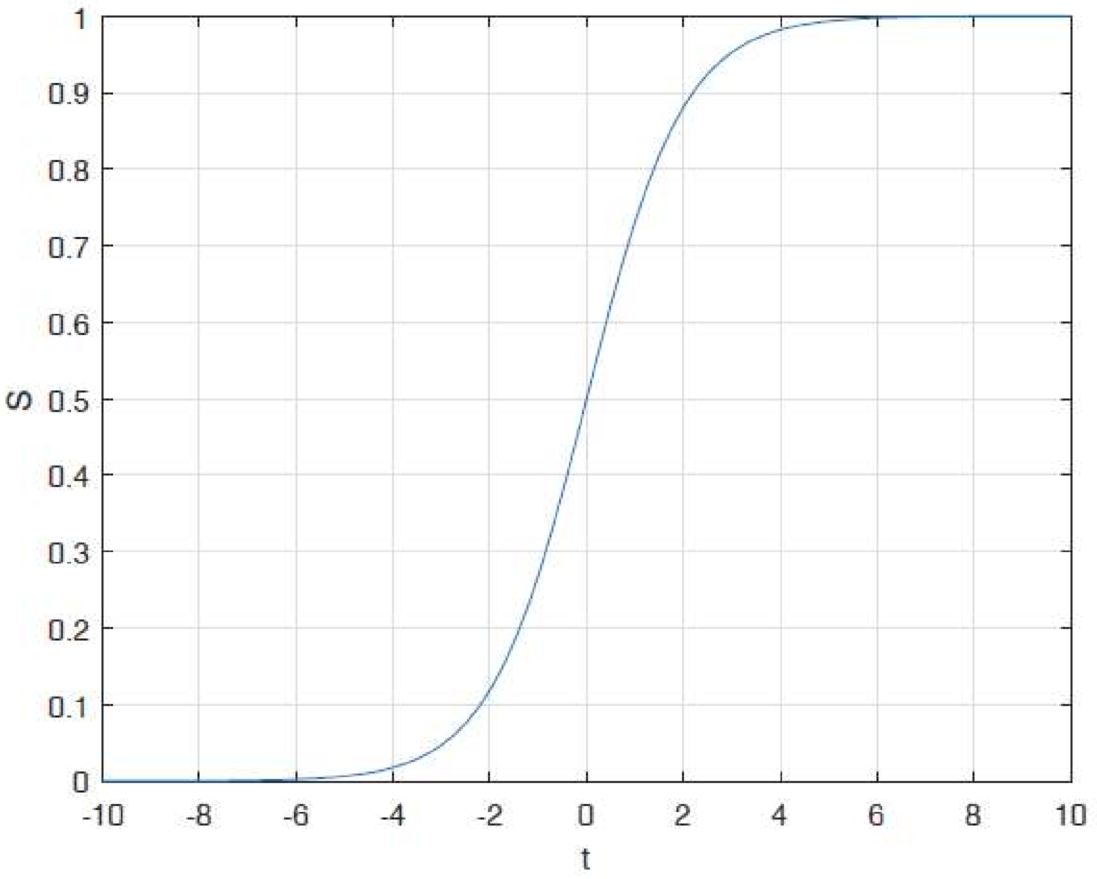

Figure 2.4: Logistic function.

2.2.3 Hyperbolic Tangent

Hyperbolic tangent function is described by

$$H(t) = \frac{e^t - e^{-t}}{e^t + e^{-t}}. \tag{2.23}$$

As shown in Figure 2.5, its output is restricted to the interval $[-1, 1]$, approaching -1 when t approaches $-\infty$, and 1 when t approaches ∞. As well as the logistic function, hyperbolic tangent belongs to the sigmoidal function class. However, an important difference between these functions is that hyperbolic tangent is antisymmetric, which explains why recently it has been preferred.

2.2.4 Rectified Linear Unit

Rectified Linear Unit (ReLU) [7] function is described by

$$R(t) = \begin{cases} 0, & \text{if } t < 0. \\ t, & \text{otherwise.} \end{cases} \tag{2.24}$$

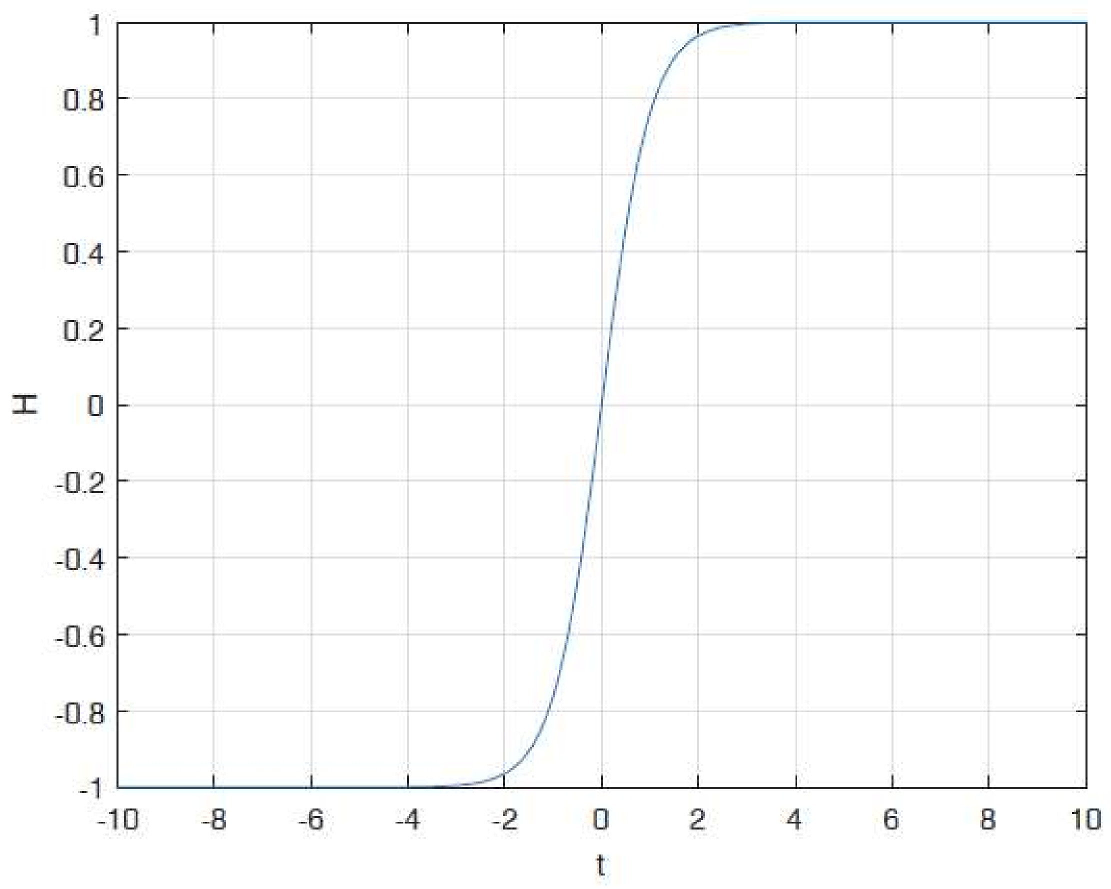

Figure 2.5: Hyperbolic tangent function.

Shown in Figure 2.6, this type of activation improves stochastic gradient convergence, since its curve does not present saturation, thereby preventing vanishing and exploding gradient problems. Comparatively to the sigmoidal functions, which are composed by exponential functions, ReLU also presents less costly implementation, since it can be performed simply with comparisons and multiplications:

$$R(t) = \max(t, 0). \tag{2.25}$$

Because of these advantages, ReLU has been widely used in deep learning algorithms, when large and complex datasets are used.

Amongst its disadvantages, its unbounded characteristics allow neuron inputs along favored paths to increase unlimitedly, which might hamper convergence stability. Another problem, referred to as dying ReLU problem, is that neurons can sometimes be pushed into states in which they become inactive for essentially all inputs. In this state, no gradients flow backward through the neuron, and hence the neuron becomes stuck in a perpetually inactive state. In some cases, large numbers of neurons in a network can become stuck in inactive states, effectively decreasing the model capacity. This problem typically arises when the learning rate is set too high.

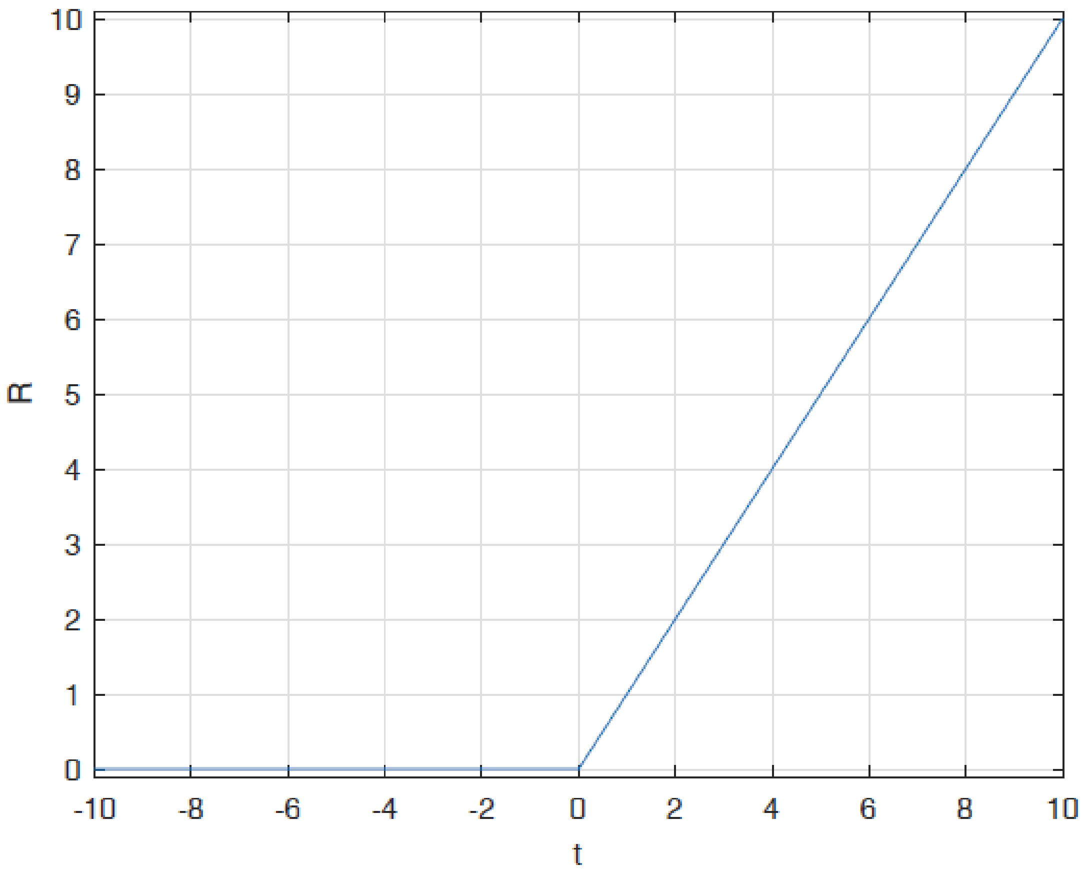

Figure 2.6: ReLU function.

2.2.5 Leaky ReLU

In order to circumvent the dying ReLU problem, leaky ReLU [8] was proposed as a function, shown in Figure 2.7, described by

$$L(t) = \begin{cases} \alpha t, & \text{if } t < 0. \\ t, & \text{otherwise,} \end{cases} \tag{2.26}$$

where α is tipically equal to 0.01, allowing a small, non-zero gradient when the unit is not active. Parametric ReLU (PReLU) [9] takes this idea further by turning the leakage coefficient into a parameter that is learned along with the other neural network parameters.

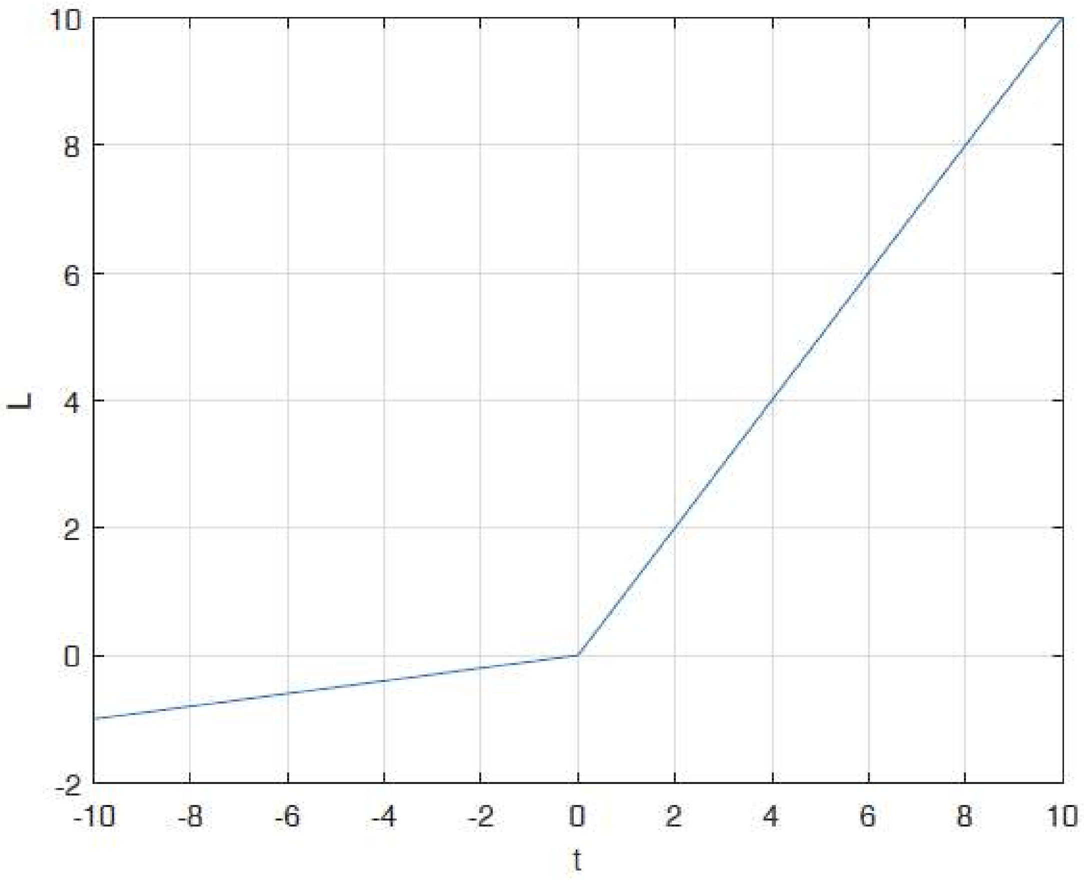

Figure 2.7: Leaky ReLU function.

2.3 Regularization Techniques

Convergence and generalization are two important neural network capabilities. In classification tasks, convergence is the machine learning algorithm ability to accurately approximate the function that maps the input patterns to their classes. Depending on the database complexity, a large number of neurons is required for the neural network to achieve good approximations, specially if the number of utilized data samples is large.

Generalization, on the other hand, is the model ability to respond correctly to samples which are different from the ones utilized in the training set. When focusing on convergence capability, a possible error is to use an excessively large number of neurons, causing the model to generate biased approximations with respect to the samples used during learning stage. This effect, referred to as overfitting [10], hampers the network generalization capability, undesirably modeling the noise present in the database.

For instance, Figure 2.8 shows a dataset sample distribution in a neural network input hyperspace, to be utilized in data regression. As can be noticed, data samples utilized to train the network generally present noise, instead of obeying exactly to a determined pattern. If the number of neurons utilized is too small, complex curves might not be correctly estimated, as illustrated in Figure 2.9. In this case, the model is too simple to learn enough information from the dataset, and the estimated function will not converge to the desired function. A more complex network is therefore required to process this amount of information, and an increase in the number of neurons or layers is generally capable of handling this difficulty. A more sofisticated neural network regression is shown in Figure 2.10. As can be noticed, the estimated function converges to the data distribution, as well as being able to correctly process new inputs, which are not present in the dataset used for training.

The increase in the number of elements might however cause the model to extract excessive information from the dataset, learning particular characteristics of the samples which do not correspond to the general pattern. This effect, called overfitting, is illustrated in Figure 2.11. It might be circumvented utilizing complex datasets, with big amount and diversity of samples.

Additionally, regularization techniques are utilized in order to improve the generalization capability of a learned model. In general, these methods add a term $R(\boldsymbol{w})$ to the loss function:

$$J(\boldsymbol{w}, b) = \frac{1}{2N} \sum_{n=1}^{N} |d(n) - y(n)|^2 + R(\boldsymbol{w}), \tag{2.27}$$

penalizing the exploration of certain regions of the function space used to build the model. Regularization can be used to learn simpler models, induce models to be sparse, introduce group structure into the learning problem, among other constraints [11]. In this section, traditional regularization techniques will be discussed.

2.3.1 ℓ_1 Regularization

ℓ_1 regularization [12] introduces as penalty the sum of the weight modules:

$$R_1(\boldsymbol{w}) = \lambda \sum_i |w_i|, \tag{2.28}$$

where λ is a constant called regularization factor. This restriction prevents high weight values to adjust the model excessively to the database. Causing the weight vector to become sparse, it makes the model basically invariant to noisy inputs. This regularization criterion selects the data dimensions to be explored by the model.

2.3.2 ℓ_2 Regularization

ℓ_2 regularization [12] introduces as penalty the sum of the weight squares:

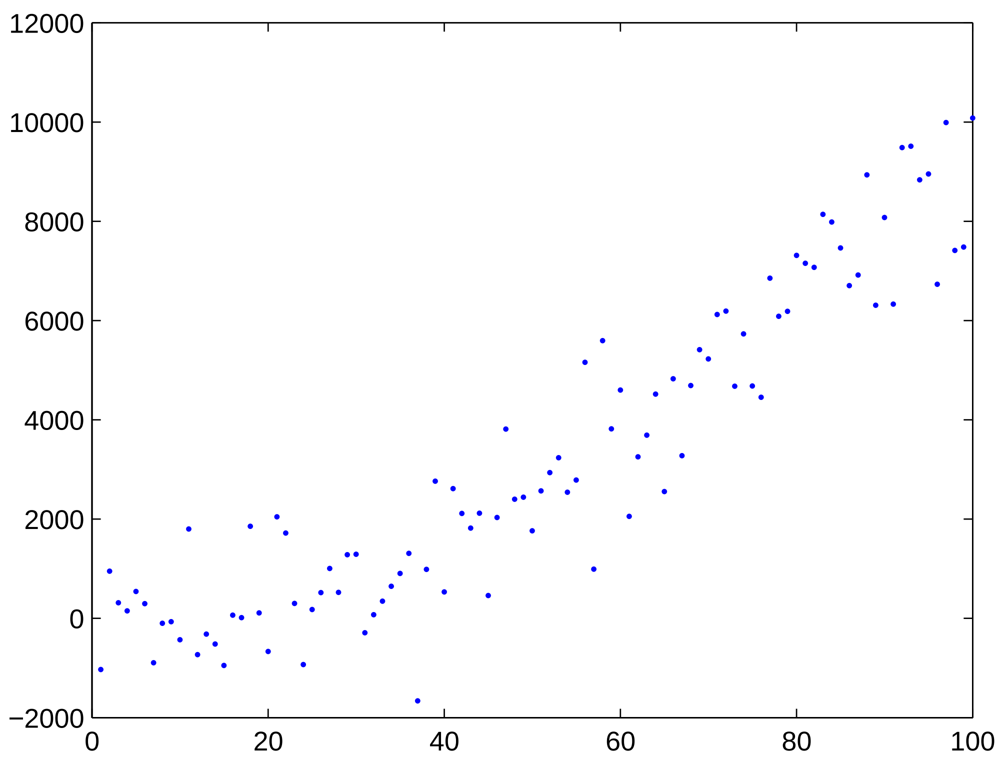

Figure 2.8: Dataset sample distribution in a neural network input hyperspace.

$$R_2(\boldsymbol{w}) = \lambda \sum_i w_i^2. \tag{2.29}$$

This regularization method penalizes high weight values, selecting diffuse weight vectors. More frequently utilized in machine learning than ℓ_1 regularization, it causes the model to consider all the data dimensions in a more balanced way, instead of selecting certain dimensions over the others.

In order to understand why ℓ_1 regularization is more likely to generate weight values close to zero, it is important to analyze its gradient. Considering that gradient descent updates the weights in the opposite direction of the gradient with a certain step size, we calculate the gradients as follows:

$$\frac{\partial R_1(\boldsymbol{w})}{\partial w_i} = \lambda \text{sign}[w_i] = \lambda \frac{w_i}{|w_i|}, \tag{2.30}$$

whereas

$$\frac{\partial R_2(\boldsymbol{w})}{\partial w_i} = \lambda w_i. \tag{2.31}$$

Figures 2.12 and 2.13 show the ℓ_1 and ℓ_2 regularization functions, as well as their respective gradient functions, for $\lambda = 1$. As can be noticed, the ℓ_1 gradient

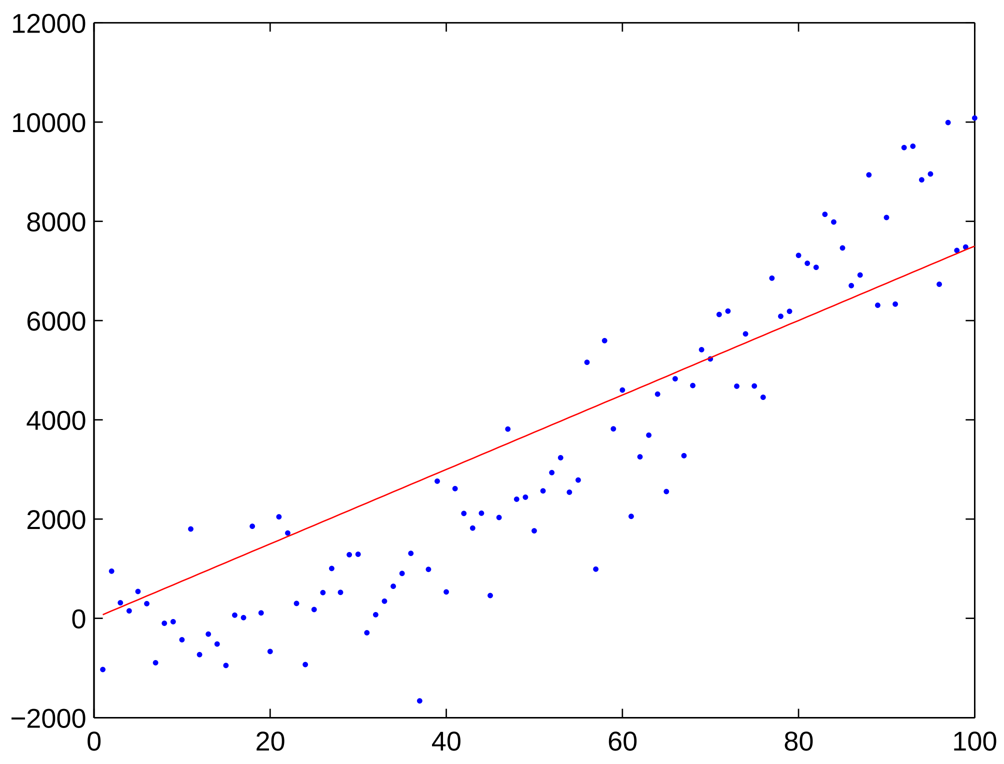

Figure 2.9: Low convergence capability data regression.

is constant. This means that ℓ_1 regularization will move any weight towards zero with constant step size, regardless of the weight value. In contrast, notice that the ℓ_2 gradient is linearly decreasing towards zero as the weights approach zero. Therefore, ℓ_2 regularization also moves the weight vector towards zero, but the steps become gradually smaller.

2.3.3 Dropout

In order to prevent overfitting, a possible solution is to combine the predictions of many different neural networks. For large models, however, this alternative would require a too long test phase, which renders unfeasible its application in certain practical tasks.

The dropout regularization technique [13] consists in randomly dropping neurons during training. In each training epoch, there is a probability p that each element is dropped, along with its connections, resulting in a thinner model, as shown in Figure 2.14. This technique approximates the effect of adding the predictions of all these thinner networks.

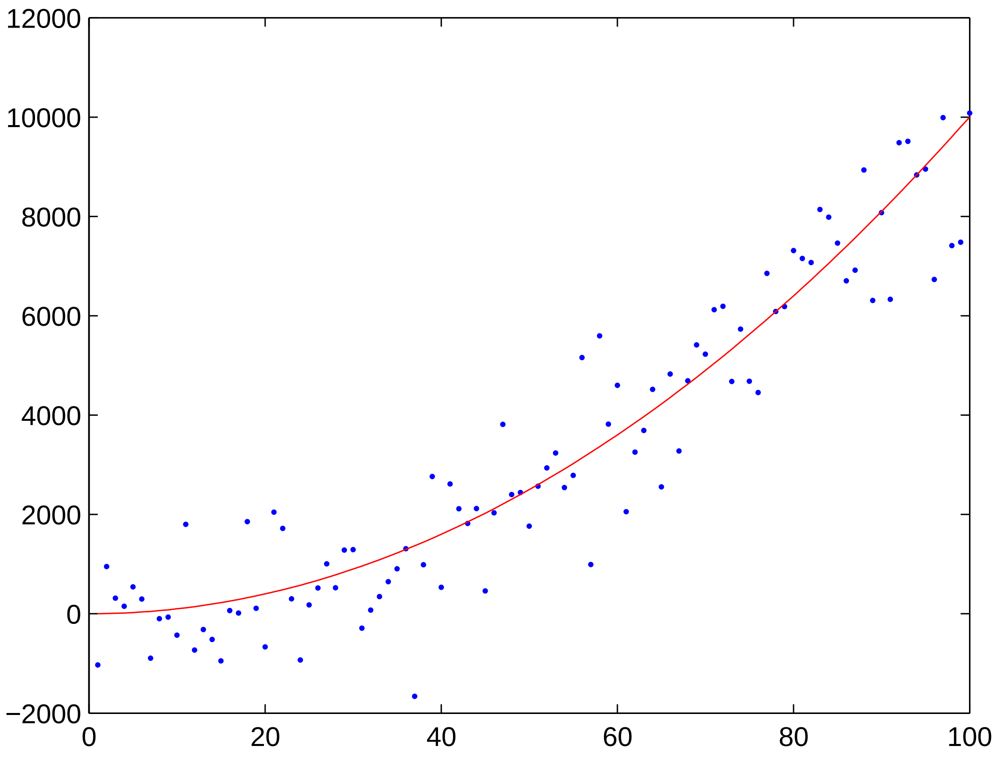

Figure 2.10: Data regression with good convergence and generalization capabilities.

2.3.4 Batch Normalization

Training deep neural networks is complicated by changes in layer input distribution during training. This problem requires the use of lower learning rates and careful parameter initialization, slowing down the training phase. This phenomenon is referred to as internal covariate shift.

In order to address this problem, batch normalization [14] is a technique which enables the use of higher learning rates and of different initialization criteria. It consists in, at each epoch, normalizing the samples within each training batch, by making their mean equal to 0 and their variance equal to 1.

Thus, for a layer with d-dimensional input $x = (x^{(1)} \ldots\ x^{(d)})$, we normalize each sample of the k-th input by the following procedure:

$$\hat{x}_l{}^{(k)} = \frac{x_l^{(k)} - \mu_x^{(k)}}{\sqrt{\sigma_x^{2(k)} + \epsilon}}, \tag{2.32}$$

where l is the sample index, and $\mu_x^{(k)}$ and $\sigma_x^{2(k)}$ are, respectively, the mean and variance of $x^{(k)}$ given by

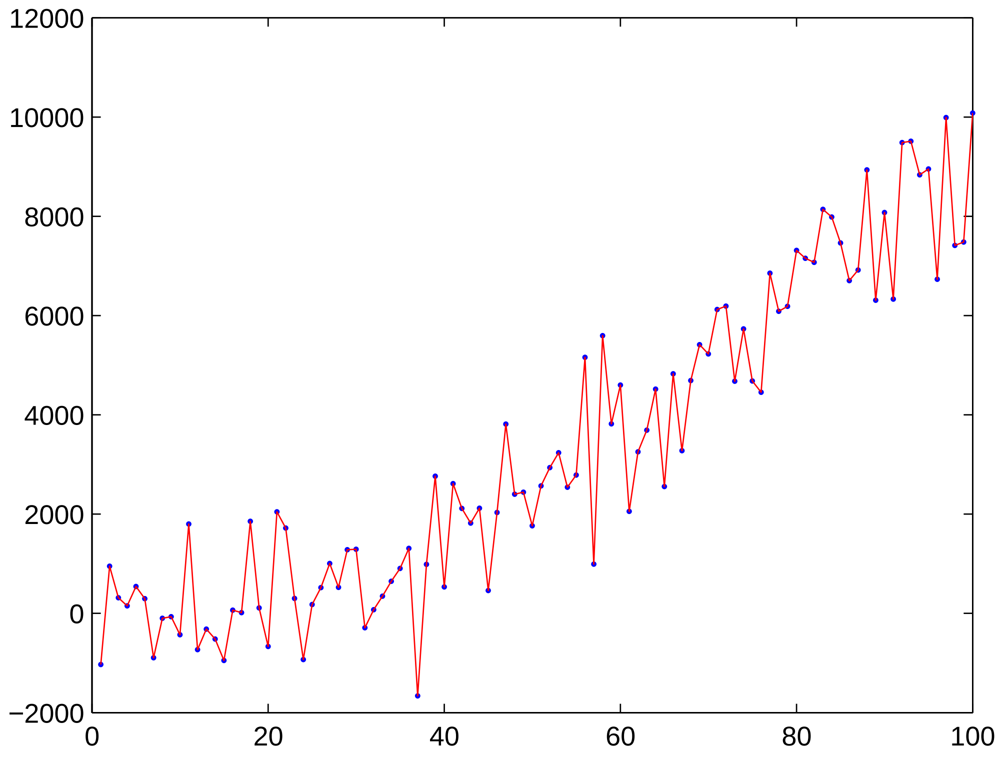

Figure 2.11: Low generalization capability data regression.

$$\mu_x^{(k)} = \frac{1}{m}\sum_{i=1}^{m} x_i^{(k)}, \tag{2.33}$$

and

$$\sigma_x^{2(k)} = \frac{1}{m}\sum_{i=1}^{m}(x_i^{(k)} - \mu_x^{(k)})^2, \tag{2.34}$$

where m is the number of samples in each batch and ϵ is a regularization parameter. Batch normalization allows the neural network to learn mean and variance parameter estimates that are suitable for reducing internal covariate shift.

2.4 Feature Extraction

Analysis with a large number of variables generally requires large amounts of memory and of computational power. Moreover, it may cause a classification algorithm to overfit to training samples and generalize poorly for new data.

Related to reducing the number of dimensions of input vectors, feature extraction is a technique that starts from an initial set of measured data and combines its samples to derive a smaller number of values, known as features, intended to be

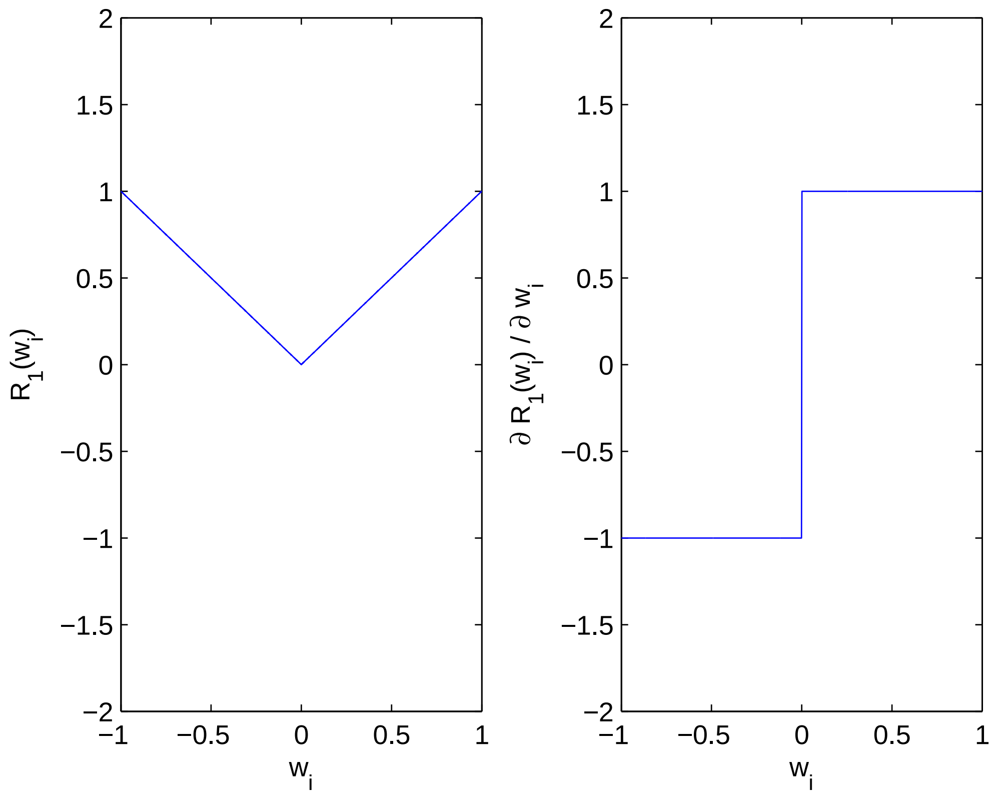

Figure 2.12: ℓ_1 regularization and its derivative function.

informative and non-redundant, thus leading to a faster learning process.

In this work, we adopted features based on the statistics of the coefficients of 2D wavelet transform of each image sample. In the next sections, we present the main concepts related to the adopted feature extraction approach.

2.4.1 Wavelet-based Feature Extraction

The regions of the images containing the events that we wish to classify in this work have structural correlations that can be well represented by first- and second-order statistics of some of their wavelet coefficients. Thus, it is possible to extract features in the wavelet domain that yield a concise representation of the different events. In this dissertation, the discussion will be restricted to wavelet-based features, although these insights can be extended to other transform-domain representations as well, such as the one resulting from applying Gabor filter banks.

Figure 2.15 illustrates the steps that compose the feature extraction method. Initially each block of the image is converted to gray scales; then the wavelet is applied to the resulting image and features of each sub-image are extracted. The wavelet transform and the statistics for the coefficients employed in this work are described in the remainder of this chapter.

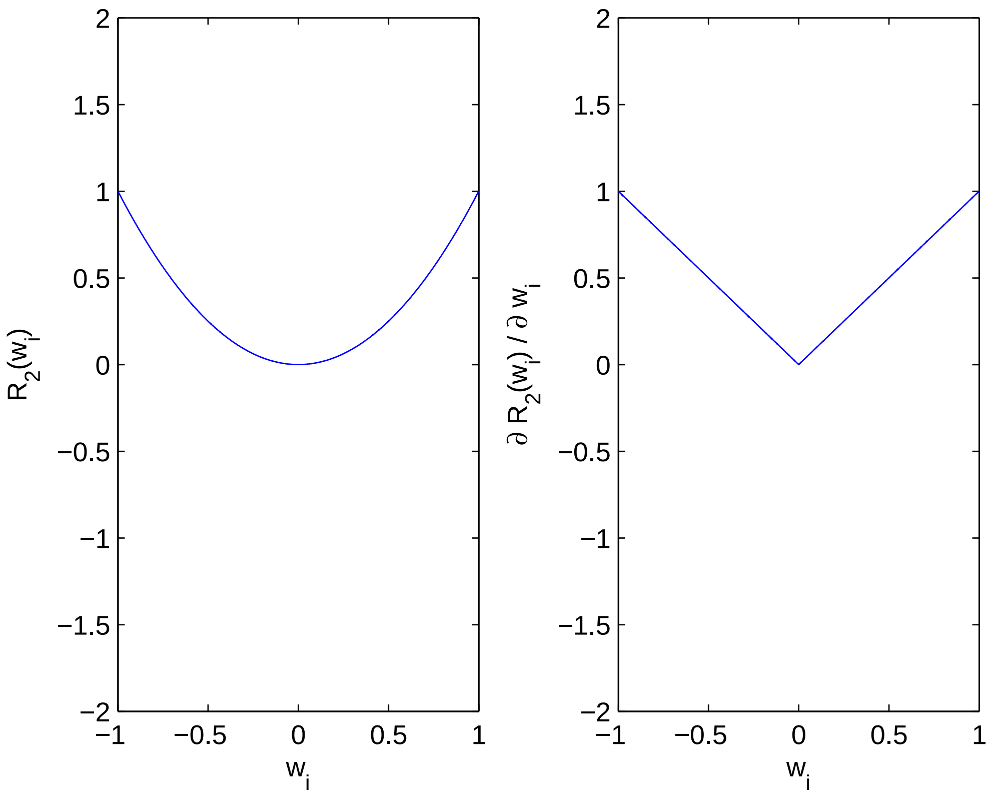

Figure 2.13: ℓ_2 regularization and its derivative function.

2.4.2 Wavelet Transform

We present in this section a summary of the mathematical concepts related to the expansion of discrete signals in series, especially the expansion carried out by the discrete wavelet transform (DWT) [15], [16], [17], [18]. Let x be a signal that belongs to the Hilbert space of complex finite energy series $\ell_2(\mathbb{Z})$. The problem of linear expansion of x is to find a set of signals $\{\varphi_i\}$ belonging to $\ell_2(\mathbb{Z})$ so that we can write x as the linear combination:

$$x = \sum_i \alpha_i \varphi_i. \tag{2.35}$$

If the set $\{\varphi_i\}$ is linearly independent (that is, $\sum_i \alpha_i \varphi_i = 0$ if and only if $\alpha_i = 0$ for all i) and spans $\ell_2(\mathbb{Z})$ (that is, if any signal $x \in \ell_2(\mathbb{Z})$ can be expanded as in (2.35)), then $\{\varphi_i\}$ is by definition a basis of $\ell_2(\mathbb{Z})$. If the basis is orthonormal (that is, if $< \varphi_i, \varphi_j >= \delta_{i-j}$, where δ_k is the unit impulse function and $< x, y >$ is the inner product of x and y), then the coefficients of (2.35) are calculated as:

$$\alpha_i =< x, \varphi_i > . \tag{2.36}$$

In the more general case, where $\{\varphi_i\}$ is a non-orthogonal basis, there will be a single dual basis $\{\tilde{\varphi}_i\}$ such that $< \varphi_i, \tilde{\varphi}_j >= \delta_{i-j}$. The coefficients in (2.35) are

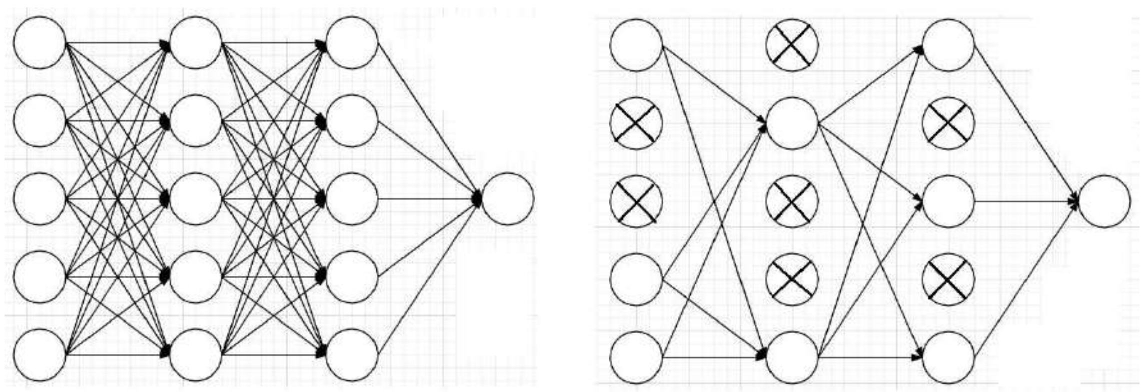

Figure 2.14: (a) Standard neural network. (b) Neural network after applying dropout.

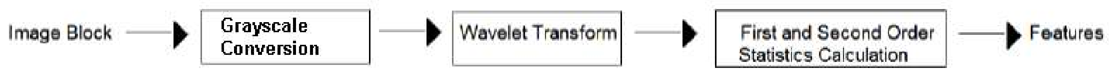

Figure 2.15: Feature calculation based on first- and second-order statistics wavelet coefficients.

calculated as

$$\alpha_i =< x, \tilde{\varphi}_i > . \tag{2.37}$$

Functions $\{\varphi_i\}$ and $\{\tilde{\varphi}_i\}$ form a pair of biorthogonal bases. The representation of a signal x as in (2.35) is done in order to analyze or process the signal in the transform domain. According to the bases $\{\varphi_i\}$ and $\{\tilde{\varphi}_i\}$ used in the signal analysis, the coefficients of the Fourier, Wavelet, Gabor, and many other transforms are calculated. The selection of the more adequate basis is done according to the application, and often depending on the type of signal to be processed. In several applications, such as denoising and signal compression, a good basis is one that allows a compact representation, or one with reduced processing cost. For practical reasons, it is generally desirable that the basis components be related to each other through simple operations, such as time-shifting, scaling and modulation. The decomposition of a signal into multiple complementary subspaces may be implemented through appropriate multi-channel filter banks, called analysis filter banks. Each channel implements the projection of the original signal into a subspace, so that this signal can be reconstructed by the summation of its projections through a synthesis filter bank, as illustrated in Figure 2.16, for a two-channel decomposition. The analysis and synthesis filter banks that provide an output signal equal to the input signal (possibly with a delay) are called perfect reconstruction filter banks. To construct multi-channel filter banks, it is possible to use the two-channel analysis filter bank of Figure 2.16 as an elementary block, from which more complex structures can be constructed by cascading these filters in binary trees. Among the many possible tree structures, the one shown in Figure 2.17 is of particular importance because of its mathematical and practical characteristics. This structure, which successively subdivides the band of lower frequencies into two subbands, each one containing approximately half the input signal spectrum, is called analysis filter bank in octaves or dyadic. The corresponding decomposition of a signal $x(k)$ is known as its DWT, where the coefficients α_J are called the scale or approximation coefficients and the coefficients β_l, $l = 1, \cdots, J$ are known as detail or wavelet coefficients.

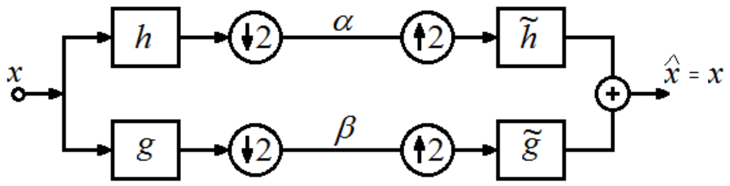

Figure 2.16: Decomposition of a signal $x(k)$ by a perfect reconstruction two-channel filter bank.

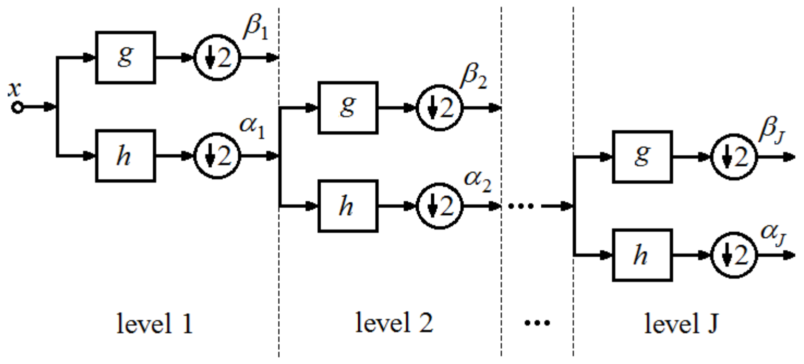

Figure 2.17: Filter bank structure of a dyadic decomposition with J levels.

If the dyadic tree of Figure 2.17 contains J levels, the input spectrum will be divided into a low frequency approximation band $[0, \pi/2^J]$ and J detail bands $[\pi/2^{J-j+1}, \pi/2^{J-j}]$ with $j = 1, 2, \cdots, J$, as shown in Figure 2.18. The DWT decomposition properties will depend on the number of decomposition levels J, on the orders of the analysis filter pair h and g of Figure 2.16 and on the family they belong to, such as Haar, Daubechies, Coiflets, Symmlets, among others. The Daubechies filters were employed in this work, since they present maximally flat responses at their pass and stop bands and, thus, avoid undesirable ripples in the frequency responses of the analysis filters (Figure 2.17), which might cause mixture of the signal contents in the different frequency subbands.

The DWT of a two dimensional image can be defined by applying the dyadic decomposition of Figure 2.17 to each dimension (rows and columns) separately. The DWT decomposition results in a set of independent spatially oriented frequency channels, which represent spatial and spectral localized components of the image.

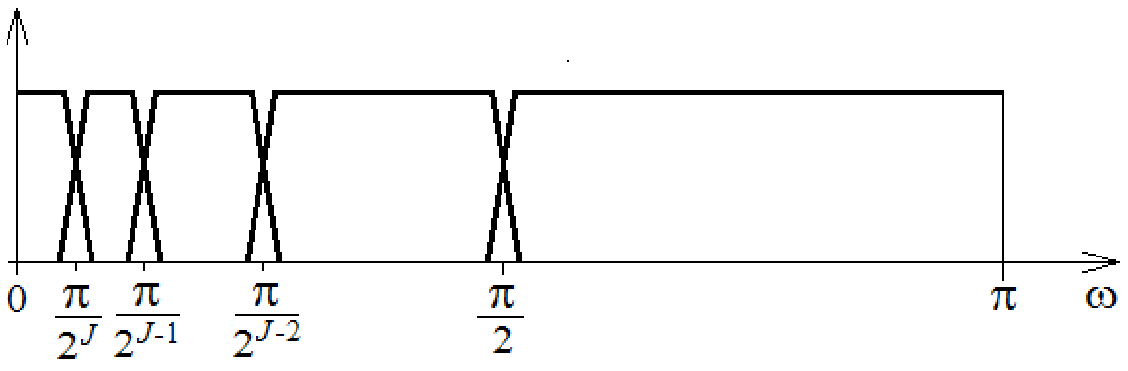

Figure 2.18: Spectrum division by a dyadic decomposition with $J = 4$ levels.

The frequency subbands, labeled as LL$_j$, LH$_k$, HL$_k$ and HH$_k$, where $k = 1, 2, ..., J$, are illustrated in Figure 2.19, for $J = 3$, where the subscript k denotes the decomposition level and J is the largest scale in the decomposition. These subbands contain different information about the image. The lowest frequency band LL$_j$ represents a coarse approximation of the image, while the LH$_k$, HL$_k$ and HH$_k$ subbands represent the horizontal, vertical and diagonal high-frequency components of the image, respectively. At the k-th decomposition level, the highest frequency band is HH$_k$, and the LL$_k$ subband is further decomposed into sub bands LH$_{k+1}$, HL$_{k+1}$ and HH$_{k+1}$.

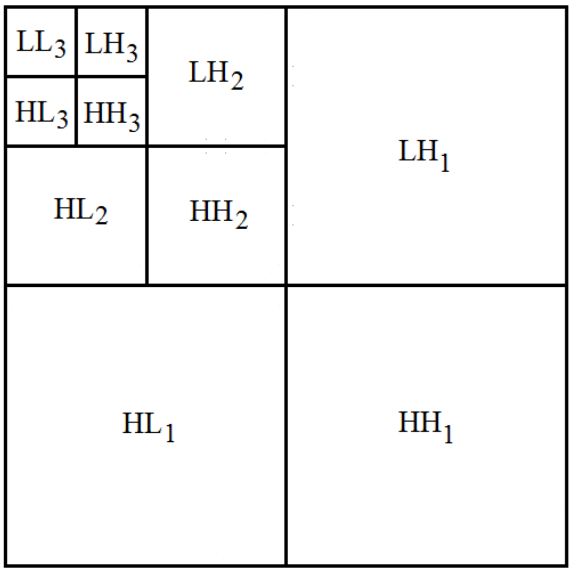

Figure 2.19: Two dimensional discrete wavelet transform with $J = 3$ levels.

2.4.3 Features based on First- and Second-Order Statistics

The texture of an image region is characterized by its smoothness, homogeneity, coarsity, presence of different types of borders and patterns. In order to describe a texture, different features can be calculated, which exploit the space relations among the image pixels. In this section, we describe a set of features obtained from the DWT of the zero-mean grayscale representation of the image region being analyzed. Such features are based on first- and second-order statistics of the DWT coefficients of each subband (see Figure 2.19).

For a given image region, of size $M \times M$, being analyzed, let $y_{j,k}(l, m)$, of size $M/2^k \times M/2^k$, represent the pixels of the approximation subband LL_k for $j = 1$, and of the detail subbands LH_k, HL_k and HH_k, for $j = 2, 3, 4$, respectively, at scale k. The first-order statistics employed in our work were the mean values of the coefficients of the sub-images at each scale (decomposition level) k, given by

$$\mu_{j,k} = \frac{1}{(M/2^k)^2} \sum_{l=1}^{M/2^k} \sum_{m=1}^{M/2^k} y_{j,k}(l, m), \tag{2.38}$$

for $j = 1, \cdots, 4$ and $k = 1, \cdots, J$. Since the mean value of the grayscale fullband image was subtracted (in order to obtain a zero mean image), $\mu_{1,1}$ is always equal to zero. Thus, the first-order statistics employed as input features were the mean-values of the other $4J - 1$ sub-images.

The second-order statistics (variances) of the sub-images $y_{j,k}(l, m)$, given by

$$\sigma_{j,k} = \frac{1}{(M/2^k)^2} \sum_{l=1}^{M/2^k} \sum_{m=1}^{M/2^k} (y_{j,k}(l, m) - \mu_{j,k})^2 \tag{2.39}$$

were also computed, leading to 4 features per wavelet level.

Thus, a total of $8J - 1$ features based on first and second-order statistics of the wavelet coefficients of each grayscale image of the region being analyzed were employed as input features in this work.

Chapter 3

Deep Neural Networks

Deep neural networks are artificial neural networks with multiple layers of nonlinear processing units. Because of their sofisticated architectures, these models can converge faster to approximations of complex functions. Moreover, the use of signal processing methods for feature extraction is generally unnecessary, since the initial layers often translate the data into compact intermediate representations, deriving layered structures that remove redundancy in representation.

Because of the large number of elements in deep neural networks, regularization and optimization methods are extremely important to prevent these models from overfitting.

3.1 Convolutional Neural Network

Convolutional neural network [19] is an architecture which explores spatial properties in samples in order to take advantage of the fact that the input consists of images. Its layers have neurons arranged in three dimensions: width, height and depth. Each neuron is only connected to a small region of the layer before it, called receptive field, instead of being connected to all the neurons in the layer, as in the multilayer perceptron.

This connectivity pattern between neurons is inspired by the organization of the animal visual cortex [20]. Individual cortical neurons respond to stimuli in restricted regions of space, and the receptive fields of different neurons partially overlap, so that they tile the visual field. The response of an individual neuron to stimuli within its receptive field can therefore be approximated mathematically by a convolution operation.

The convolutional neural network is also known as shift invariant or space invariant [21] artificial neural network, which is named based on its shared-weight architecture and translation invariance characteristics.

Among the layers utilized to build convolutional neural networks, the three main types are convolutional, pooling and fully-connected layers.

3.1.1 Convolutional Layer

The convolutional layer parameters consist of a set of learnable filters. Every filter is small along width and height, and its depth is equal to the input depth. During the forward pass, each filter is slided across the width and height of its input, and, in each position, the dot products between the filter parameters and the inputs are computed, which characterizes a convolution. As we slide the filters along the input volume, two-dimensional activation maps are produced, exhibiting the patterns that the filters are most sensitive to.

In order to calculate the size of the layer output volume, three hyperparameters are determinant: depth, stride and zero-padding. The depth corresponds to the number of filters to be used, each learning to look for different characteristics in the input, such as oriented edges.

The stride with which the filter is slided must also be specified. When the stride is equal to 1, for instance, the filter is slided one pixel at a time. When the stride is 2, the filter jumps two pixels between each dot product. Strides equal or greater than 3 are also possible, although less commonly used. The greater the stride, smaller are the output width and height.

It might also be convenient to pad the input volume with zeros around the border. The zero-padding parameter allows the output size to be controlled. Commonly, it is used to preserve input height and width, so that the output has the same spatial size.

Thus, each spatial dimension of the output volume can be calculated as follows:

$$D = (W - F + 2P)/S + 1, \tag{3.1}$$

where W is the input volume relative dimension, F is the receptive field size of the convolutional layer neurons, P is the zero-padding value utilized and S is the stride.

Figure 3.1 shows an example of spatial arrangement with only one dimension. In this example, the input volume size is equal to 5, the receptive field size is equal to 3, the padding value is 1 and the stride is 1. Therefore, the output size is equal to $(5 - 3 + 2)/1 + 1 = 5$.

Figure 3.2 illustrates an example in which the stride is 2. Consequently, the output size is equal to $(5 - 3 + 2)/2 + 1 = 3$.

Figure 3.3 shows a three-dimensional convolutional layer. As can be noticed, the input height and width are equal to 5, there are two filters with spatial dimensions equal to 3, the stride is 2 and the zero-padding parameter is 1. Consequently, the

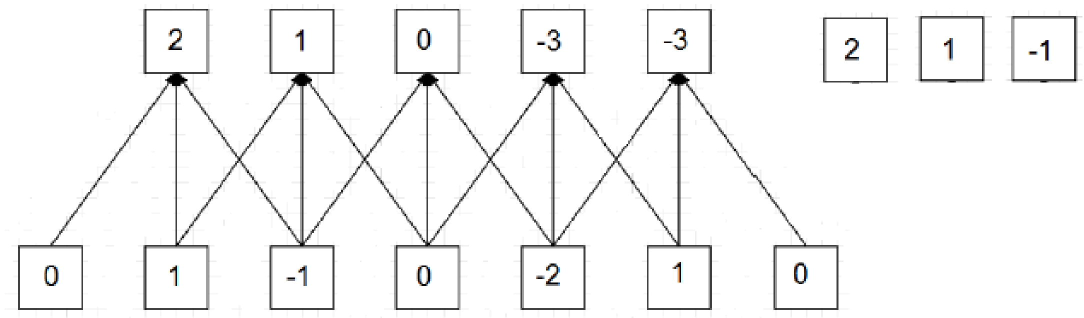

Figure 3.1: Illustration of arrangement with only one spatial dimension. The neuron weights in this example are $[2, 1, -1]$.

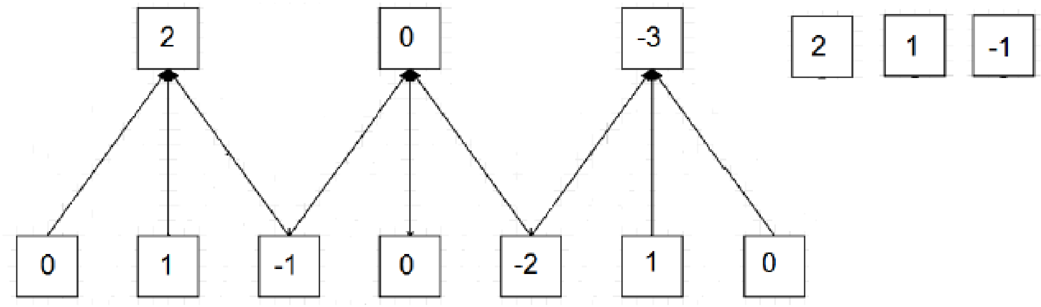

Figure 3.2: Illustration of arrangement with only one spatial dimension. In this example, the stride is equal to 2.

output spatial dimensions are equal to $(5 - 3 + 2)/2 + 1 = 3$, whereas its depth is equal to 2, which corresponds to the number of filters. The filter depth is necessarily equal to 3, which is the depth of the input volume.

After each convolutional layer, rectified linear units are used, in order to introduce nonlinearities and, consequently, improve the feature extraction process.

3.1.2 Pooling Layer

Models with very large number of neurons are susceptible to overfitting, and their training presents high computational complexity. Between successive convolutional layers, a pooling layer is commonly inserted in order to reduce the spatial size of the representation. The pooling layer operates independently on every depth slice of the input and resizes it spatially.

Commonly utilized, max-pooling extracts the maximum value from every square of elements where it is applied, so that downsampling is performed. Whereas the depth dimension remains unaltered, each output spatial dimension is equal to

$$D = (W - F)/S + 1, \tag{3.2}$$

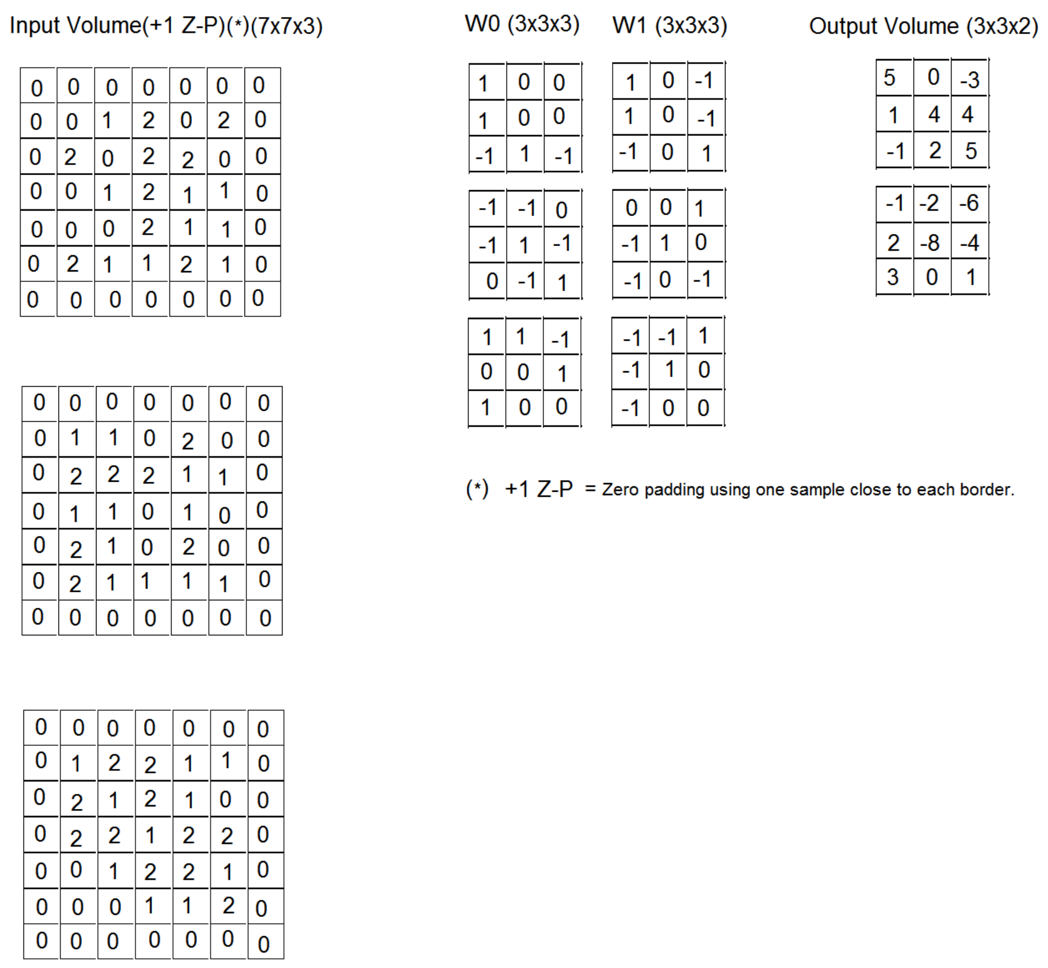

Figure 3.3: Three-dimensional convolutional layer.

where W is the input dimension, F is the operator spacial extent and S is the stride.

Figure 3.4 shows an example of 2×2 max-pooling operation, with stride equal to 2. The input width and height are equal to 4. Consequently, the output width and height are equal to $(4-2)/2+1=2$.

Figure 3.5 illustrates an example of 2×2 max-pooling operation over a three-dimensional volume. The stride is equal to 2, and the input size is 128×128×32. Notice that the depth dimension remains unaltered, whereas the output width and height are equal to $(128-2)/2+1=64$.

As well as max-pooling, batch normalization is often performed in between convolutional layers, in order to prevent overfitting.

3.1.3 Fully-Connected Layer

As in the multilayer perceptron, neurons in a fully connected layer have full connections to all activations in the previous layer. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

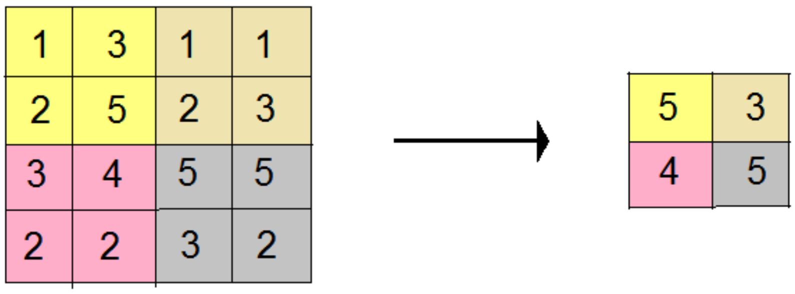

Figure 3.4: Max-pooling with 2×2 filters and stride equal to 2.

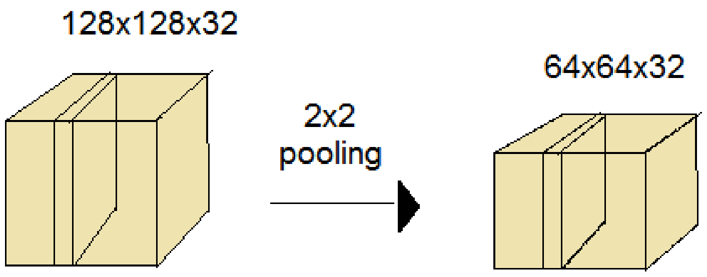

Figure 3.5: Max-pooling with 2×2 filters and stride equal to 2, over a three-dimensional volume.

Whereas the convolutional layer output represents high-level features from the data, the fully-connected layers flatten these representations and learn nonlinear combinations from them, in order to perform the final classification task. Commonly, multiple fully connected layers are sequentially utilized, so that the model can learn more complex patterns from the data samples.

3.2 Optimization Methods

Mathematical optimization is the selection of the best value for a variable from a set of available alternatives. Frequently, it consists in minimizing a real function by sistematically selecting input values according to a certain criterion and computing the value of the function.

In machine learning, a variety of weight optimization criteria and optimization functions are widely used.

3.2.1 Stochastic Gradient Descent

Neural network algorithms consider the problem of estimating the weight vector $\boldsymbol{w}$ that minimizes a loss function over N training samples:

$$J(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} J_n(\boldsymbol{w}). \tag{3.3}$$

Gradient descent method [22] updates the weight vector by taking a step in the opposite direction of the loss gradient:

$$\Delta \boldsymbol{w} = -\eta \nabla J(\boldsymbol{w}) = -\eta \frac{1}{N} \sum_{n=1}^{N} \nabla J_n(\boldsymbol{w}), \tag{3.4}$$

where η is the learning rate.

For large datasets, however, computing the cost gradient based on the complete training set can be very costly, since a single step is taken for one pass over the training set. In order to address this problem, stochastic gradient descent (SGD) [23] updates the weights after each training sample, instead of accumulating the weight updates:

$$\Delta \boldsymbol{w}_n = -\eta \nabla J_n(\boldsymbol{w}). \tag{3.5}$$

Hence, the update based on a single training sample is a stochastic approximation of the cost gradient.

3.2.2 AdaGrad

One downside of SGD is that it is sensitive to the learning rate hyperparameter. AdaGrad [24], for adaptive gradient algorithm, is a modified stochastic gradient descent with adaptive learning rate, which increases it for more sparse data samples and decreases it for less sparse ones. This improves convergence performance in applications in which sparse data are more informative, such as image recognition and natural language processing.

AdaGrad still has a base learning rate η_0, but this parameter is multiplied by the elements in the diagonal of the outer product matrix

$$\boldsymbol{G} = \sum_{\tau=1}^{t} \boldsymbol{g}_\tau \boldsymbol{g}_\tau^T, \tag{3.6}$$

where t is the current iteration and $\boldsymbol{g}_\tau = \nabla J_\tau(\boldsymbol{w})$.

Hence, the diagonal elements are given by

$$G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2. \tag{3.7}$$

The weight vector is updated after every iteration, and the update is given by

$$\Delta \boldsymbol{w} = -\eta_0 \text{diag}[\boldsymbol{G}]^{-\frac{1}{2}} \circ \boldsymbol{g_\tau}, \tag{3.8}$$

where $\circ$ denotes the Hadamard product.

Therefore, the per-parameter update is given by

$$\Delta w_j = -\frac{\eta_0}{\sqrt{(G_{j,j})}} g_{\tau,j}. \tag{3.9}$$

3.2.3 RMSProp

In RMSProp [25], for root mean square propagation, the learning rate is also adapted with respect to the weight vector. In this method, the learning rate is divided by a moving average of the magnitudes of recent loss gradients.

Hence, first the running average is calculated in terms of mean-squares:

$$v(\boldsymbol{w}, t) := \gamma v(\boldsymbol{w}, t-1) + (1-\gamma)(\nabla J(\boldsymbol{w}))^2, \tag{3.10}$$

where $\gamma \in [0, 1]$ is the forgetting factor.

The weight vector update is then given by

$$\Delta \boldsymbol{w} = -\frac{\eta_0}{\sqrt{v(\boldsymbol{w}, t)}} \nabla J(\boldsymbol{w}). \tag{3.11}$$

3.2.4 Adam

Adam [26], short for adaptive moment estimation, is an update to RMSProp. In this method, moving averages of the gradients and the second moments of the gradients are used. The gradient moving average update is given by

$$m(\boldsymbol{w}, t) := \beta_1 m(\boldsymbol{w}, t-1) + (1-\beta_1)\nabla J(\boldsymbol{w}). \tag{3.12}$$

The running average of the second moment of the gradient is updated by

$$v(\boldsymbol{w}, t) := \beta_2 v(\boldsymbol{w}, t-1) + (1-\beta_2)(\nabla J(\boldsymbol{w}))^2. \tag{3.13}$$

The weight vector update is then given by

$$\Delta \boldsymbol{w} = -\eta_0 \frac{\hat{m}(\boldsymbol{w}, t)}{\sqrt{\hat{v}(\boldsymbol{w}, t)}}, \tag{3.14}$$

where

$$\hat{m}(\boldsymbol{w}, t) = \frac{m(\boldsymbol{w}, t)}{1-\beta_1}, \tag{3.15}$$

$$\hat{v}(\boldsymbol{w}, t) = \frac{v(\boldsymbol{w}, t)}{1 - \beta_2}, \tag{3.16}$$

and β_1 and β_2 are the forgetting factors for gradients and second moments of gradients, respectively.

Chapter 4

Experiments and Databases

The classifier developed in this work was used to detect four different event types. Inner coating exposure (ICE) occurs when the pipeline surface is damaged. The outer cover disruption is caused by object impact and by natural circumstances, such as waves, sea currents, among others. Visually, it can be described as a texture region containing several parallel stripes, possibly surrounded by homogeneous regions. Figure 4.1 shows ICE samples.

The presence of algae can be characterized by a variety of shapes, colors and textures. This event might hide damages on the pipeline surface, hampering their detection. Figure 4.2 shows algae samples.

Flanges are structures commonly found at pipeline junctions, and they are used for holding pipeline sections together. When they are seen from a frontal view, these events are outlined by hexagonal prisms surrounding cylinders. These formations can also be observed from a lateral view, and in that case they are characterized by thinner rectangles emerging from thicker structures. Figure 4.3 shows flange samples.

Concrete blankets (CB) are structures placed under or over the pipelines, and they are constructed to give support or to protect the pipelines from vibrations. These events are usually identified by a regular brick array. Figure 4.4 shows CB samples.

In order to train and test the implemented classification system, windows containing event samples were extracted from high resolution images (1280 $\times$ 720), thus composing databases that are used as neural network inputs. Windows that do not contain any of these classes were also extracted, to compose the negative sample database. Figure 4.5 shows negative samples.

For each event class, positive and negative samples were mixed, so that the system would perform binary classification. 60 $\times$ 60 pixel windows were extracted for ICE, algae and CB samples, whereas 80 $\times$ 80 pixel windows were extracted for flanges, due to the need to include their entire geometry in each sample.

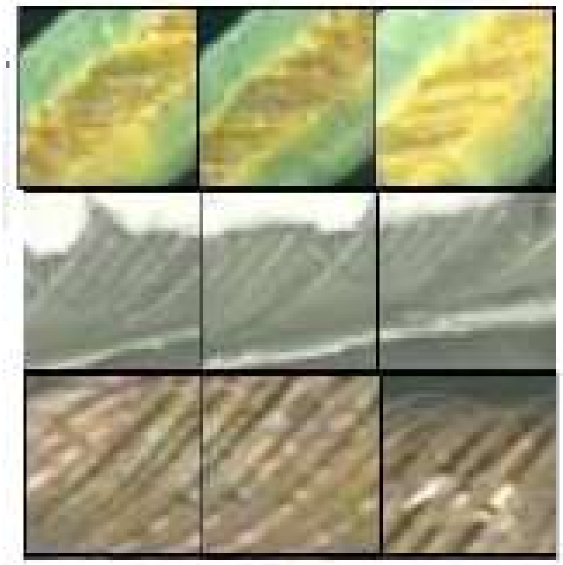
Figure 4.1: Samples from the ICE dataset.

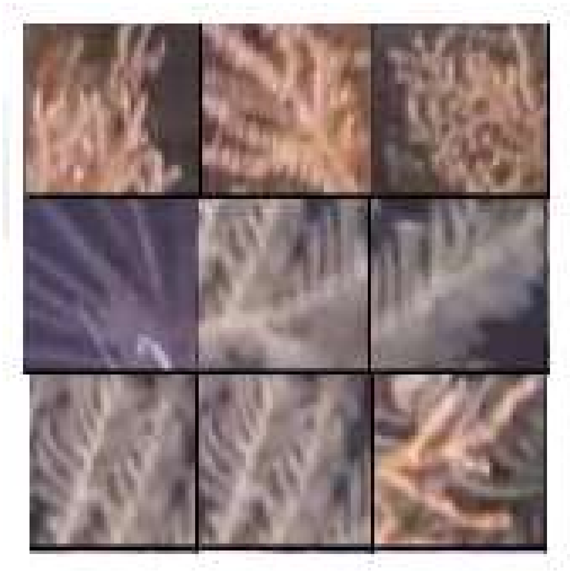
Figure 4.2: Samples from the algae dataset.

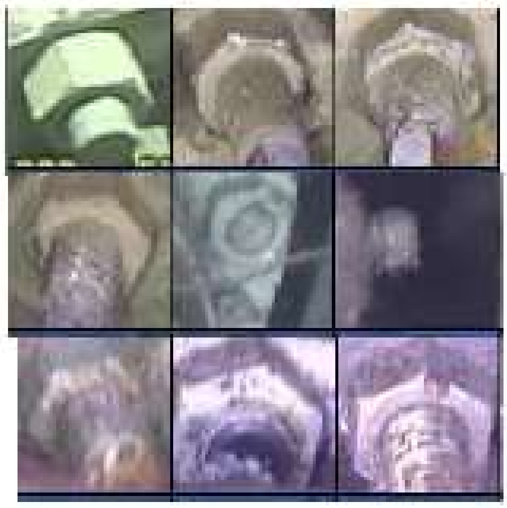
Figure 4.3: Samples from the flange dataset.

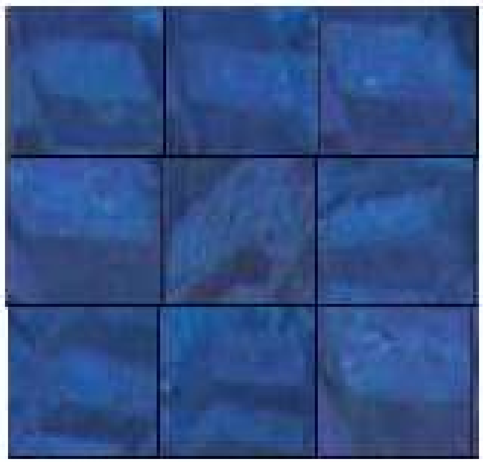
Figure 4.4: Samples from the CB dataset.

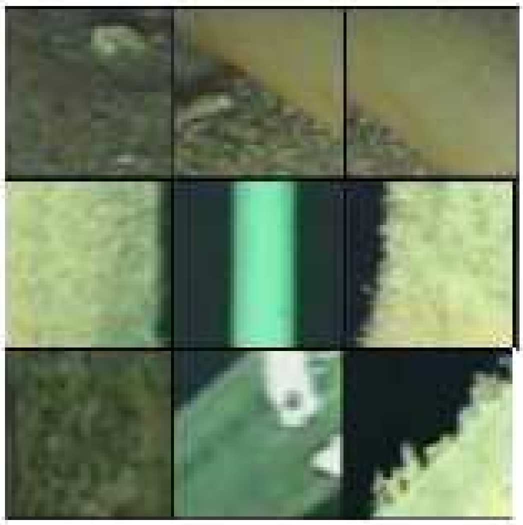

Figure 4.5: Negative samples.

As can be seen in Figure 4.6, the implemented CNN topology consists of two convolutional, two max pooling and three fully connected (FC) layers. The first layer is a 32 kernel convolutional layer, which detects relatively simple features, which can be easily recognised and interpreted. After that, a pooling layer is used, so that max pooling is applied to 2x2 regions, with strides of 2. Subsequently, another 32 kernel convolutional layer is applied, in order to detect more abstract and detailed features, which are usually present among the ones from the previous layer. A pooling layer applies max pooling once again to 2x2 regions, with strides of 2. Three FC layers are subsequently applied. They map the previous layer outputs into deeper features, to allow for a better classification performance. The FC layers have, respectively, 512 outputs, 256 outputs and 1 output.

At each convolutional layer output, batch normalization is performed. The optimizer utilized is Adam, with learning rate set to 0.001. ReLu activation function is used after each convolutional and FC layer. After every FC layer, dropout regularization is applied, with dropout probability of 50%. The loss function chosen is cross entropy. The batch size is 100. The CNN is implemented using Keras API.

The CNN classification accuracy is compared to the one from a system composed by an MLP preceded by a wavelet-based feature extractor [27]. As shown in Figure 4.7, the MLP input layer consists of 23 elements, whereas its hidden layer is composed by 12 elements. A three-level Daubechies 2 (Db2) wavelet is employed, and the mean and the variance of the wavelet coefficients at each level are used as features for the neural network.

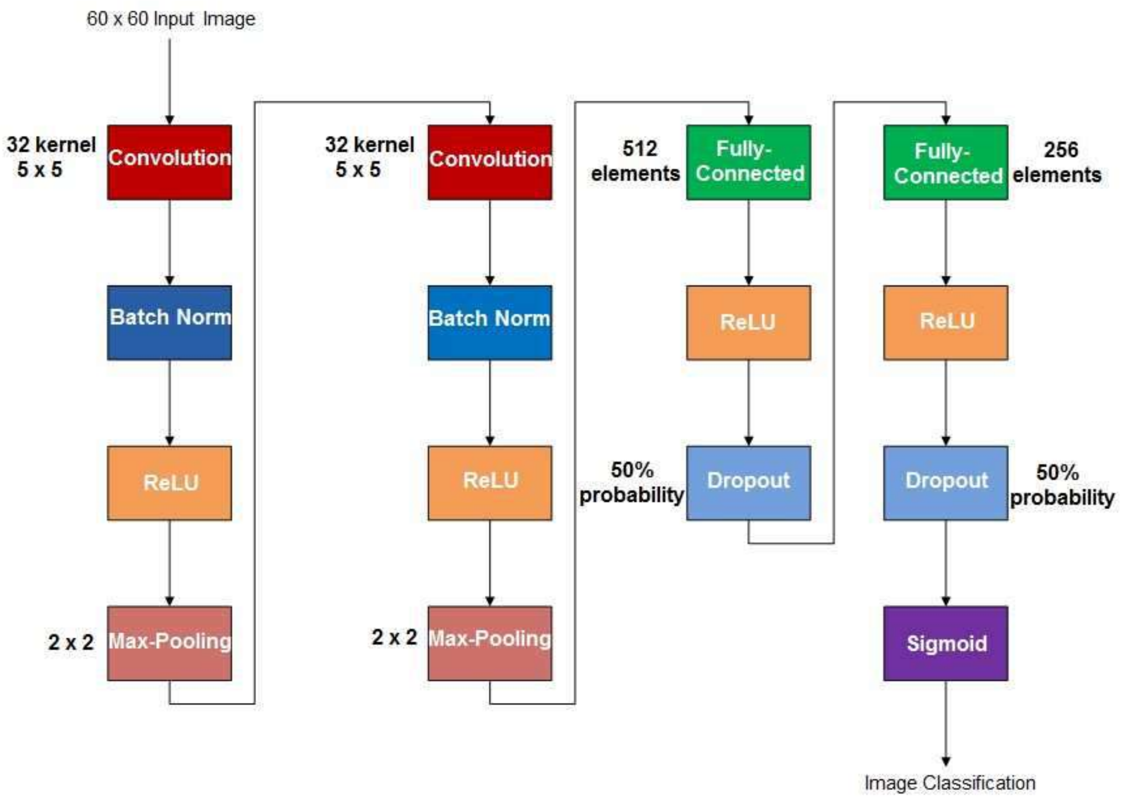

Figure 4.6: CNN architecture.

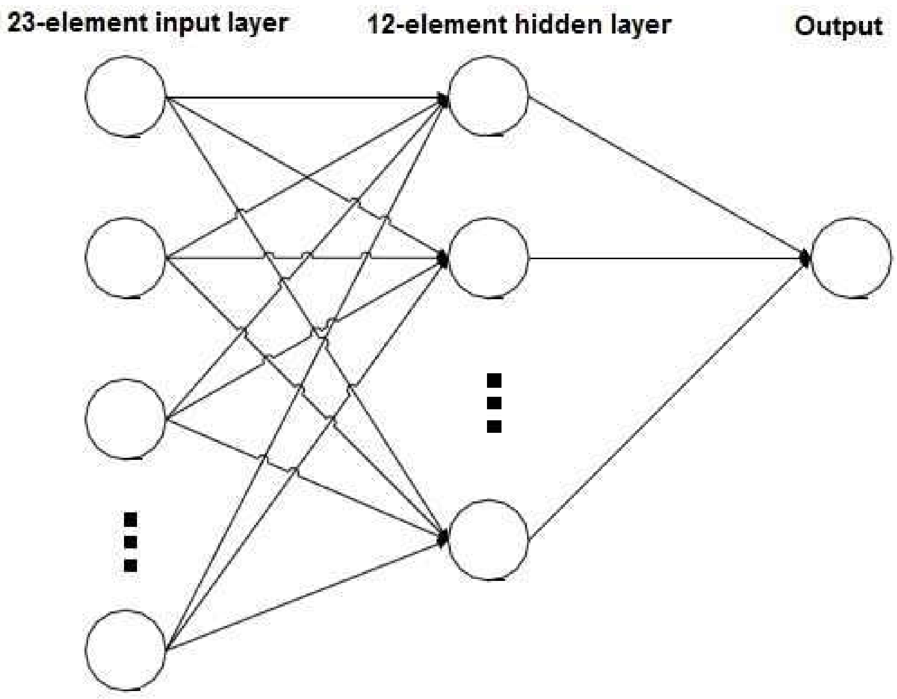

Figure 4.7: MLP architecture.

Chapter 5

Results

Results obtained by the CNN and by the MLP on the test set, for the four classes of events, are shown in Table 5.1. For each event class, 100,000 positive and 100,000 negative samples were randomly mixed. Among these, 162,000 samples were used for training, 20,000 for testing and 18,000 for validating the networks. Before being applied to the neural network input, each window is converted to grayscale, in order to eliminate color dependence.

Table 5.1: Classification accuracy for the four different event classes.

Accuracy(%)	ICE	Algae	Flange	CB
CNN	96.5	98.3	83.0	95.0
MLP	94.6	97.0	82.4	90.8

The CNN leads to a higher classification accuracy for all four event classes considered. The biggest difference obtained is in CB, for which the CNN performs 4.2% better than the perceptron. For flanges, the CNN accuracy is only 0.6% higher. This is also the event for which both methods yield the lowest classification capability, possibly because of the complexity of its structure and because of the variety of angles from which its samples were recorded. Algae is the most easily classified event, possibly because of the regularity of its aspect. For ICE, the classification accuracy is also high for both methods, and the CNN performs 1.9% better. Its aspect, with regular set of parallel stripes, is one of the reasons for the high accuracy.

Figures 5.1, 5.6, 5.11 and 5.16 show, for ICE, algae, flanges and CB, respectively, the classification accuracy on the validation set. For all classes, the accuracy curves converge within 40 epochs. The accuracy obtained for ICE has the fastest convergence, reaching around its maximum value in 30 epochs. Figures 5.2, 5.7, 5.12 and 5.17 show the cross entropy curves on the validation set, for ICE, algae, flanges and CB, respectively. The loss curves generally approach the minimum value after approximately 35 epochs.

Figures 5.4 and 5.5 show, respectively, the first and second convolutional layer

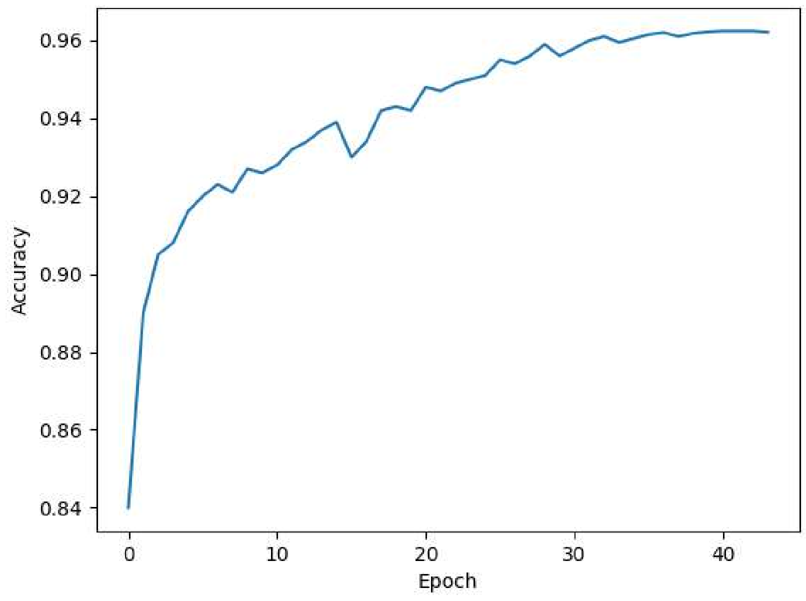

Figure 5.1: Accuracy for ICE.

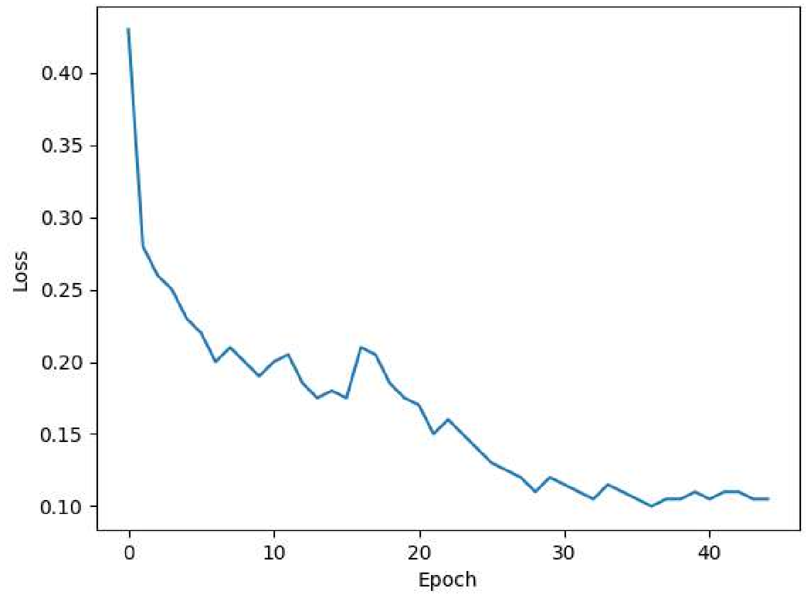

Figure 5.2: Loss function for ICE.

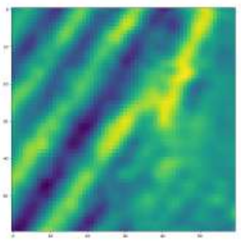

Figure 5.3: Original ICE sample.

outputs for an ICE sample, shown in Figure 5.3. As can be noticed, the convolutional layers reinforce stripes along a common direction. The first layer output contains more details from the original sample, whereas, in the second layer output, blurring effect can be observed. This effect occurs due to the max-pooling operation (downsampling), as well as the second set of convolution processes, which extracts the most important information from the first convolutional layer outputs for the classification task.

Figure 5.8 shows an original algae sample, whereas Figures 5.9 and 5.10 present the first and second convolutional layer outputs for this sample, respectively. Blurring effect can also be observed in the second layer for the algae sample. Differently from the case of ICE, however, the first layer extracts edges along different directions.

Figure 5.13 illustrates an original flange sample. Figures 5.14 and 5.15 indicate the first and second convolutional layer outputs for this sample, respectively. In this case, the second layer reinforces the screw edges, which characterize this class. Downsampling effect can be specially observed in some of the second layer output images.

Figures 5.19 and 5.20 show, respectively, the first and second convolutional layer outputs for a CB sample, depicted in Figure 5.18. It is possible to observe, in the first layer output, the presence of edges along a preferred direction (vertical, in the example) and along a secondary direction (horizontal, in the example). These perpendicular lines characterize the frontal, lateral and bottom faces of the parallelepipeds. Besides, the difference of intensities of the regions delimited by the edges is preserved. In the second layer, blurring effect can be noticed due to downsampling, and a bigger distinction occurs between the filter outputs, so that some filters reinforce the edges and others preserve the differences of intensities between the regions separated by the lines.

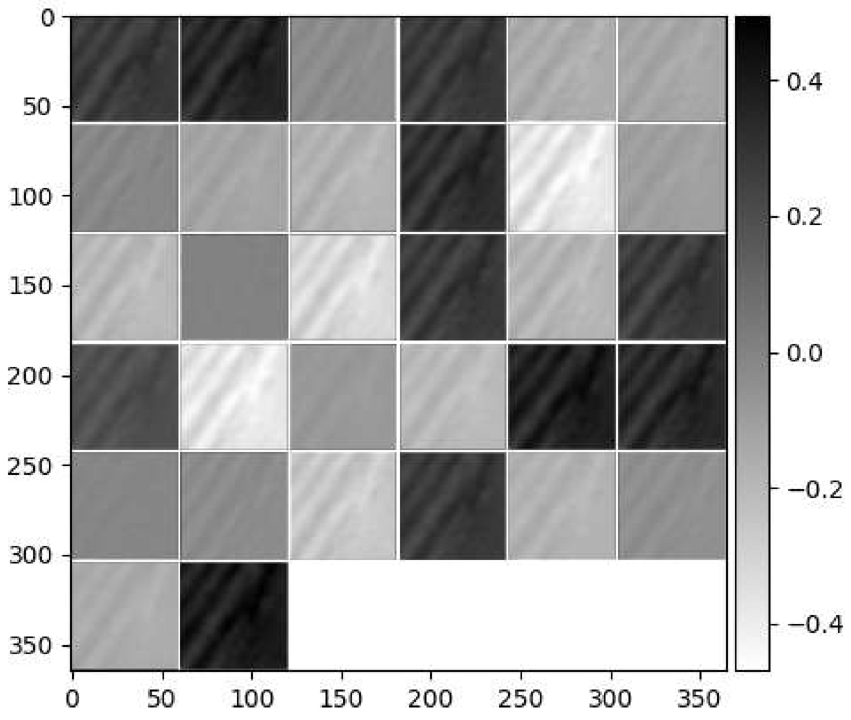

Figure 5.4: First convolutional layer outputs for ICE.

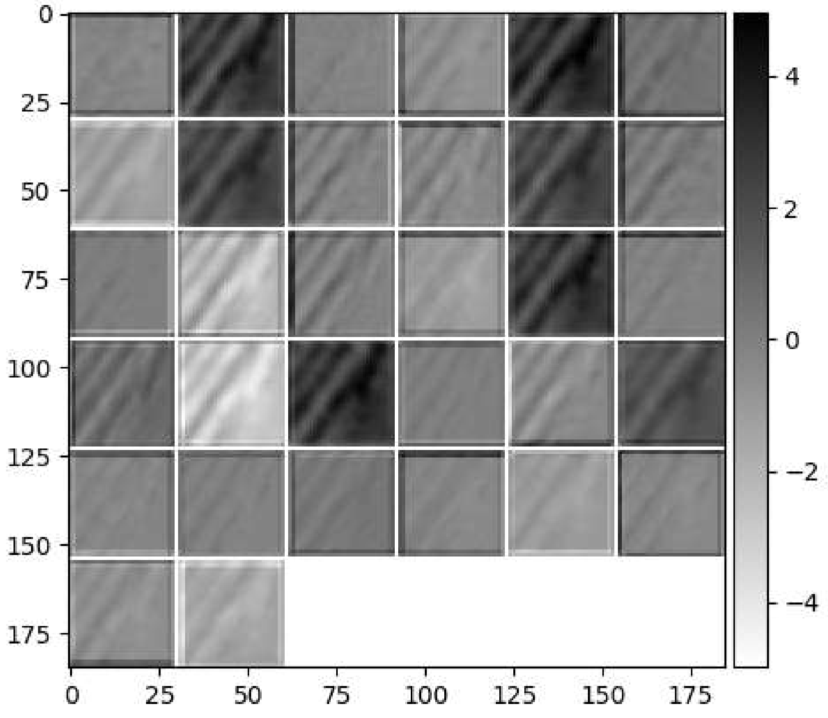

Figure 5.5: Second convolutional layer outputs for ICE.

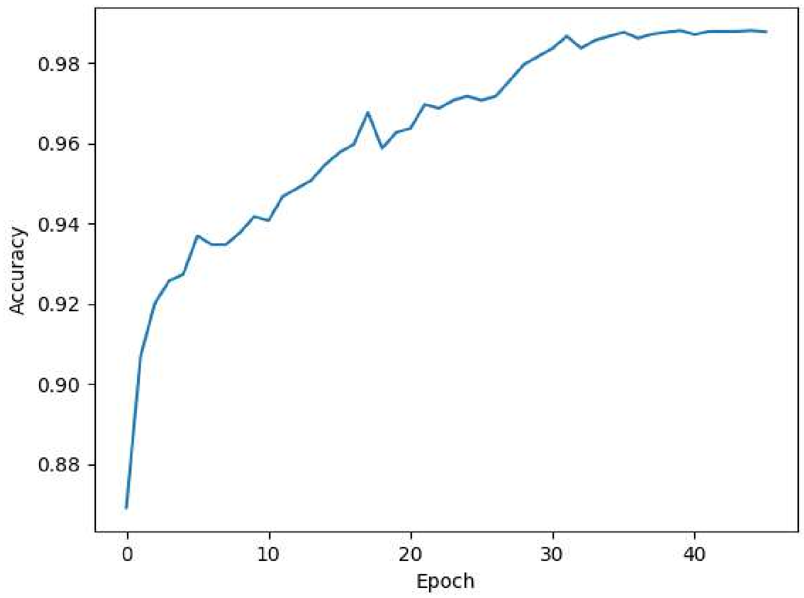

Figure 5.6: Accuracy for algae.

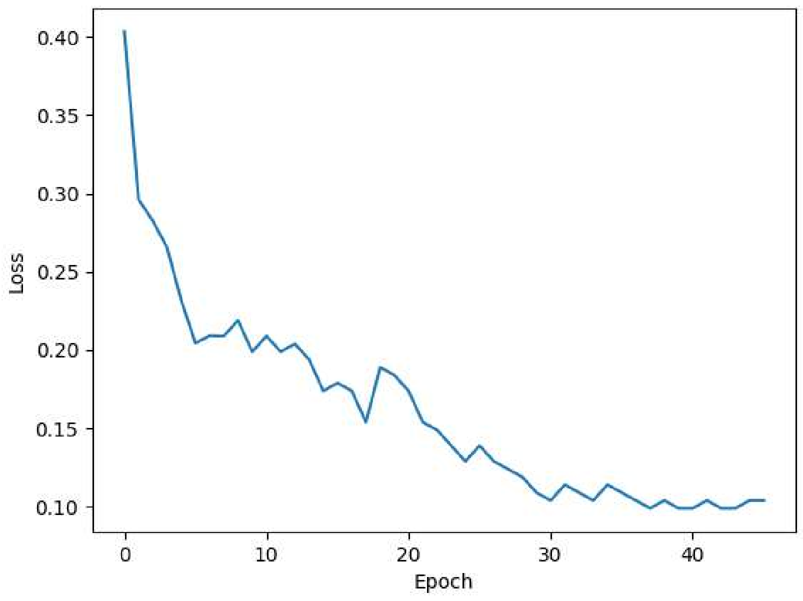

Figure 5.7: Loss function for algae.

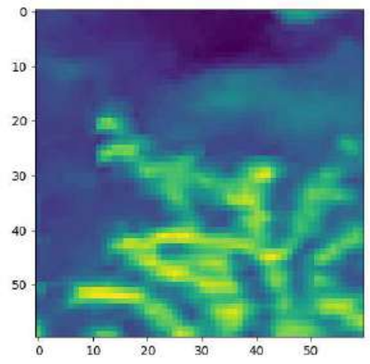

Figure 5.8: Original algae sample.

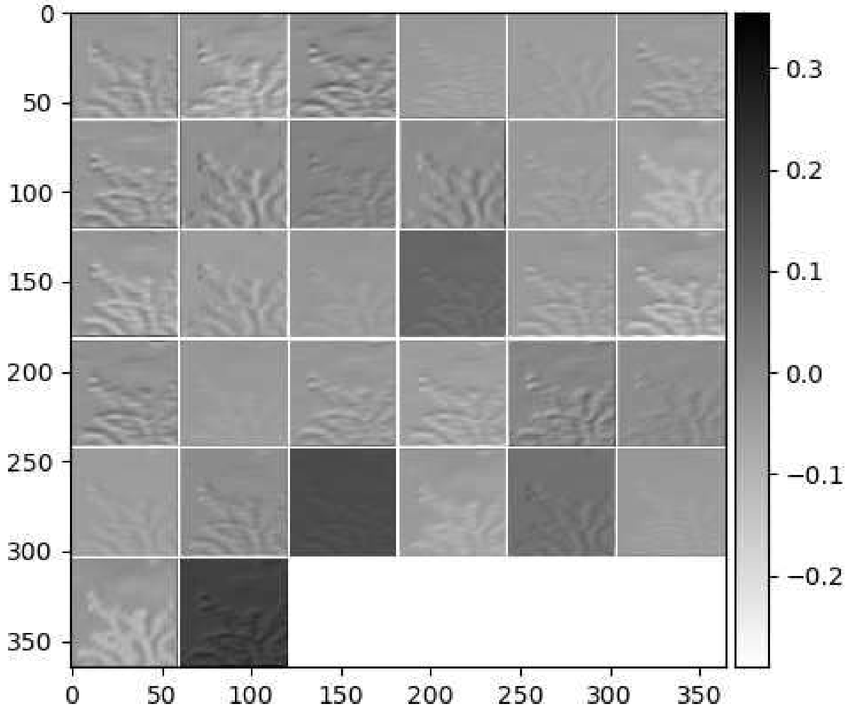

Figure 5.9: First convolutional layer outputs for algae.

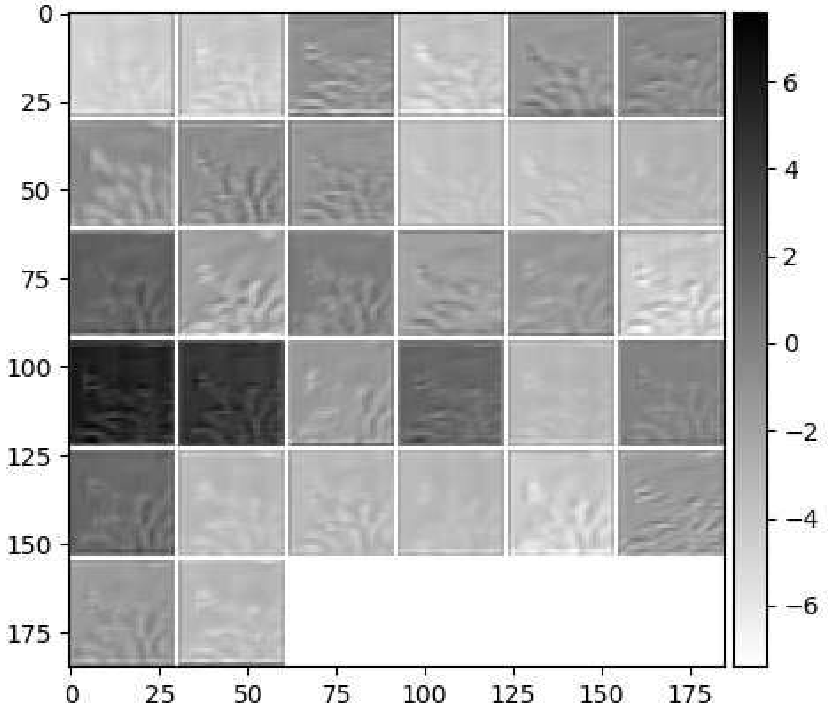

Figure 5.10: Second convolutional layer outputs for algae.

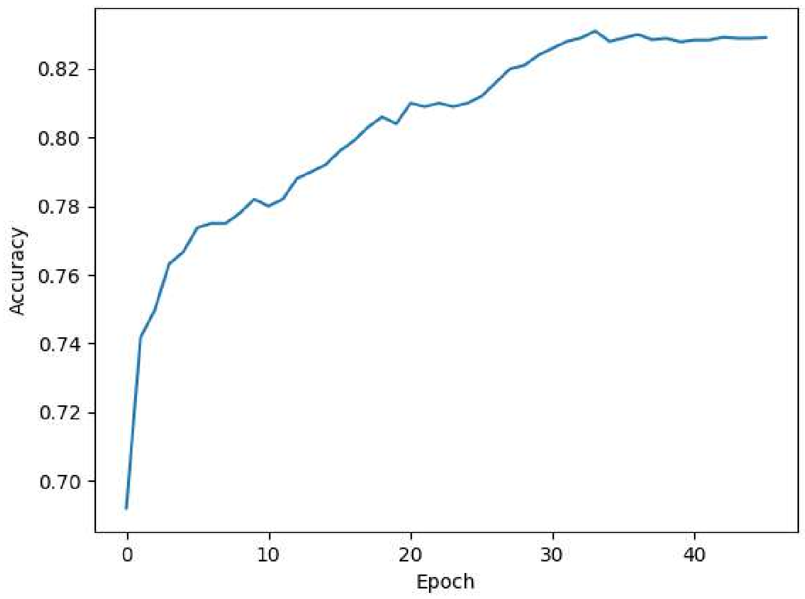

Figure 5.11: Accuracy for flange.

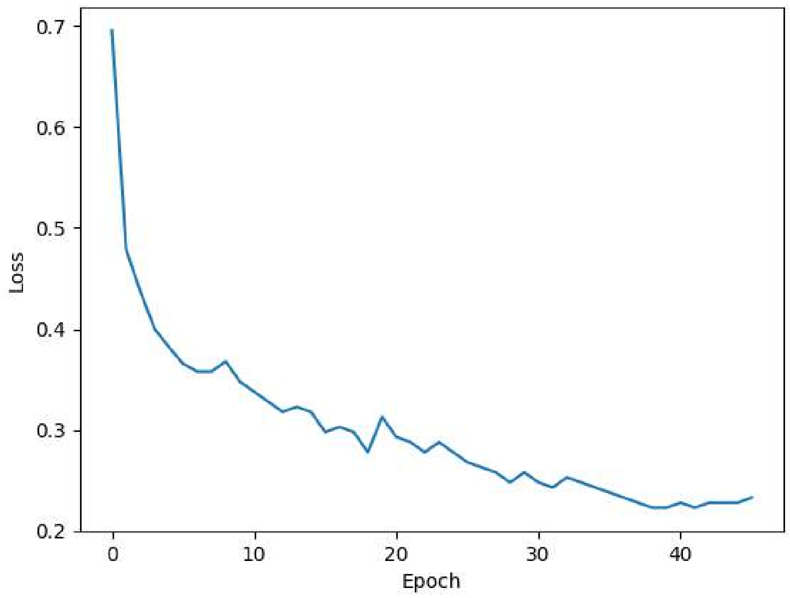

Figure 5.12: Loss function for flange.

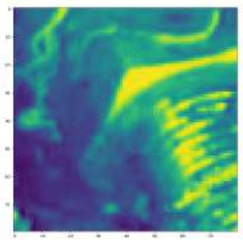

Figure 5.13: Original flange sample.

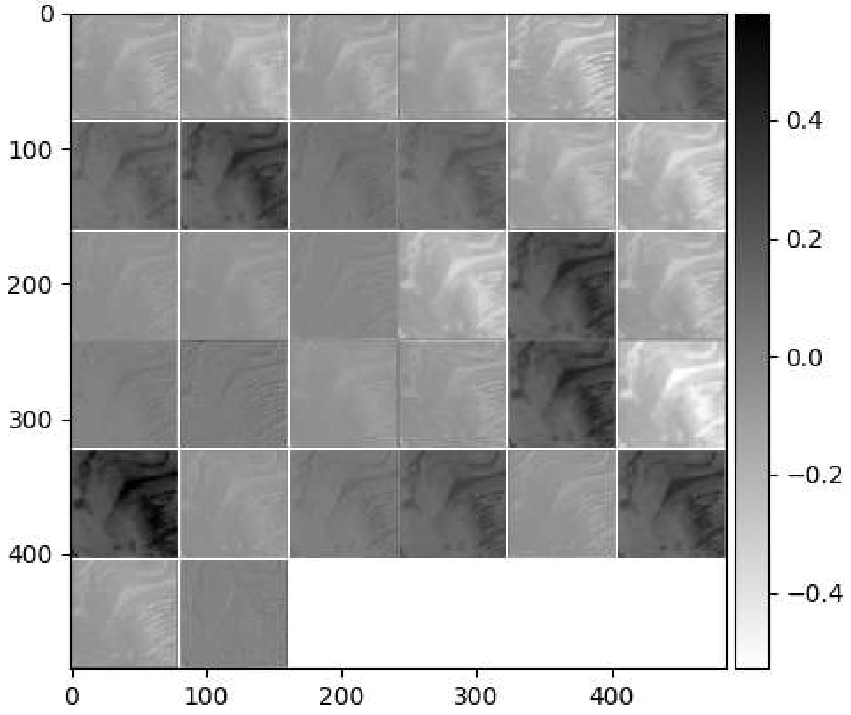

Figure 5.14: First convolutional layer outputs for flange.

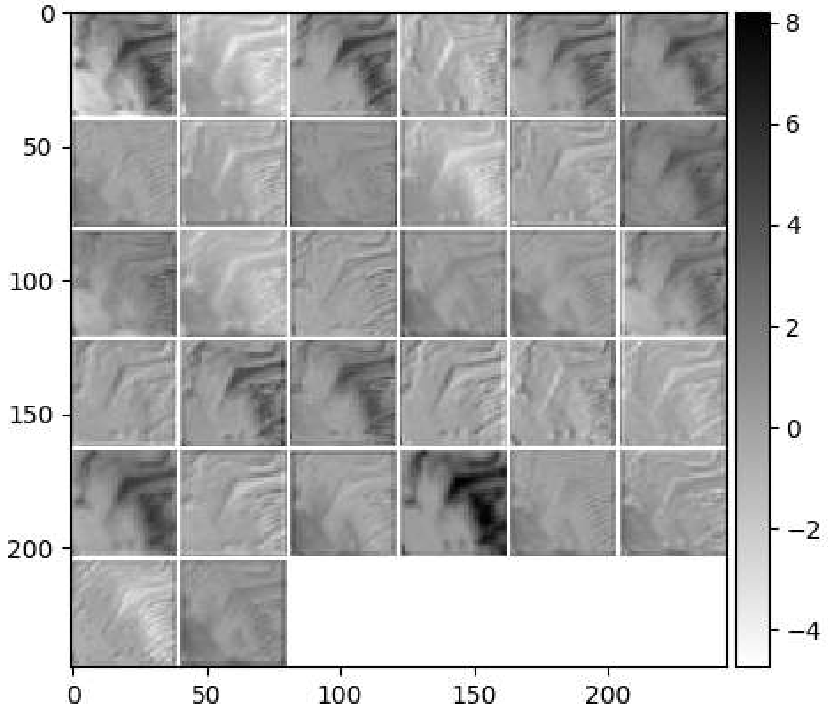

Figure 5.15: Second convolutional layer outputs for flange.

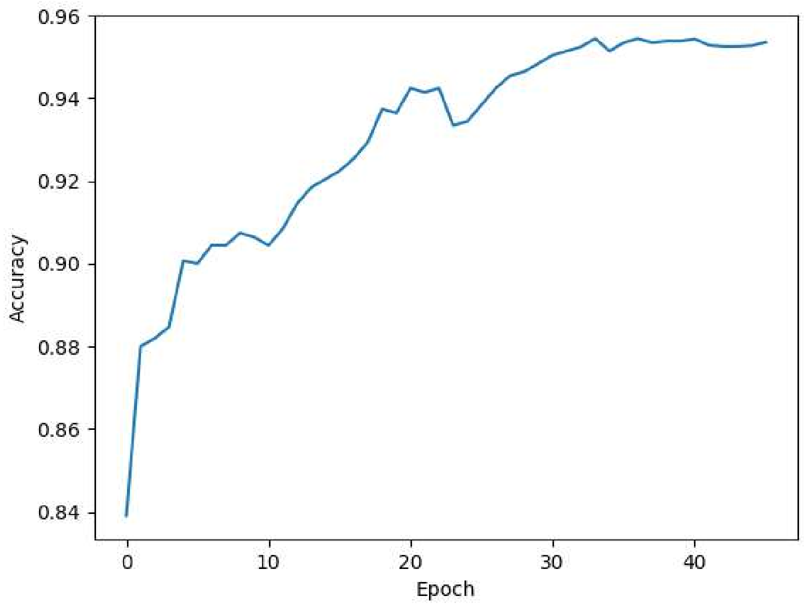

Figure 5.16: Accuracy for CB.

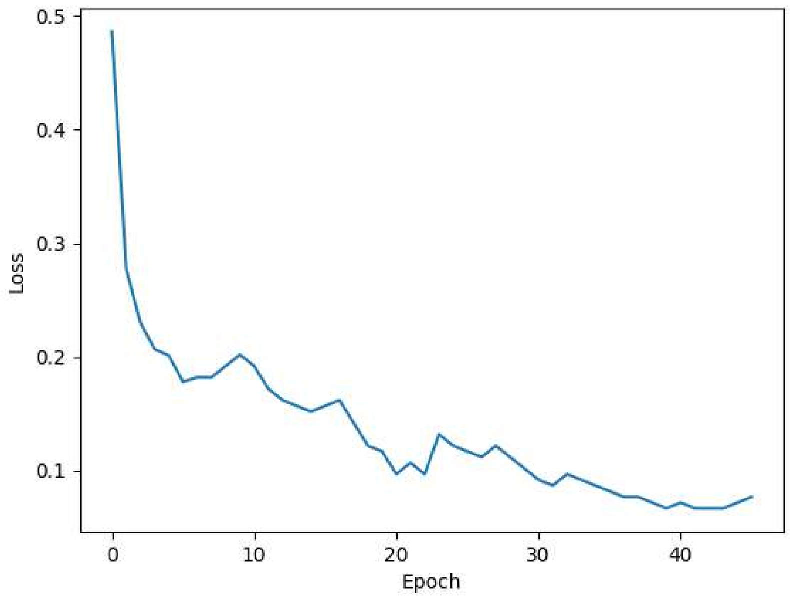

Figure 5.17: Loss function for CB.

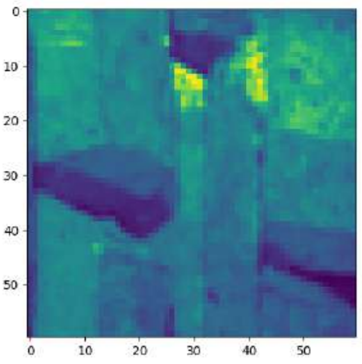

Figure 5.18: Original CB sample.

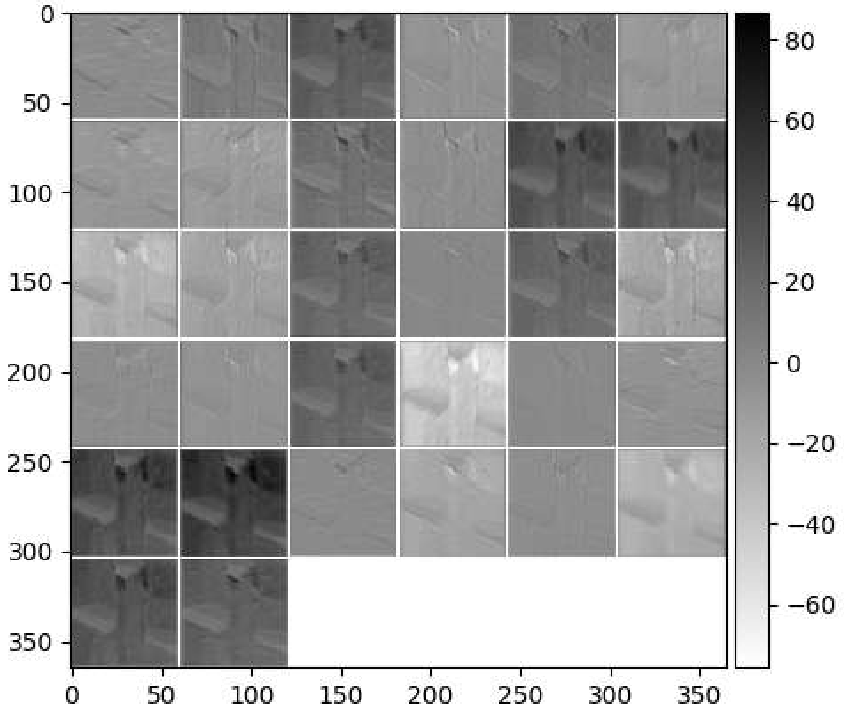

Figure 5.19: First convolutional layer outputs for CB.

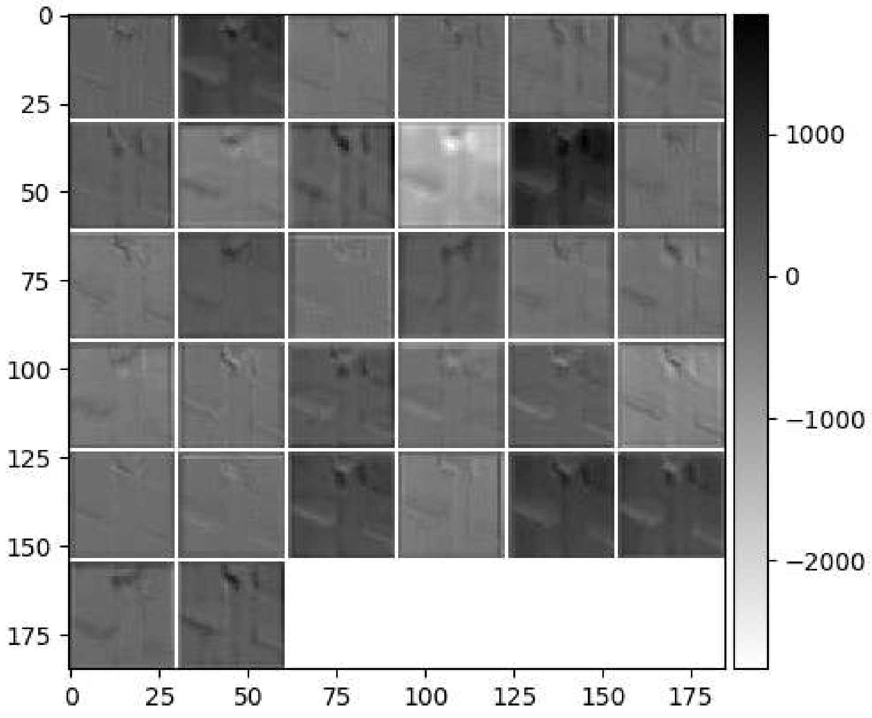

Figure 5.20: Second convolutional layer outputs for CB.

Chapter 6

Conclusions

The classification of underwater pipeline events has become essential for pipeline inspection, which has progressively turned into an automatic task. As opposed to classic neural networks techniques, which strongly depend on manually selected feature extraction methods, deep learning algorithms are able to extract their own features from original data, which significantly improves classification performance depending on the event class.

In this work, a CNN algorithm was implemented in order to classify four types of events: ICE, algae, flange and CB. With the purpose of preventing this complex model from learning specific characteristcs from the samples utilized for training, regularization techniques were utilized. Results from the CNN were compared to the ones obtained using a perceptron preceded by a wavelet-based feature extractor.

In Chapter 1, the machine learning topic was introduced. The objective of this thesis, to develop a machine learning architecture capable of achieving high classification accuracy for pipeline events without the need of manually selected feature extraction techniques, was described. Finally, the thesis structure was presented.

Chapter 2 presented basic neural network concepts, such as the perceptron, the MLP and the backpropagation learning algorithm. In order to provide complementary insight into neural network performance, activation functions, regularization techniques and feature extraction methods were described. Wavelet-based feature extraction method, utilized in this work, was detailed, as well as fundamental concepts related to this technique, such as wavelet transform and features based on first- and second-order statistics.

In Chapter 3 deep neural networks were described. CNN, a deep learning technique, specially utilized in image processing, which was implemented in this work, was specified. Regularization techniques vastly utilized in order to prevent deep neural networks from overfitting were presented. Optimization methods, such as SGD, AdaGrad, RMSProp and Adam, were described.

Chapter 4 detailed the experiments conducted and the database utilized in this work. The dataset extraction performed in this work was explained, as well as the composition of the training, validation and test sets. The four event classes considered were detailed, and the utilized CNN and MLP architectures were specified.

Chapter 5 presented the experimental results. Accuracy and loss curves obtained by the CNN and by the perceptron for the four event classes were exhibited and compared. In order to assist the understanding of the CNN behavior, visualizations of intermediate layer outputs were provided and analyzed.

The CNN was shown to classify underwater pipeline events better in comparison with the MLP based on wavelet features. Without the need of manually selected feature extraction, the CNN obtained a higher classification accuracy on all four event classes that were considered in this work, achieving 93.2% on average, whereas the perceptron accuracy reached 91.2% on average.

During this work, two papers were published in international conference proceedings [28], [29].

In this work, samples extracted for the four classes of events were independent, and treated as such. In order to further assist the detection of pipeline events based on video samples, in the future, temporal sequences of images can be used for training, validating and testing the model. To explore temporal sequencing of the image samples, recursive neural network models, such as the long short-term memory (LSTM), can be applied.

Bibliography

[1] ANTICH, J., ORTIZ, A. *Underwater cable tracking by visual feedback.* Proceedings of Pattern Recognition and Image Analysis, Lecture Notes in Computer Science, 2003.

[2] JACOBY, M., KARIMANZIRA, D. *Underwater pipeline and cable inspection using autonomous underwater vehicles.* MTS/IEEE Oceans, Bergen, 2013.

[3] SAHARIA, A., KEDIA, Y. *A Review paper on Artificial Neural Networks.* SSRG International Journal of Electronics and Communication Engineering, 2016.

[4] ROSENBLATT, F. *The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain.* Psychological Review, 1958.

[5] HAYKIN, S. *Neural Networks: A Comprehensive Foundation.* 2nd ed. , Prentice-Hall, 1999.

[6] LECUN, Y. *A Theoretical Framework for Back-Propagation.* Proceedings of the 1988 Connectionist Models Summer School, 1988.

[7] NAIR, V., HINTON, G. *Rectified Linear Units Improve Restricted Boltzmann Machines.* 2010.

[8] XU, B., WANG, N., CHEN, T., et al. *Empirical Evaluation of Rectified Activations in Convolutional Network.* 2015.

[9] HE, K., ZHANG, X., REN, S., et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.* 2015.

[10] CAWLEY, G. C., TALBOT, N. L. C. *On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation.* Journal of Machine Learning Research, 2010.

[11] BACH, F., JENATTON, R., MAIRAL, J., et al. *Optimization with Sparsity-Inducing Penalties.* 2011.

[12] NG, A. Y. *Feature Selection, L1 vs. L2 Regularization, and Rotational Invariance.* ICML, 2004.

[13] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., et al. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting.* Journal of Machine Learning Research, 2014.

[14] IOFFE, S., SZEGEDY, C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* 2015.

[15] VETTERLI, M., KOVACEVIC, J. *Wavelets and Subband Coding.* Pentice-Hall, Englewood Cliffs, New Jersey, 1995.

[16] MITRA, S. K. *Digital Signal Processing: A Computer-Based Approach.* 4 ed. McGraw-Hill, 2011.

[17] VAIDYANATHAN, P. P. *Multirate Systems and Filter Banks.* Prentice-Hall, Englewood Cliffs, New Jersey, 1993.

[18] STRANG, G., NGUYEN, T. *Wavelets and Filter Banks.* Wellesley-Cambridge, 1998.

[19] GOODFELLOW, I., BENGIO, Y., COURVILLE, A. *Deep Learning.* 2016.

[20] KHERADPISHEH, S. R., GHODRATI, M., GANJTABESH, M., et al. *Deep Networks Can Resemble Human Feed-forward Vision in Invariant Object Recognition.* Scientific Reports 6, Nature, 2016.

[21] LECUN, Y., BENGIO, Y. *Convolutional Networks for Images, Speech, and Time Series.* The handbook of brain theory and neural networks, 1995.

[22] RUDER, S. *An Overview of Gradient Descent Optimization Algorithms.* 2016.

[23] BOTTOU, L. *Large-Scale Machine Learning with Stochastic Gradient Descent.* Proceedings of COMPSTAT 2010, Springer, 2010.

[24] DUCHI, J., HAZAN, E., SINGER, Y. *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.* Journal of Machine Learning Research 12, 2011.

[25] DAUPHIN, Y., VRIES, H., CHUNG, J., et al. *RMSProp and Equilibrated Adaptive Learning Rates for Non-Convex Optimization.* 2016.

[26] KINGMA, D. P., BA, J. *Adam: A Method for Stochastic Optimization.* 3rd International Conference for Learning Representations, San Diego, 2015.

[27] MEDINA, E., PETRAGLIA, M. R., GOMES, J. G. R. C. *Neural-network based algorithm for algae detection in automatic inspection of underwater pipelines.* 15th International Conference on Artificial Intelligence, 2016.

[28] PETRAGLIA, F. R., GOMES, J. G. R. C. *Classification of Underwater Pipeline Events Using Deep Convolutional Neural Networks.* Proceedings of the 2017 International Conference on Acoustics, Speech and Signal Processing (ICASSP 2017),, New Orleans, Mar. 2017.

[29] PETRAGLIA, F. R., RUIZ, R. E. C., GOMES, J. G. R. C., et al. *Pipeline Tracking and Event Classification for an Automatic Inspection Vision System.* Proceedings of the 2017 IEEE International Symposium on Circuits and Systems (ISCAS 2017), Baltimore, Jun. 2017.