Internship Report: Automatic Defect Recognition over X-ray tire images using deep learning

Identification de manière automatisée des défauts sur les images radiographiques des pneus par apprentissage approfondi

BIRCK LOPES Liasse
Intern at *CyXplus*
Student at Third Year Automatic Control, Systems and Intelligent Systems

Grenoble
Ense3 - 2020

Acknowledgments

I would like to thank first Alexandre FOUCHARD, who trusted me with this internship and was an excellent mentor in the initials months. Then, I would like to thank Martin SAMOUILLER for keeping up the good work as a mentor and for always giving me support even when the conditions were not the best. I would also like to thank the whole CyXplus team, especially the IT design office, the R&D team and Agnès DECROUX, for their welcome and support.

In addition, I would like to thank my friends that made France my new home. We went through joys and difficulties always supporting each other. They are considered my family.

Abstract

The current CyXplus's software, CyXPERT, is able to automatically detect, localize and classify defect for in-line X-ray tire inspection. However, it depends on a configuration that is the time-consuming and user-variant. This project proposes an automatic defect recognition on X-ray manufactured tire inspection by deep learning algorithm, a robust method with a smaller margin of error. It also proposes processing and augmentation of the database, which are the prerequisites to carry out the learning of a neural network. A computer vision approaches called classification was chosen to be developed. It presented good results with high accuracy, however it has a significant number of false detection and in addition the defect localization is done by large zone areas. Due to the complicated nature of deep learning algorithms there is still a significant amount of work to be done before applying this algorithm in the industry. But generally, the project had promising results and a good evolution.

Keywords : Deep Learning; Image Processing; Non-Destructive Testing.

Résumé

Le logiciel actuel de CyXplus, CyXPERT, est capable de détecter, localiser et classer automatiquement les défauts pour l'inspection des pneus par rayons X en ligne de production. Cependant le taux de détection du logiciel est dépendant de la configuration faite par l'utilisateur. La configuration des traitements automatiques est également chronophage et demande des compétences poussées en pneumatique et logiciel. Ce projet propose une reconnaissance automatique des défauts lors de l'inspection radiographique des pneus par deep learning, une méthode robuste avec une marge d'erreur plus faible. Il propose également le traitement et l'augmentation de la base de données, qui est la base pour la phase d'apprentissage d'un réseau de neurones. Un type d'approche de vision par ordinateur a été choisi pour être développés, la classification. Elle a présenté de bons résultats avec une grande précision, mais elle comporte un nombre important de fausses détections et, de plus, la localisation des défauts n'est pas précise. En raison de la nature complexe des algorithmes d'apprentissage profond, il reste encore beaucoup de travail à faire avant d'appliquer cet algorithme dans un logiciel industriel. De manière générale, les résultats ont été prometteurs et le projet a eu une très bonne évolution.

Mots-clés : Deep Learning ; Traitement d'Images ; Essais non Destructifs.

Contents

List of Figures

List of Tables

List of Equations

Glossary

CNN :	Convolution Neural Network.
API :	Application Programming Interface
GPU :	Graphics Processing Unit
NDT :	Non-Destructive Testing
LR :	Learning Rate
Grad-CAM :	Gradient-weighted Class Activation Mapping
Feature Map :	Mapping of a certain kind of feature found in an image/sample

Introduction

Acknowledged to be a major player in the non-destructive testing (NDT) industry, CyX-plus supplies the most efficient inspection equipment and software to demanding manufacturing industries. As a leading supplier of final finish equipment for the Tire Industry, it has 90% of its income connected to tire X-ray inspection hardware and software.

The current software, CyXPERT, is able to automatically detect, localize and classify defects for in-line X-ray tire inspection. However, its use has some limitations. Its base configurations changes according to the tire reference as well as the type of defect that is desired to detect. The client is the one in charge of the parameters adjustment, requiring a trained technician and a significant amount of time. Therefore, there is no guarantee consistency on the results and there is still some false detection.

Provided an automatic defect recognition by deep learning, it would be no longer needed to spend man-hours on the time-consuming and user-variant task that is parameters configuration. The suggested approach would also replace the current image processing, which takes a long time to perform. The results are expected to be robust and have less errors.

This project has as main objective the development of deep learning algorithm able automatically to detect, classify and localize defects on X-ray manufactured tire inspection. Within constraints of cycle time in an in-line production. As secondary objective is the database processing and augmentation, which is the foundation of the deep learning algorithms.

First, a context is given. The company is presented, its products developed and the fields of activity. Also, the context of study, an important background to understand about tires, their defects and image acquisition. Another point is the work environment, the tools used throughout are announced. Then, some computer vision approaches are discussed and, finally, the scientific approach is presented.

Next, the theory behind the deep learning algorithms is presented. Then, the database are discussed along with how they were processed. Followed by the development of the classification deep learning approach and its results are discussed. Finally, a global conclusion is explored over the project and a personal one over the internship is presented.

1 Context

1.1 Company

CyXplus is a simplified joint stock company (in french *société anonyme par actions simplifiée - SAS*) that manufactures non-destructive testing (NDT) and measurement equipment and software to the manufacturing industries.

The 35 employees are divided into two places. At Montbonnnot-Saint-Martin there are the Research & Innovation in NDT technologies team and the X-ray micro-tomography equipment. At Les Pennes Mirabeau is the operational headquarters, engineering and development departments, X-ray and computed tomography equipment, workshop and the teams of after-sales, purchasing, sales, administration/accountancy, human resources.

Figure 1: CyXplus logo.

CyXplus is part of TechnipFMC Company, a global leader in oil and gas projects, technologies, systems, and service. With more than 37 000 employees in 48 countries and a turnover of US$12b in 2018.

The company's markets are the tire industry, aerospace and defense, oil and gas industry and pharmaceutical industry. It's activity is mainly focused on on inspection of passenger, truck, bus and off the road tires. The company is present on the five continents and works with the most prestigious tire brands, like Michelin, Goodyear, Apollo and others.

1.1.1 Activities & Technologies

CyXplus develops and sells complete non-destructive testing systems, from the machine to the associated software. The company offers solutions based on 2D digital X-ray, computed tomography, laser and industrial vision. In its products ranges there are tire X-ray inspection, in-house X-ray/CT inspection facilities and software.

CyXplus builds X-ray inspection machines for its customers and in particular for the tire industry. Different types of machines are offered by the company to check tires of several sizes, from 15" to 72". Furthermore it builds custom applications such pipe inspection and X-ray for pharmaceutical equipment.

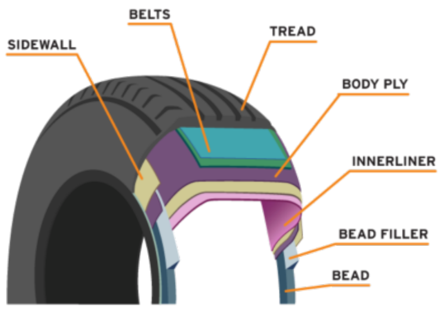
The company also builds in-house X-ray/CT inspection machines, with focus on the aerospace industry. They can X-ray or CT small to medium size objects. CyXplus' role is the development, troubleshooting, expertise and final inspection before assembly or delivery.

The software for the machines is also developed. These include software for image acquisition, visualization and automatic defect detection. Some software allow the control of the machine, from the management of the X-ray chain (source, detector) to the robotics of the machine (rotation of the axes for example). For its tomographs, CyXplus also develops a 3D reconstruction and visualization with measurement tools.

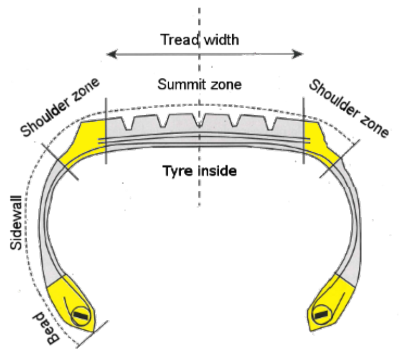
1.2 Context of the study

1.2.1 Tires

The materials under analyses are tires. They can be for motorcycle, passenger, light truck, truck, bus and off-the-road (OTR) tires, with radial tires between 15" and 72". A tire has a lot of structures, layers and materials.



(a) Tire Structure.



(b) Tire Zones.

Figure 2: Tire.

On Figure 2a and according to [10], the tire structure consists on: Inner liner: Layer of synthetic rubber, where the air pressure is retained; Belts: they are made of steel, providing stability to the tread, which affects wear, handling, and traction. Beads: They clamp firmly against the tire's rim to ensure an airtight fit and keep the tire properly seated on the rim; Sidewall: It covers the body plies on the side of the tire and provides abrasion, scuff, and weathering resistance. Important details about the tire are written on the sidewall, such as tire size and speed rating; The sidewall is composed of rubber compounds that. Body ply: it serve as the structural foundation of the tire, providing the strength to contain the tire air pressure and carry the load. Most tires have one or two body plies. Tread: Provides traction and turning grip for the tire and is designed to resist wear, abrasion and heat. It is is constructed from different rubber compounds.

These structures and layers can be divided into four mirrored zones, described in the Figure 2b.

1.2.2 Acquisition

In X-ray imaging a beam of high energy electrons, accelerated by a strong potential difference, crashes into a target made of heavy metal and interact with it, producing x-rays. After filtering, only the high energy rays pass through a sample toward the sensor. X-ray can penetrate liquids, gas and solids. The penetration is based on the intensity, quality and wavelength of the X-ray beams. The stronger the X-ray beam the more it can penetrate the material.

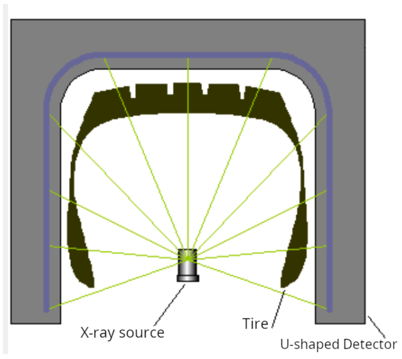
The flow transmitted through each sub-element of homogeneous material of thickness x obeys the Beer-Lambert law:

$$I = I_0 \cdot exp(-\mu(\rho, \lambda, Z) \cdot x) \tag{1}$$

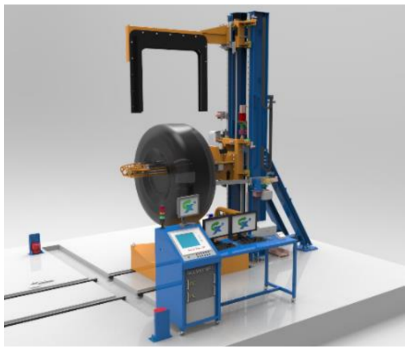
where $\mu(\rho, \lambda, Z)$ is the linear absorption coefficient of the material under consideration, depending on its density ρ, its atomic number Z and the energy of the radiation under consideration (its wavelength λ)

This means the waves of X-ray can penetrate more easily through materials of light atoms, such as air, plastic and rubber. Metals absorb the X-rays, not letting them arrive at the sensor. Meaning the higher the density of the material the darker it will be imaged on the result image.

In the this panoramic NDT the source is positioned in between the beads. The sensor is wide but thin, to capture only a section at a time, and is positioned around the tire's outside part. The image formation is done by moving the object between source and detector.



(a) Schematics.



(b) Machine.

Figure 3: X-ray acquisition.

1.2.3 Defects

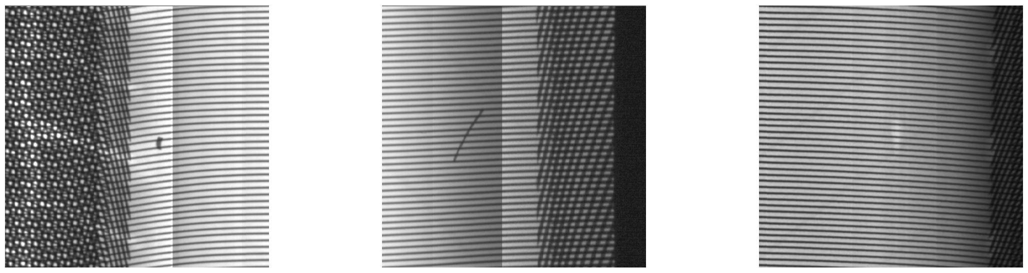
There are three types of defects that this project focused on: bubbles, free wires and foreign bodies. Bubbles happen when the tire layers are not glued together properly and air bubbles are trapped in between. This causes structural damage, degrading durability, the bubble can expand with continuous use and in some rare, but dangerous, cases can cause tire rupture.

Foreign bodies are metal pieces of any type or shape. They accidentally end up on the tire layers during fabrication process. They can severely degrade durability and increasing the risk of tread separation. Free wires is a special case of foreign bodies. They happen when a single metal wire from the many wired layers gets lose.

1.3 Work Environment

All this project was done using Python as the programming language, through Jupyter Notebook [7] environment. Which is an open-source web application for documents creation with interactive code, equations, visualizations and narrative text. It can be used for data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and more.

The main libraries used, the hardware and the platform of multi computing environment will be described in the following.



(a) Foreing Body. (b) Free Wire. (c) Bubble.

Figure 4: Defect Types.

1.3.1 Keras

Keras [2] is one of the leading high-level neural networks APIs. Written in Python, it supports multiple back-end neural network computation engines. It was “designed for human beings, not machines”, new modules, such as new classes and functions, are simple to add and all standalone modules that can be combined to create new models.

It allows easy and fast prototyping through user friendliness, modularity, and extensibility. It supports convolutional networks, recurrent networks and the combinations of both. It runs seamlessly on CPU and GPU.

Keras does not do its own low-level operations, like tensor products and convolutions. It relies on a back-end engine for that. Which, in this project, is TensorFlow,

1.3.2 TensorFlow

TensorFlow is an open source software library, created by the Google Brain team, for numerical computation and large-scale machine learning using data-flow graphs. It has a comprehensive, flexible system of tools, libraries and community resources. It runs on nearly GPUs and CPUs, including mobile and embedded platforms.

The most important benefit that TensorFlow provides for machine learning development is abstraction. Instead of dealing with all the small details of implementing algorithms, the developer can focus on the overall logic of the application, leaving TensorFlow to take care of them. It uses Python to provide a front-end API for building applications, while it executes them in high-performance C++.

1.3.3 Docker

Docker is an open source tool designed to make it easier for the user to create, deploy, and run applications by using containers. Containers allow to package up an application with all of its essentials parts, like libraries and other dependencies, and deploy it as one package. Then, the application will run on any other Linux machine regardless of different machines’ settings used for writing and testing the code.

Rather than creating a whole virtual operating system, Docker uses the same Linux kernel as the system that they’re running on and only requires applications be shipped with things not already running on the host computer. Giving a significant performance boost and reducing the size of the application.

It is reproducible, it will run exactly the same on any device capable of running a Docker container. The specifications of a container are stored in a Dockerfile, any Docker image built from it will function identically. Dependencies or settings within a container are isolated, they will not affect any configurations on the user's computer, or on any other containers. However, Docker has performance costs, it does not run processes quite as fast as those run on a native operating system.

1.3.4 GPU

Graphics processing unit (GPU)-accelerated techniques, as the ability to conduct complex simulations faster, allows developers to explore more design choices, optimize their designs for performance and cost, and consequently brings groundbreaking products to market faster.

This project's main computer has a *NVIDIA Quadro P6000* GPU and the second computer has three GeForce GTX 1080 Ti GPUs. Their specifications can be found in Table 1. They are able to handle the most complex datasets and computationally intensive workloads. They maximize productivity and accelerate data access time, ensuring the highest level of compatibility, functionality and reliability.

	Quadro P6000	GeForce GTX 1080 Ti
GPU Memory	24 GB GDDR5X	11 GB GDDR5X
Memory Interface	384-bit	352-bit
Memory Bandwidth	432 GB/s	484 GB/s
NVIDIA CUDA Cores	3840	3584
Max Power Consumption	250 W	250 W
Thermal Solution	Active	Active
Form Factor	4.4"H x 10.5" L, Dual Slot, Full Height	4.376"H x 10.5" L, Dual Slot
Display Connectors	4x DP 1.4 + DVI-D	DP 1.43, HDMI 2.0b
Max Resolution	2560 x 1600 @ 60Hz	7680 x 4320 @ 60Hz

Table 1: NVIDIA Quadro P6000 and GeForce GTX 1080 Ti specifications.

1.4 Computer Vision Approaches

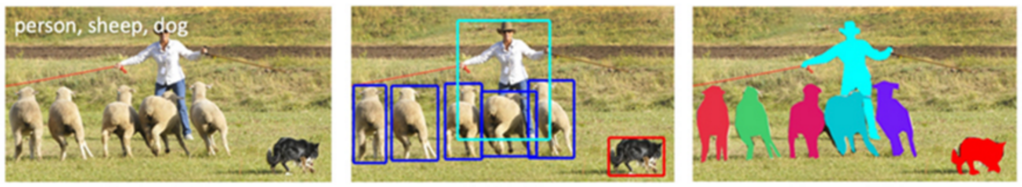
Actions such as recognizing an animal, describing a view, differentiating among visible objects is on humans daily basis. Computer vision aims to imitate this functionality of human eye and brain components that is responsible for your sense of sight.

It took decades of research to discover and impart the ability to detect an object to a computer with reasonable accuracy. In the past decade the field of computer vision has witnessed continual advancements. Today, deep CNNs form the crux of most sophisticated fancy computer vision application, such as self-driving cars, gesture recognition, automatic number plate recognition, auto-tagging of friends in Facebook pictures.

There are many types of implementation computer vision which uses CNNs, this project is going to focus only on three: Classification (Figure 5a), Detection (Figure 5b) and Segmentation (Figure 5c).

1.4.1 Classification

Image classification refers to a process of classify/label an image according to its visual content. An image can be classified into a number of categories. Classification tries to answer the question, *"What is in this image?"*. For example, an image classification



(a) Classification. (b) Detection. (c) Segmentation.

Figure 5: Some types of Computer Vision

algorithm may be designed to tell if an image contains a human figure or not, or if it contains a defect A or a defect B or even no defect.

Two main exemples are VGG [17] and ResNet [4], as they are commonly referred. VGG is a concise approach and highly efficient in extracting features from images. ResNet is a powerful backbone model that is used very frequently in many computer vision tasks. It uses skip connection to add the output from an earlier layer to a later layer. This allows the network to go ”deeper” than the others without damaging performance, a traditional ResNet counts with 152 layers. Its architecture is displayed in Appendix A - Figure 37.

1.4.2 Detection

Object detection is a computer vision technique that deals with distinguishing between objects in an image. It is more specific than classification in what it identifies in addition it uses bounding boxes to show where each object is in an image. This technique is useful to identify particular objects in a scene, such as the cars parked on a street, detect faces and detect defects. The most common and powerful architectures are YOLO [12] and R-CNN [3].

R-CNN divide the image in regions to localize and classify the objects. It does not analyze at the complete image,only parts which have high probabilities of containing an object. In YOLO a single convolutional network predicts the bounding boxes and their class probabilities, being a much faster way to detect objects and it is able to be applied in real time. Its architecture is displayed in Appendix A - Figure 38.

1.4.3 Segmentation

Segmentation is a type of labeling where each pixel in an image is labeled with given concepts, creating a mask of objects. This technique gives a far more granular understanding of the objects in the image. It tries provide the exact outline of the object within an image, as opposed to classification models, where the model identifies what is in an image, and Detection models, which places a bounding box around specific objects.

It is often used in medical area for cellular separation, tumor identification and others. A very effective structure is U-Net [13], as the name suggest it its u-shaped structure allows the features to be extracted while maintaining their locations. Its architecture is displayed in Appendix A - Figure 39.

1.5 Scientific approach

In order to face this project's problem, which is the development of a deep learning algorithm able to detect, classify and localize defects on manufactured tires, within constraints of time cycle in an inline production, the first step was the study of deep learning algorithms, presented on chapter 2.

Then, it was to collect the data from the clients. In deep learning application, the data is the base of everything, it must be labeled and manipulated according to the project's needs. On chapter 3 is described the type of data, its characteristics and how it was processed and handled.

Finally, the classification approach was developed. The network structure was improved in order to be a good model, making good prediction on both database and, in consequence, on the real life application. Each step's results were analyzed regarding the network performance. They were the basis to decide which changes were to be made on the network structure and which modifications was necessary to make the data generation closer to the reality. After that, the network's hyper parameters were tuned in order to get a better performance. Every step is described in chapter 4.

Due the confidentiality the work is presented more generally and some information were not presented. The clients' names were hidden, being referred as Client X and Client Y. The tire references from Client X were also hidden, referred as R1, R2, R3, R4 and R5.

2 Theoretical Background on Deep Learning

Deep Learning is a machine learning technique inspired by the structure of the brain. Machine learning is when a machine is able to learn through algorithms, with programs that alter themselves.

2.1 Neural Networks

Artificial Neural Networks (ANN) are collections of a large number of simple processing units called neurons, each of which makes simple decisions, calculations as shown in Figure 6. Together, the neurons can provide accurate answers to complex problems, such as natural language processing, computer vision, and artificial intelligence.

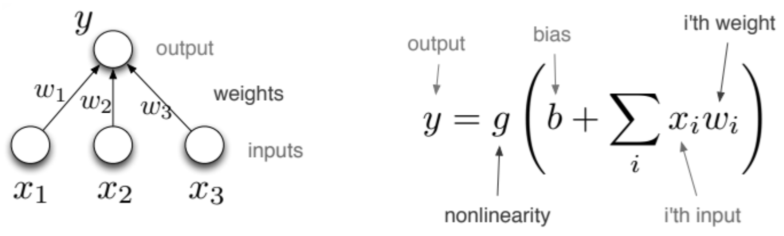


Figure 6: Neuron's structure

A neural network can be "shallow", meaning it has an input layer of neurons, only one hidden layer that processes the inputs, and an output layer that provides the final output of the model. A Deep Neural Network (DNN) commonly has between 2-8 additional layers of neurons. Neural networks increase in accuracy with the number of hidden layers.

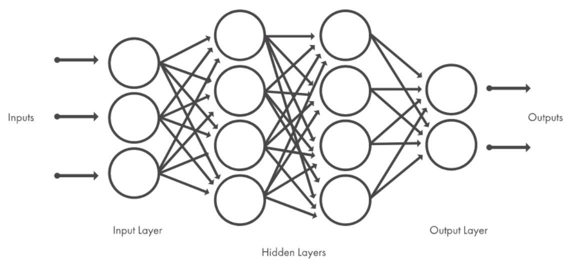


Figure 7: Simple Deep Neural Network scheme

Deep learning models are trained by using large sets of labeled data and neural network architectures that learn features directly from the data without the need for manual interference.

Neural networks learn by example. They receive the input data and make a guess. Then, they compare their guess with the output's true value. According with the error, or loss (section 2.3.3), they updates their own internal weights through the use of an optimizer (section 2.3.3). Then they make another guess and the loop continues. To stop the loop the developer can analyze the models metrics (section 2.3.2) and decide that the model is a good representation of the data and there is no overfitting or underfitting (section 2.1.1). Or the decision to stop the training can be done after the network passed through the whole dataset, also called epoch, a certain number of times.

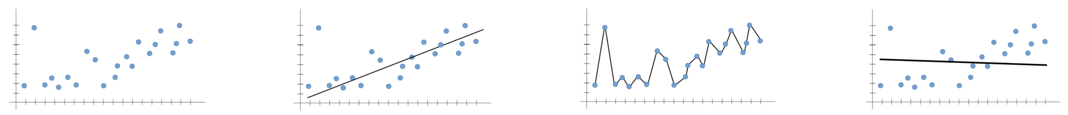
2.1.1 Overfitting vs Underfitting

In statistics, a fit refers to how well a target function is approximated. In supervised machine learning the algorithms seek to approximate the unknown underlying model for the output variables given the input variables. Its goal is to generalize well from the training data to any data from the problem domain, edging overfitting.

Generalization is when the concepts learned by a machine learning model respond well applied to specific examples not seen by the model during the learning. When it does not occur there are two biggest causes for such poor performance of machine learning algorithms: overfitting and underfitting.

Overfitting is when there is a good performance on the training data, but poor generalization to other data. The model learns the noise or random fluctuations of the training data, as seen in Figure 8c. However, these concepts do not apply to new data and negatively impact the model's ability to generalize.

Underfitting is when there is poor performance on the training data and poor generalization to other data, Figure 8d. It is not a suitable model, is often not discussed as it is easy to detect given a good performance metric.



(a) Data. (b) Adaquate model. (c) Overfitting. (d) Underfitting.

Figure 8: Model Fitting

2.1.2 Activation Functions

As seen in Figure 6, the neuron's output is a linear transformation on the inputs using the weights and biases followed by a non-linearity. This non-linearity in the network is introduced by an activation function. This kind of strategy gives more power to the network, which will be able to learn the complex patterns from the data. Also it have a major effect on the neural network's ability to converge and the convergence speed.

There is a significant amount of activation functions, this project is going to use only two. Which are described in the following.

- Rectified Linear Unit (ReLU)

  The main advantage of the ReLU function is that it does not activate all the neurons at the same time. For the negative input values, the result is zero (2), that means the neuron does not get activated. Since only a certain number of neurons are activated, the ReLU function is computationally efficient when compared to other activation functions. This allows the network to converge very quickly.

$$f(x) = max(0, x) \tag{2}$$

- Softmax

  This function is able to handle multiple classes, which others can only one. It normalizes the outputs for each class between 0 and 1, and divides by their sum, giving the probability of the input value being in a specific class (3). Softmax is used only for the output layer, for neural networks that need to classify inputs into multiple mutually exclusive categories.

$$\sigma(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{N} e^{x_k}}, for j = 1, ..., N \tag{3}$$

2.1.3 Dropout

Overfitting is a serious problem in deep neural networks, being slow to use makes difficult to deal with it by combining the predictions of many different large neural nets at test time. Dropout [19] is a technique for solving this problem. It to randomly drop nodes and their connections from the neural network during training, given a chosen rate. Preventing the nodes from co-adapting too much. This significantly reduces overfitting and gives major improvements over other regularization methods.

2.1.4 Batch Normalization

When training Deep Neural Networks the distribution of each layer's inputs changes as the parameters of the previous layers change. Which slows down the training because it requires lower learning rates and careful parameter initialization. This problem is solved by normalizing the layer's inputs.

Batch Normalization [5] performs the normalization for each training batch. By applying a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. Acting as a regularizer, in some cases eliminating the need for Dropout. It also allows the use of much higher learning rates and to be less careful about initialization.

2.2 Convolution Neural Networks

The Convolution Neural Networks (CNN) generally involves a sequence of one or more convolutional layers followed by fully connected layer (section 2.1), as depicted in Figure 9. The transition between this two layers is usually a flatten function, which transforms a three dimension element into an one dimension.

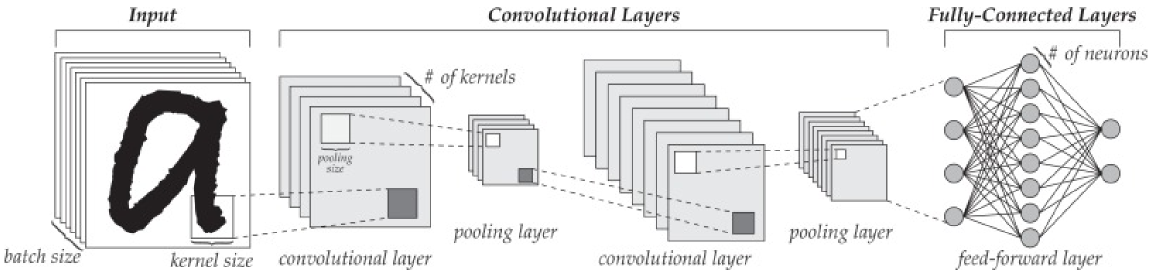


Figure 9: Typical structure of a sequential convolutional neural network [1]

These convolutional layers are responsible for performing feature extraction, they learn which features from the data are relevant. The first convolutional layer input are the data and the output will be its feature maps, which will then be introduced as the input for the subsequent layer.

In the convolutional layer's core there are three main components, convolution, pooling and activation functions, which the two firsts are going to be described in the following. The last was already described in 2.1.2.

2.2.1 Convolution

A convolution operation is a linear application of a smaller filter to a larger input that results in an output feature map. This operation consists in the sum of the element wise product, which a single number. The systematic left-to-right and top-to-bottom application of the filter to the input results in a two-dimensional feature map, Figure 10. One filter creates one corresponding feature map. The the output's number of channels of one convolutional layer is defined by the number of parallel filters applied to the input.

It is important to point out that a filter must have the same number of channels as the input.

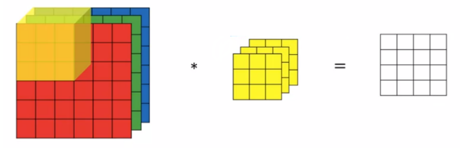
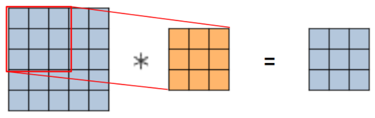


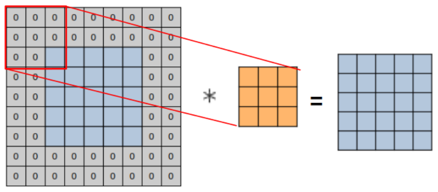
Figure 10: Convolution of a volume.

- **Padding :** means to fill, in CNNs is used zero padding. There are two most commons type, *Same* and *Valid*. In *Valid* there is no padding. It can be noticed in Figure 11a that the spatial dimensions decrease, when convolution is applied using this type. As the convolutional layers keep being applied the size of the volume will continuously decrease faster than desired.

  In *Same* the output has the same length as the original input, then the input image is padded in other to have this outcome, Figure 11b. It is very useful when dealing with temporal data, where it shouldn't be violated the temporal order.



(a) Valid Padding.



(b) Same Padding.

Figure 11: Types of Padding

- **Stride :** controls how the filter shifts around the input volume, how many cells the filter is going to skip. This value is true for the width and the height, however it is not for the channels. An example is depicted in Figure 12, where a convolution is being applied with a stride of 2.

The equation of the convolution output's size in the following, with n and f as the source's and filter's two dimensional size, respectively, n_c the number of channels, n_f the number of filters, p the padding and s the stride.

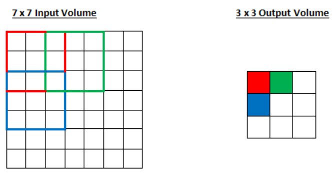


Figure 12: Convolution $7 \times 7 * 3 \times 3$ with a stride of 2.

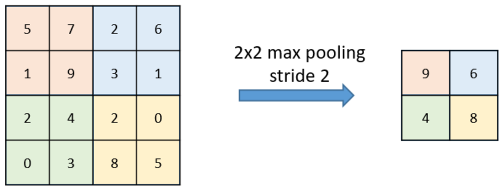


Figure 13: Visual exemple of max pooling

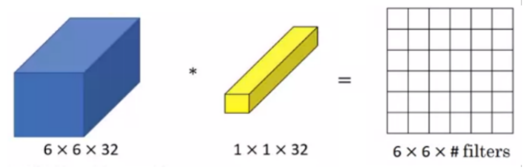


Figure 14: Convolution 1×1

$$n \times n \times n_c * (f \times f \times n_c) \times n_f = \left\lfloor \frac{n + 2p - f + 1}{s} \right\rfloor \times \left\lfloor \frac{n + 2p - f + 1}{s} \right\rfloor \times n_f \tag{4}$$

2.2.2 Pooling

There are two types of pooling typically used in CNN's, max pooling and average pooling. The second calculates the average value of the cells that the filter goes through, however it lost its popularity and it is not commonly used anymore. The first, max pooling, keeps the maximal value between the cells, a visual representation is available on Figure 13.

It's important to point out that the computation of max pooling is performed on each of the channels independently. Furthermore, there are no parameters to learn and there's no padding. The formula of this operation output's size in the following.

$$n \times n \times n_c * (f \times f \times n_c) \times n_f = \left\lfloor \frac{n - f + 1}{s} \right\rfloor \times \left\lfloor \frac{n - f + 1}{s} \right\rfloor \times n_f \tag{5}$$

2.2.3 Special Case: 2D Convolution 1×1

In this special case the same rules of convolution are applied, as is verified in Figure 14. However, some interesting properties come to light. A 1×1 convolution can be used to down sample the number of feature maps, it is equivalent to cross-channel parametric pooling layer and it can be also used to replace a fully connected layer.

In the first cases, as the number of filters used in convolutional layers often increases with the depth of the network, it results in an increase in the number of output feature maps. A large number of feature maps in a CNN can cause computational problems, as convolutional operation must be performed down through all the channels of the input.

A single 1×1 filter can summarize the input feature maps. With multiple 1×1 filters it is possible to tune the number of summaries of the input feature maps to be created, increased or decreased the depth of the feature maps as needed. This is often called channel-wise pooling, as opposed to traditional feature-wise pooling on each channel.

A 1×1 filter will have only one parameter for each channel, resulting in a single output value. This structure acts like a single neuron with an input from the same position across each of the feature maps. Then, adding more filters in parallel would be like adding nodes to a fully connected layer.

In the superior part of Figure 15 is represented in the two dimensional space a CNN, in which its fully connected part is not made by nodes, but by 1×1 convolutions. It is important to highlight that the previous flatten function has become a convolution that has the same shape of its input, with its output being a $1\times1\times n_{filters}$.

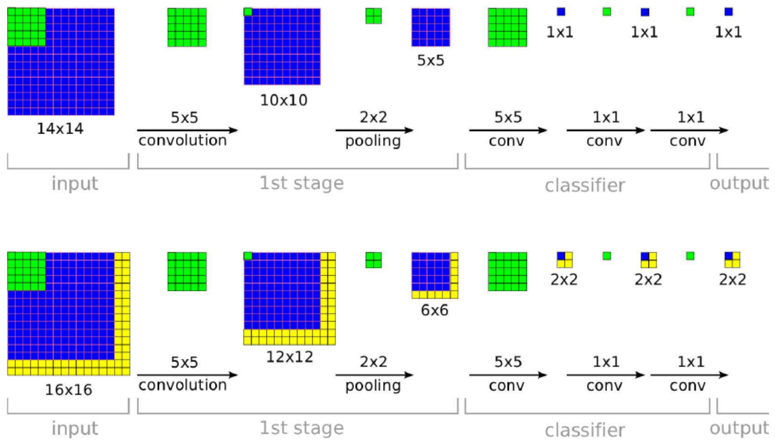


Figure 15: Convolutional Implementation of Sliding Windows [16]

The main advantage of this property is that now the CNN can have a larger image as input, without the need of preprocessing it or passing it through the network repetitive times. It is called convolutional implementation of sliding windows, it is depicted in the below part of Figure 15. When applying this network to larger images at test time, is simply applying each convolution over the extent of the full image. Extending the each layer's output to cover the new image size, producing at the end a map of output predictions, with one spatial location for each field of view (or "window") of input.

2.3 Classification Networks

Classification networks label an image according to its visual content. An image can be classified into a number of categories. An image classification algorithm may be designed to tell if an image contains a human figure or not, or if it contains a defect A or a defect B or even no defect. A labeled database can be seen in Figure 16

2.3.1 Architecture

There are several classification structures, some of them do not even use convolution operations, such as Support Vector Machine (SVM), Decision Tree, however they are not part of the scope of this project. However, the most popular architecture used for image classification is CNNs, among those architectures there are AlexNet, VGG, ResNet, Inception and others.

Figure 16: Examples from CIFAR-10 database.

2.3.2 Model Metrics

In order to analyze if the model's performance is good, if the network is giving the right predictions it can be used some quantitative metrics. The first one is the loss value, discussed in 2.3.3, this value should decrease as network is trained. However it can be misleading, masking the model's overfit. Other criteria can give a more realistic overview and they are described below.

a) **Confusion Matrix**

Confusion matrix is a visual way to analyze the obtained results. It charts the predicted versus the actual classification. Its representation for two classes is depicted in Table 2, for multi-class models the representation increases in size accordingly, however the over all idea remains. From these results the other quantitative metrics are calculated.

		Predicted	
		Negative	Positive
Actual Value	Negative	*True Negative*	*False Positive*
	Positive	*False Negative*	*True Positive*

Table 2: Confusion Matrix

b) **Accuracy**

To calculate the accuracy it is used the sum of the right diagonal of the confusion matrix - which are the values that were predicted correctly - divided by the total amount of samples. As it is shown in Equation 6

$$Accuracy = \frac{TruePositive + TrueNegatives}{Totalsamples} \tag{6}$$

Model accuracy may not be a reliable metric of performance, if the validation data set is unbalance the results can be misleading. If a class represents 80% of the samples and the model predicts all samples as being this one, the accuracy will be 80%.

Furthermore, in certain situations the costs of having a false negative is very high. Using this project as example, having a defect piece produced in the industry predicted as a good one can cause malfunction and even accidents. Then, accuracy is not only metrics to be analyzed.

c) **Precision**

Precision measures the rate over all predictions said to be true (right column of Table 2), which ones model got it right.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{7}$$

This metrics is relevant to be analyzed when the costs of false positive is high.

d) **Recall**

Recall calculates the rate of true positives over all the actual positives (last row Table 2 that passed trough the model.

$$Recall = \frac{TruePositive}{TruePositive + FalseNegatives} \tag{8}$$

Recall shall be the model metric to analyzed when there is a high cost associated with false negative.

e) **F1 score**

When a balance between Precision and Recall is needed, the best to use is F1 Score .

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{9}$$

That is, a high F1 score means there are few false positives and few false negatives, so it is correctly identifying real threats and it is not disturbed by false alarms.

2.3.3 Loss and Optimization

a) **Loss Function**

A neural network is trained based on a set of input and outputs. It learns its weights by verifying if its output guess is similar to the true output. The difference between the guess and true value, called error, is the value that the neural network wants to minimize.

The loss function, also called the cost function, reduces all the various aspects, positives and negatives, of a complex system down to a single scalar value, which indicates how well those weights accomplished in the intended task. This allows the comparison and ranking of the solutions.

These functions often measure the squared or absolute error between the network's output and the desired output. Classification models have loss functions designed specifically to minimize the distance between the network's distribution over class labels and the distribution from the dataset. Find the correct loss function can be a challenging problem as it must capture the properties of the problem and be motivated by concerns that are important to the project.

For regression models it can be used mean squared error loss, mean squared logarithmic error loss or mean absolute error loss. As for binary classification models it can be used binary cross-entropy, hinge loss or squared hinge loss. Then, for a multi-class classification model, as is the one of this project, there are sparse multi-class cross-entropy loss, Kullback Leibler divergence loss and multi-class cross-entropy loss, which is the loss function used in this project. Its formula is presented in equation 10, with N as the number of samples, $\hat{y}_i$ as the predicted output and y_i as the real output.

$$J = -\frac{1}{N}\left(\sum_{i=1}^{N} y_i \cdot log(\hat{y}_i)\right) \tag{10}$$

b) **Optimizer**

The weights of a neural network have a direct relationship with the error it produces. Since, the goal is to minimize the error, also known as loss function or objective function, it is needed to change the values of the weights. This can be done through the use of optimization algorithms and back propagation.

A neural network receive the signal of the input data, propagates through its parameters and then give an output. It calculates the error between the output and the true values, and then back propagates this information, in reverse through the network, so that it can alter the wrong-headed weights accordingly to the optimizer.

Some of the most common optimization algorithms are Gradient Descent, Adam, Adagrad, Adadelta, RMS Prop and Momentum. A visual example of Gradient Descent is depicted on Figure 17, where two paths are drawn, the one on the left leads to a local minima while the one on the right leads to the global optima. This project is going to focus on the second the Adam optimizer.

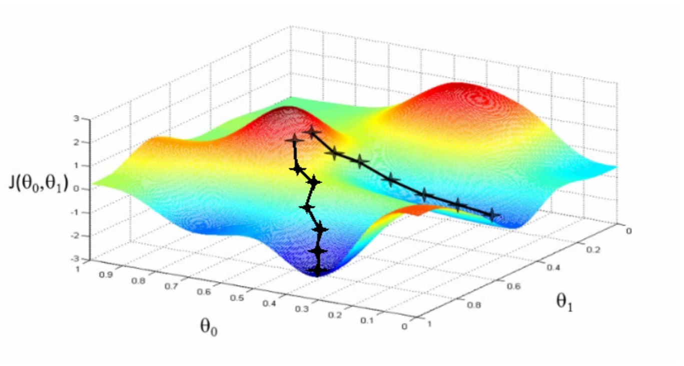


Figure 17: Gradient Descent example.

Adam [6] a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement, it combines the advantages of two other extensions of stochastic gradient descent, AdaGrad and RMSProp. It leverages the power of adaptive learning rates methods to find individual learning rates for each parameter, from estimates of first and second moments of the gradients. The name Adam is derived from adaptive moment estimation.

It is straightforward to implement, computationally efficient, doesn't require a lot of memory, well suited for large data and/or parameters. It is also invariant to diagonal rescale of the gradients, appropriate for non-stationary objectives and for problems with very noisy or sparse gradients. Its hyper-parameters usually require little tuning and are intuitive to interpret.

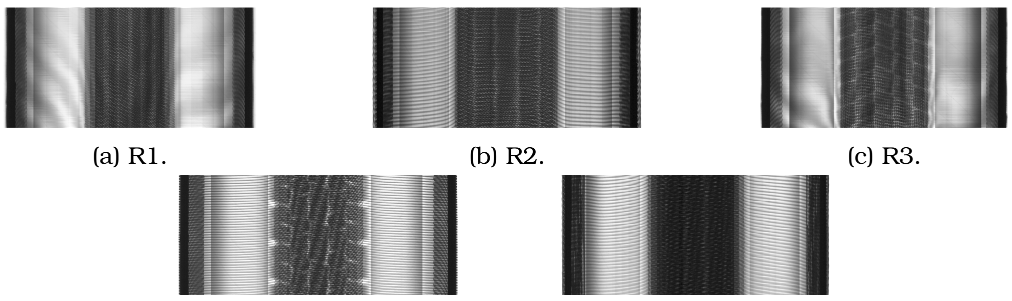
3 Database

The database is the foundation of this project. The project is designed to learn its characteristics and recognize them in unseen data. Poor sample data creates a system that is unable to performed its task properly. In this section is described the data and how it was manipulated to be applied in the work of chapters 4.

There are three types of defects to be identified : bubbles, free wires and foreign bodies. Each one appears different on the X-ray image. Bubbles show themselves whiter than the rest, since the air's absorption rate (equation 1) is low. Foreign bodies are signaled as dark shapes. Free wires, a special case of foreign body, are dark and very thin, its size can vary.

3.1 Client X database

The Client X database is formed by five tire truck references: R1, R2, R3, R4 and R5. The acquired images are about 3.072×17.000, the height can vary a little depending on the acquisition, but the width is fixed. An horizontal cut of an example image of each reference is depicted in Figure 18.


(a) R1. (b) R2. (c) R3.

(d) R4. (e) R5.

Figure 18: Horizontal cut from Client X database images.

The database is formed by 83 images, having approximately ten images without defect per reference and an average of six images with defects per reference. In which the defects are 90% foreign body, 10% free wire and 0% bubbles.

3.2 Client Y database

The Client Y database consists of acquired tire truck images of 3.455×8.593 pixels. There are in total 65 images representing the four classes, in which 34% of all images are bubble defect. 21% of the images have "cord spacing", which is a small gap between wires on the sidewall, and 14% have "shoulder stretch", which is a stretch on the shoulder zone's rubber, as the name suggests it. Those configurations are not defects and the network must be robust to them.

This type of tire differs from the ones of Client X on size and wire distance distance, both smaller. The summit zone is also narrower and the returning-layer zone is wider, resulting in a darker image.

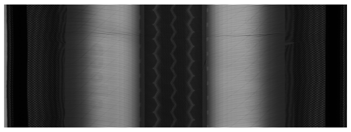


Figure 19: Horizontal cut from Client Y database images.

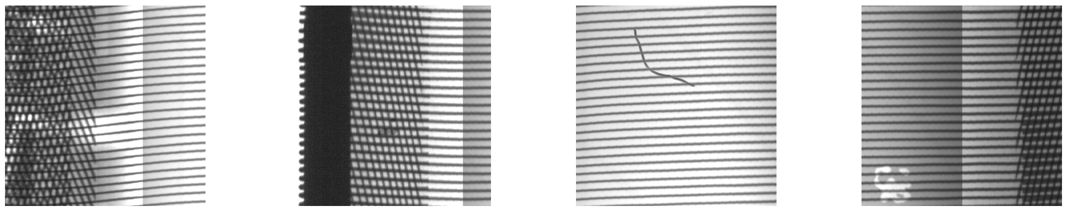
3.3 Artificial Generation of Defects

The database provided by Client X lacks representativity and quantity. A significant amount of diverse samples is necessary to train a classification and a detection neural network. Also, classification of big images takes longer to train and demands more memory space. Therefore, it is needed to generate a dataset, with the image size, representativity and quantity required to this project.

It was decided that the images containing the real defects would be kept to validate the network's performance. Then, thumbnails would be generated from the images without defects and they would be used to train the network. The thumbnail's size was defined as 400×400.

The thumbnails are chosen in a random manner from the original images. 1,300,000 samples were generated from the images without defect, for each type of defect (even for without defect). Some data augmentation techniques were used, such as brightness change. This simulates real life situations, where the density of the source or the object can change.

Machine learning techniques and signal processing operations were used to generate the defects on the thumbnails. They create masks which when applied on the images simulates a defect. In Figure 20 is depicted some results. This dataset is going to be referred as D0.



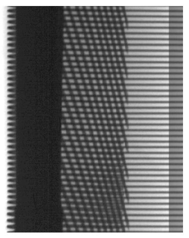
(a) No defects. (b) Foreing Body. (c) Free Wire. (d) Bubble.

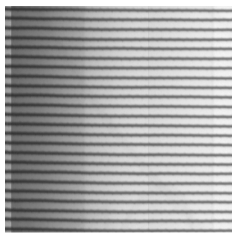
Figure 20: Articial Generated Thumbnails

This dataset has some generation problems which were proved to result in misleading noise on the samples. For example, Figure 21a is an sample from the class free wire. However, the defect was generated on the bead, making it impossible for a human or for the network to see. This condition, found in many images of free wire and stain defect, happens due to the randomization of the place selected to insert the defect. One way to adjust that is by defining zones were the defect can be placed.

Other problem, found in Figure 21b, is a sample from the class stain. The generated defect is so small, or maybe even nonexistent, that it cannot be seen by human or machine. This problem is also due to the randomization, this time on the size of the defect. To best avoid that it was increased the base size of the defect.

The last significant problem is saturation. As it can be seen in Figure 21c, the bubble on the image is of the same value as the background, both saturated. The generation of bubbles is done by brightening the original image in a specific zone, however, if the original image is already saturated there's not much to be done. It can be only soothed by avoid brightening too much the original image before placing the defect.


(a) Dark defects on the bead.


(b) Tiny defect.

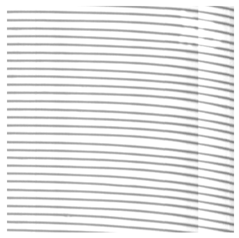
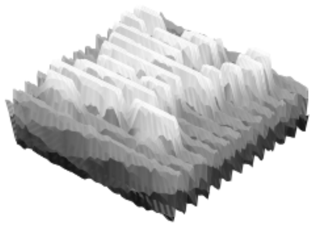

(c) Saturated image.

Figure 21: Problems on the dataset.

A new dataset was generated to mitigate those problems with four times the amount of samples and with zoom in and out augmentation to simulate a possible change in the source distance. This dataset is going to be referred as D1.

Using the same generation functions a dataset was created for Client Y, with 900,000 samples it is going to be referred as A0.

After further analyses other problems were found on the dataset. The bubble generation does not correspond with the reality, in Figure 22 there are the gray levels topology of the defects same size cut, it can be seen that the original one is more long than wide, also the transition between wire and defect is smoother. A new dataset was generated attenuating this problem, it was also reduced the amount of samples to 20,800. This adapted Client X's dataset is going to be referred as D2.


(a) Gray levels topology for the generated bubble.

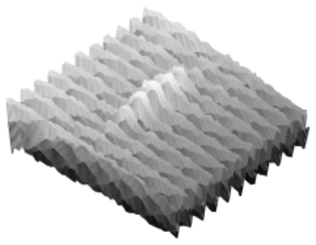

(b) Gray levels topology for the original bubble.

Figure 22: Other problems on the dataset.

3.4 Classification dataset

In a classification algorithm the input is a image, however, on the output it has a label/class. And as a supervised machine learning technique it must be fed the input with the corresponding output so it is able to learn to classify them. Taking into account those rules the dataset must be organized. A separation per class was made, as show below, for both clients' database.

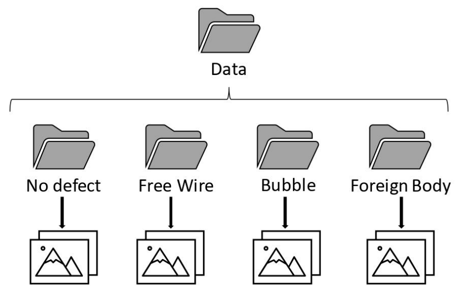


Figure 23: Classification dataset organization.

3.5 Dataset split

The database must go through a preprocessing phase before it is ready for the network's use. The first step is to split the generated data, randomly, in to three different sets:

- **Training set** : The actual dataset that is used to train the model. The model sees and learns from this data. It usually holds a significant amount of the database.
- **Validation set** : This dataset provides an unbiased evaluation of a model fit on the training dataset. It is used by the developers to fine-tune the model hyperparameters. The model sporadically sees this dataset but never uses it to learn the weights.
- **Test set** : This dataset provides an unbiased evaluation of a final model fit on the training dataset. It is only used once a model is completely trained. This test set can be only a random sample of the database or it can contain carefully sampled data that spans the various classes present in the real world.

In this project was used the split 80%-10%-10%, respectively.

4 Classification Workflow

The goal of this project is to develop a deep learning algorithm able to detect, classify and localize defects on manufactured tires. Within constraints of time cycle in an in-line production. The classification neural network method was chosen due the amount of research and material available, its adaptability to the many industry sectors and its performance.

In this section it is described how it was developed the network structure through the use of Client X dataset and how it was modified to make predictions over the original database images. Then, it is shown how the same structure can also be applied to Client Y's and keep the same level of performance. After that, the hyper parameters are tuned, aiming to improve the network's performance. Finally, a conclusion is done over the approach, the performance and the results.

4.1 Network Structure

At the start of the project a convolutional was provided by the company. In its structure, depicted in Figure 24, the convolutions are followed either by a batch normalization or a max pool layer. There are in total two ReLu activation, there are four blocs with 208 convolution 5×5 followed by a batch normalization in sequence and at the end a fully connected layer for classification.

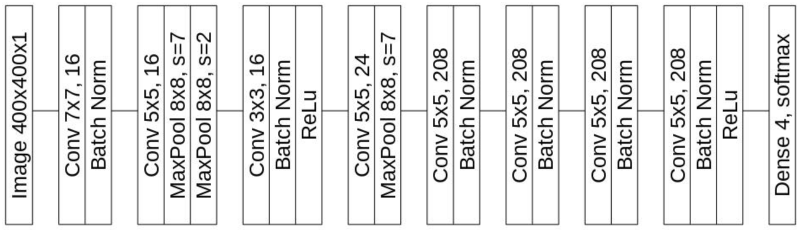


Figure 24: Network structure.

This network was first implemented in Matlab™for the Client X tires of reference R1, R2, R3, R4, R5 on the D0 dataset. The results of each reference separately and for all them together can be found in Appendix B - Table 6.

The work of this paper started by getting to know what was already done by translating it to Python code. This first adaptation was made using the equivalent Matlab functions in TensorFlow and Keras, there was no change on the network's structure or parameters (its architecture is depicted in more details on Appendix D - Figure 43). The results over the same references and database can be found in Appendix B - Table 7. It can be seen that in the training set there was not much of a change, however, in the validation set and test set there were some significant degradation. Comparing the results, the translation was validated.

This structure only allows as input images of the same size on which the network was trained, 400×400. In order to make a prediction over the original database, even on the acquired NDT images, it is needed to pass a observation window with the network input's shape over the entire image. Taking 200 pixels as the sliding window step and that it takes approximately 0.02 seconds for the network to make a prediction, it would take about 15 seconds to classify the entire image. Which is unfitted for an inline verification.

To improve the total prediction time it was made a convolution implementation of sliding window, as explained in Section 2.2.3. The fully connected layer was replaced by a convolution layer with the same quantity of inputs and outputs, also the network input height and width was not defined. With these modifications the network's ability to process income data changes, without changing its internal overall structure. This configuration does not require the preprocessing of the acquitted image and the network only has to make one prediction, taking about 2.4 seconds. However, it does not allow free choice of the step between observation windows, since it is define by the multiplication of the internal strides.

When this modification was added it was tested only for the reference R1, which had the best results so far, in order to save time. The results are on Appendix B - Table 8, they show that there was degradation on all datasets performances.

To improve the results, and to try to reverse all the degradation that occur, it began the process to optimize the network structure. Analyzing the work in [17][13][4] and in many more, some decisions were made. The first was that all the convolutions will have an activation function and a batch normalization. The second was over the max pool layers, they would not be in sequence, but evenly spread. The third, was that there were too many convolutions that were exactly equal with no nonlinearity between them, so they would be reduced in half.

The first decision left a dilemma, which was the best layer order. In the bibliography the results were contradictory, each application had its own best. Then a specif test was made, with the implementation of the other two decisions over the reference R1990, to define the layers order. The results in Appendix B - Table 9 show that the best order is: convolution followed by batch normalization, then by the activation function. The max pool layer will enter directly after the convolution.

The new structure can be found in Figure 25 and in more details in Appendix D - Figure 44.

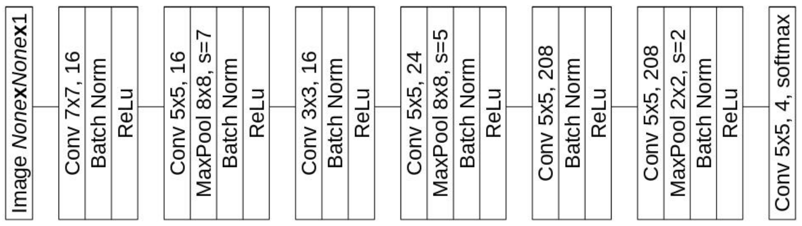


Figure 25: Network structure V1.

This network was trained for the five references separately, the results (Appendix B - Table 10) showed a significant improvement over the initial ones in Matlab, that were the best until now.

All the analyses so far were done using the artificial generated images, however, to truly validate the model it is needed to try it over the original database. The prediction over these images were done by a fixed 400×400 window of observation and a fixed step of 70. It was chosen that a defect prediction over the volume of predictions (output) would be considered valid if its certitude is greater than 90%. Then, the resultant matrix made of 0's, 1's, 2's and 3's, corresponding to each class, is compared with the ground truth through the use of *sklearn* [11] functions.

From now, and on the following predictions, the reference used was R1 due its good response on the model.The resultant confusion matrix over the evaluation of the original database can be seen in Figure 27a, more results detail can be seen in Appendix B - Table 11. The results show an elevated number of false positives (upper diagonal), even for a median accuracy of 95.99%.

In order to reduce this problem, the network structure was analyzed. On the max pool layers there were some inconsistencies, the very first division (due to the stride) did not give an entire value, requiring it to be rounded. Then, a new architecture was made, it was change the size and stride of some of the max pool layers, as it can be seen in Figure 26 and in better details in Appendix D - Figure 45, leaving a resulting observation window step of 80.

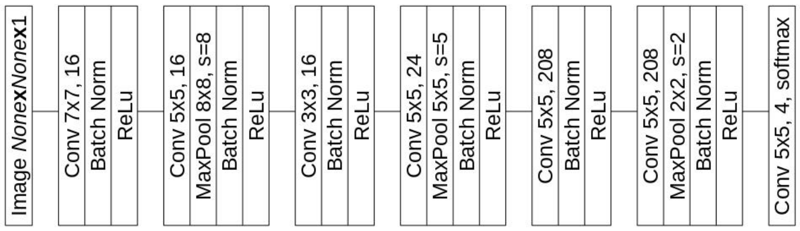
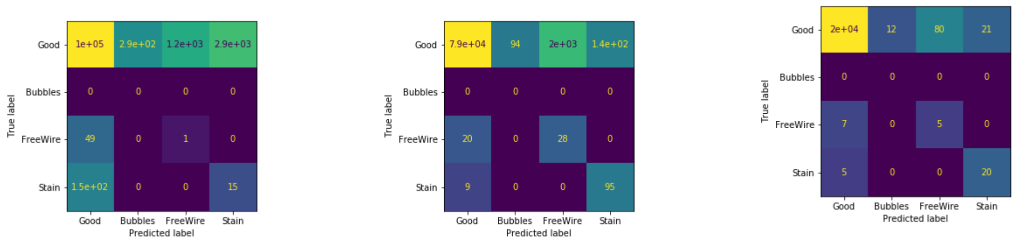


Figure 26: Network structure V2.

The results over the datasets show a slight degradation on validation and test dataset (Appendix B - Table 12) , however, when compared to the results on the original database in Figure 27b (and on Appendix B - Table 13) the number of false positives is almost reduced in half and the number of false negatives (lower diagonal) is reduced by 75%.

Making further analyzes and try-and-error tests a conclusion was reached, if the sliding window step is bigger the accuracy is improved. It was decided then to add at the very end an extra average pool layer of size 2 and stride also 2, exclusively to the evaluation on the original. This addition does not require to re-train the network, since there is no parameter to learn. Then, the resulting sliding window step is 160. The results on the original database prove the assumption made, as it can be seen in Figure 27c (Appendix B - Table 14). The amount of false positives was reduced by 20 times and the amount of false negatives was reduced by 60%. The maximum accuracy reached 100% and the median accuracy increased 2%.



(a) On network V1.

(b) On network V2.

(c) On network V2 with Average Pool.

Figure 27: Resultant confusion matrix on the original R1 database.

Since ideally there will be only one model for all the references, the network V2 was trained for the five Client X references and then it was evaluated on the whole original database. Its results can be seen in Figure 28 and in Appendix B - Tables 15 and 16.

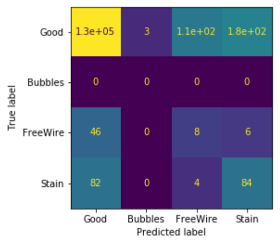


Figure 28: Resultant confusion matrix on the whole original database.

The result over the 83 images from the database is promising. However, there still are a significant amount of false positives and an amount not negligible of false negatives.

To improve the network performance, further architectures changes were made. Since the network is entirely a convolutional networks it was added three fully connected layers at the end, made of convolutions 1×1. Its schematics can be seen in Figure 29 and in more details in Appendix D - Figure 46.

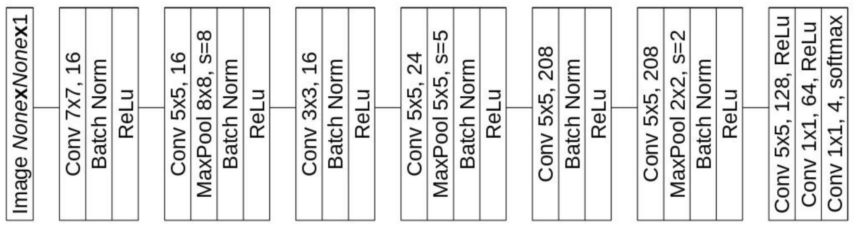


Figure 29: Network structure V3.

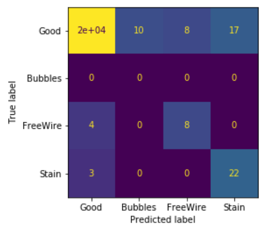
The trained network showed an improvement on the validation and test sets for all references as it can be seen in Appendix B - Table 17. As for the results on the original database, seen in Figure 30 and in Appendix B - Table 18, the amount of false negatives had a significant reduction for all references except R5, which still has a high amount of false free wire detection. The median accuracy also showed a significant improvement, being 100% for R1.

The results related to the network structure are satisfying. From now on all the training is going to be made on Network V3.

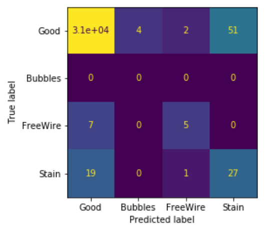
On Figure 31 is depicted the process and some internal results of a thumbnail prediction on Network V3. It is possible to see the feature maps for each convolution, as they go deeper into the network, they stop making sense to the human eye and they start to be understandable for the machine. On Appendix C it is possible to see the results for the other types of defect.

4.1.1 Application on different datasets

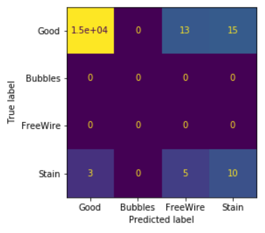
The network structure was defined, however, there is still a significant amount of false detection. One possibility is that the artificial generated dataset is not well representative. When the images that were misclassified by the network were analyzed, some problems were highlighted, as shown in Figure 21 in section 3.3. The new datasets A0 and D1 try to mediate that.



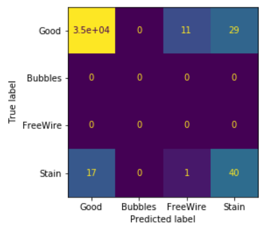
(a) R1.



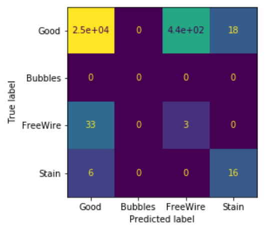
(b) R2.



(c) R3.



(d) R4.



(e) R5.

Figure 30: Resultant confusion matrix on the original database on Network V3.

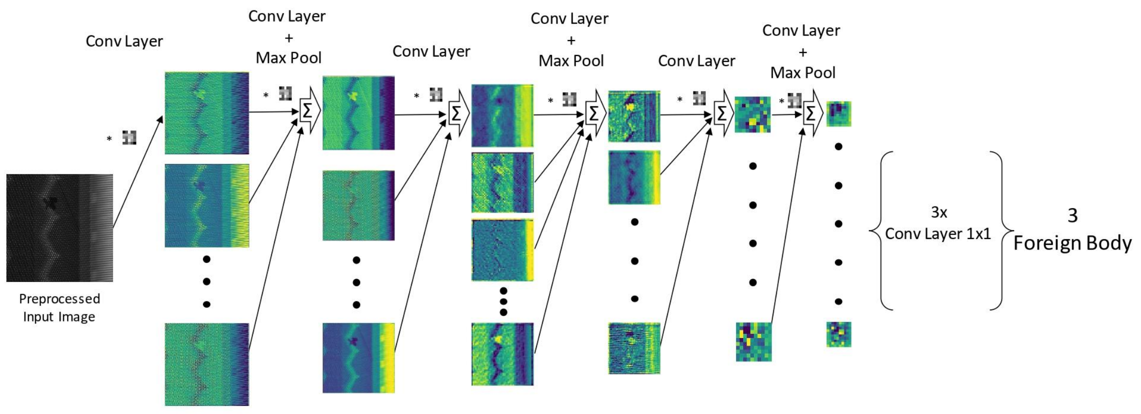


Figure 31: Visual of Internal Network Results - Foreign Body.

Furthermore, it is needed to see if the network is robust, not only for different references but also for different clients. And it is needed to verify if the bubble detection works, since there was none in Client X's database.

Then, tests were made with dataset A0, Client Y's dataset, over the network V3, the results can be seen in Figure 32 and in more details in Appendix B - Table 19 and 20. It can be seen that even with a median accuracy of 99.71% over the database, there are a significant amount of false negatives and false positives. The majority of them are bubble defects.

This evidence shows that our bubble generation does not represent well the reality of the defect, as explained in more details in section 3.3 and as it can be seen in Figure 22. This is the reason why it was decided to revise and work more on the artificial generation on the short therm. However, it can be concluded that the network architecture is robust

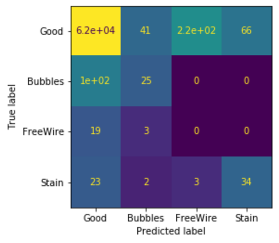
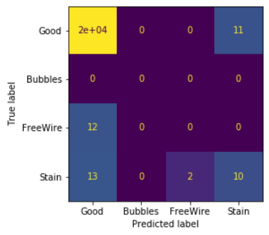


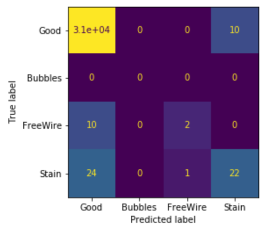
Figure 32: Resultant confusion matrix on the Client Y database.

and works similarly for different clients.

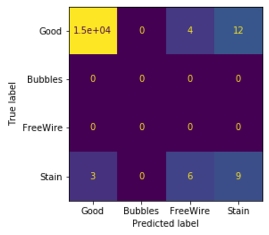
Finally, the performance of D1 was analyzed (Figure 33 and in more details in Appendix B - Table 21 and 22) and it was compared with the previous D0 results. It can be seen that there is a reduction in the number of false positives, on the other hand, there is a significant increase in false negatives. I can be concluded that the network overfitted, further studies about the optimal size of the dataset must be made.



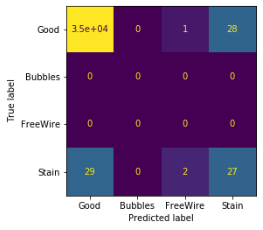
(a) R1.



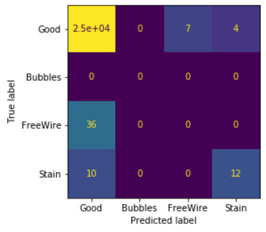
(b) R2.



(c) R3.



(d) R4.



(e) R5.

Figure 33: Resultant confusion matrix on original database for Network V3 in D1.

4.2 Hyper parameters Tuning

The goal of performing a hyper parameter tuning is to obtain a better performance result, in other words, is to obtain a higher accuracy on the datasets and on the original database. Each hyper parameter changes the way that the network learns in its own way, as is described on the beginning of each section. The analyzes were done separately in order to obtain a better visualization.

4.2.1 Batch size

The batch size is a set of N samples. Each batch is processed independently, in parallel and they result in only one update to the model. A study was done over the batch size to verify which amount worked best. The use of small batch sizes lead to faster training dynamics and good generalization, however it may never converge to the global optima. The bigger the batch size the greater are the chances of arriving on the global optima, but slower is the convergence.

At first the reference R2 was used with batch sizes equal to 16, 32, 64, 128, 256. The dataset used was D0 and the network was trained over 30 epochs, with validation frequency of 100. The results over the original database are displayed in Table 3.

When analyzed the minimal accuracy, the results from 128 and 16 are not satisfactory. The others show similar performance through the three criteria. However, if looked to the total amount of false positives, it is possible to see that 64 has a good low quantity.

Then, 64 and 256 were chosen to be evaluated more thoroughly. This time for the first test, it was used Client Y tires over its own dataset A0 under the same configuration as previous. For the second test it was used the same reference R2, trained on the reduced dataset D2 over 100 epochs, with validation frequency of 10.

The results over the original database, Table 4, show a better performance with batch size 256, which was the same used when developing the network structure.

	Accuracy				
	256	128	64	32	16
Max	100	100	100	100	99.78
Min	99.05	98.82	99.27	99.16	97.87
Median	99.73	99.95	99.95	99.78	99.06

Table 3: R2 Batch size from 256 to 16.

	Accuracy			
	Client Y – 256	Client Y - 64	R2 – 256	R2 – 64
Max	100	100	100	100
Min	95.77	91.95	98.11	97.72
Median	98.35	95.46	99.66	99.72

Table 4: Batch size test follow up.

4.2.2 Number of epochs

The number of epochs defines how many times the entire dataset passes over the network. A small number of epochs can lead to underfitting, because the network was unable to learn main features, resulting in a weak accuracy on all datasets. On the other hand a big number of epochs can lead to overfitting, the network learns too much, even the noise specific to the training set, also resulting in a weak accuracy, but this time only in the validation and test sets.

Throughout the many results, the accuracy and loss curves over the epochs were analyzed for the training and the validation sets. Over the course of 30 epochs, for the majority of the networks, the training accuracy was always rising, as expected, the validation accuracy was a little oscillatory, but it was also rising. On the other hand, as the loss curve decreased for the training over the epoch, the validation one started to slowly increase after the 15^{th}-20^{th} epoch. This behavior show that the network is overfitting on the training set.

Other aspect present on most results is the non variance of the accuracy also starting around the 15^{th}-20^{th} epoch, the value change so little that is worth to stop the training to save time. A visual example of both behaviors can be seen in Appendix B - Figure 40.

In Keras library there are two functions to use in those cases: *ModelCheckPoint* and *EarlyStopping*, respectively. The first was configured to monitor the validation loss and save the network weights when the loss is smaller than its previous value, allowing to always have the best fitting result. The latter also monitors the validation loss, stopping the learning when its value has increased over the last five epochs, saving processing time.

4.2.3 Learning Rate

Learning rate is a hyper-parameter that controls how much the network weights are going to be adjusted in respect to the loss gradient. The lower the value, the slower, higher are the chances to not miss any local minima, however it could take a longer converge.

So far it was used the optimizer Adam with the automatic initial value 10^{-3}. To verify if those choices were the best some tests were made. The evaluations were made with network V3 over the A0 dataset.

- **Learning Rate Finder :** Trough the use of a function developed by in [9], that is based on the work from [18], it was analyzed the resulting graph of the loss per LR, Figure 34. The optimal LR can be found were the curve is the steepest, in this case 10^{-4} is the value where this happens. Upper and lower values are also defined, avoiding the parts where the curve is ascending, as 10^{-3} and 10^{-4}.

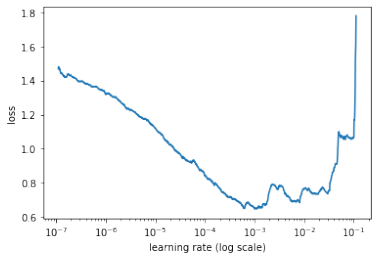
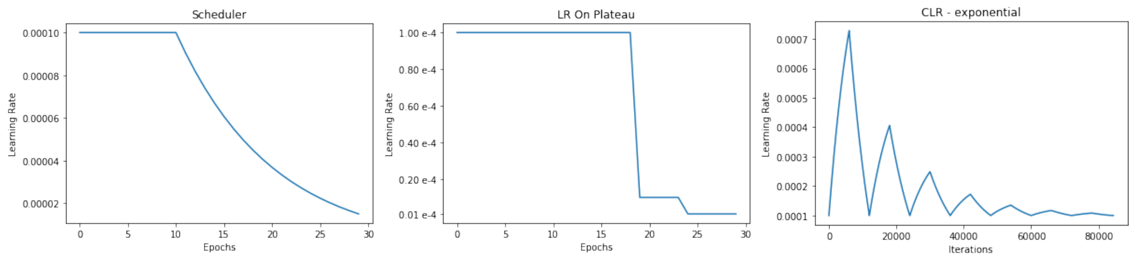


Figure 34: Result LR finder.

- **Scheduler :** The identified values were used to create a learning schedule over the epochs, where it would be constant on the first ten epochs with value 10^{-4} and then it would decrease exponentially as shown in Figure 35a.
- **Reduce Learning Rate On Plateau :** This is a built-in function from Keras that changes the learning rate by a desired factor when the monitored value, usually the loss, does not improve after a certain amount of epochs. An example is depicted in Figure 35b
- **Cyclic Learning Rate :** The work done in [18] proposes a change on the LR over the iterations in a epoch. It is defined a lower bound and an upper bound that changes exponentially. They were chosen as said above, its behavior can be seen in Figure 35c.

Other two tests were made, one with the LR fixed in 10^{-4} and the other with the initial value for Adam as 10^{-4}. The result over the database can be seen in Table 5, there is also previous method result displayed on the far right.

As it can be seen none of the results showed any improvement. It can be concluded that Adam is already very well adapted to change the LR as is convenient on the situation, as explained on section 2.3.3.



(a) Scheduler. (b) Reduce LR On Plateau. (c) CLR - exponential.

Figure 35: Learning rate curves.

	Accuracy					
	0.0001	**Scheduler**	**LR On Plateau**	**Adam 0.0001**	**CLR**	**Adam Auto**
Max	100	99.59	99.48	99.79	99.9	100
Min	90.92	89.99	87.51	97.42	86.58	98.14
Median	93.81	93.09	92.98	98.76	90.61	99.17

Table 5: LR test result on database.

4.3 Grad-CAM

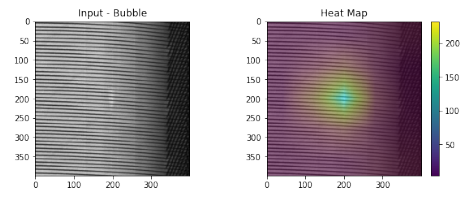
CNN is a like black-box, the input is controlled and the output is observable, however it is not know exactly whats going on inside. Sometimes the network can learn unexpected and undesired features and base its predictions on it, working well on the development database but poorly in real life application. Understanding the reasons of this behavior is not a simple task, since as the network goes deeper, the features maps become more abstract for the human comprehension.

One way to open a little this black-box is by using Gradient-weighted Class Activation Mapping (Grad - CAM) [15]. It allows to understand each neuron through the gradient information flowing into the last convolutional layer of the CNN. In this project's case, all layers are convolutional layers, then it would be the last convolutional layer before the convolutions 1×1. This tool also provides better localization which guides to the demystification of the complexity behind CNNs.

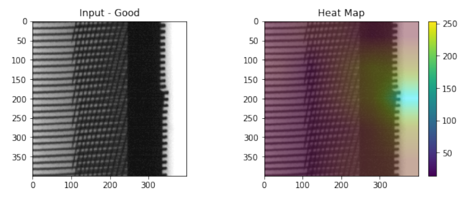
In this project Grad - CAM was developed and used throughout the many steps described in section 4.1. It was verified if the network was identifying the right features and it was especially used for understanding why a false prediction was made. Some results are displayed in Figure 36.

It can be seen that when the network succeeded in classify the images, Figures 36a and 36c, the right features were the ones identified. In addition, on the case of a false positive, Figure 36b, it is clear that the network misinterpreted a discontinuity on the image section generated on the acquisition phase.

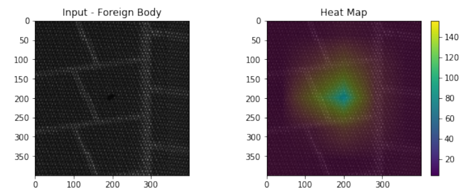
However, on the case of a false negative, Figure 36d, the conclusion is not that straight forward. The network classified it as "No defects", then all the zones highlighted by Grad-CAM are the parts of the image that lead the network to this result. The defect on the image was not identified, it can be speculated that the training dataset is not representative enough, or maybe that the artificial generation of free wires does not correspond to the reality. Both ideas were taken into consideration during the network development.



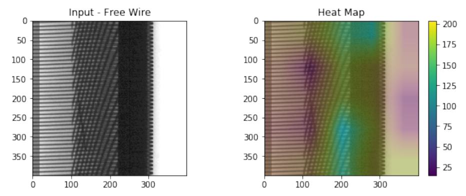
(a) True positive.



(b) False Positive.



(c) True Positive.



(d) False Negative.

Figure 36: Grad - CAM Results.

5 Conclusion

This project focused on the development of a deep learning algorithm able to detect, classify and localize defects on a manufactured tires. Within constraints of time cycle in an in-line production. In order to solve this problematic two computer vision approaches were developed over a processed database.

The database was not representative enough nor has it enough samples for a neural network to learn on. An artificial generation was implemented to meet this need, however it was evidenced that there were some flaws. Some of them were corrected over this project and the changes affected positively the results, nevertheless there is still room for improvement. The work will be carried on with a high priority. It will focus on fixing the gray levels topology and not only enhancing the overall shapes to be more like the defects found in the industry, but also augmenting the variety of those shapes.

This project developed a classification network structure for identifying and localizing defects on tire trucks x-rays. The results over the whole x-ray images showed an accuracy of around 98% and a prediction time of just a few seconds. However, this metrics can be misleading. The prediction is made over a image of around 3.000×17.000 pixels, with the observation window of 400×400 and a stride of 160, which means that in reality about 1100 predictions are made. Which means for an accuracy of 99% there are more than 20 false positives and/or false negatives per image.

This prediction method allows the localization of defects, however it is not precise given the size and stride of the observation window. A defect of height and width of 60 pixels can have a localization of 560×560 or even more, representing only 1% of the detected area.

The hyper parameter tuning was an important step, even if most of the results were confirmations that what was used before was already adequate. This can be explained by the fact that the network and its hyper parameters were provided by the company, where they were tuned for other use.

One analysis that still needs to be done is the optimal dataset size. When was briefly discuss in section 4.1.1 it was seen that this is a very relevant question to be made, however due the time constraints and it was decided to give preference to other topics.

Overall the project had promising results and a good evolution. Due to the complicated nature of deep learning algorithms there is still a significant amount of work to be done before applying this algorithm in the industry. The work will be continued by the company.

This internship was an opportunity to put into practice the dynamic training promoted by Grenoble INP - Ense^3. As a personal learning experience, this project has given me autonomy. I was able to propose, discuss and develop ideas that I found worthy. The weekly follow-up with my supervisors was a very enriching environment, results were discussed, ideas were exchanged and propositions were made. Allowing me to grow as a professional and to improve my communication, given that French is my third language. I have also developed my presentation skills through monthly exposition to coworkers on the research and development area.

In addition, I had the opportunity to work with different people in the company in an open-space. This was not only an enriching experience but it also gave me a better perception of the professional environment. As a result of the coronavirus pandemic I also had the experience of working remotely, which has taught me the importance of a clear and effective communication, record keeping and time management.

On top of that, it has allowed me to acquire a very enriching experience in machine learning and image processing. An area of study that I had already worked with, both together and separate. However, this time I was able to go deeper in the field with hands-on experience and autonomy on a work of an exciting complexity. I hope on the future to keep studying and working on those subjects.

6 Bibliography

[1] Alejandro Baldominos, Yago Sáez, and Pedro Isasi. Evolutionary convolutional neural networks: an application to handwriting recognition. *Neurocomputing*, 283:38–, 03 2018.

[2] François Chollet et al. Keras. `https://keras.io`, 2015.

[3] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[7] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

[8] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2013.

[9] Arun S. Maiya. ktrain: A low-code library for augmented machine learning, 2020.

[10] Milestar. Tire construction. `https://www.milestartires.com/help/tires-101/tire-construction/`, 2020.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[12] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.

[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[14] Seyed Sadegh Mohseni Salehi, Deniz Erdogmus, and Ali Gholipour. Tversky loss function for image segmentation using 3d fully convolutional deep networks, 2017.

[15] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, Oct 2019.

[16] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks, 2013.

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[18] Leslie N. Smith. Cyclical learning rates for training neural networks, 2015.

[19] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[20] Kentaro Wada. labelme: Image Polygonal Annotation with Python. `https://github.com/wkentaro/labelme`, 2016.

[21] Pavel Yakubovskiy. Segmentation models. `https://github.com/qubvel/segmentation_models`, 2019.

APPENDIX

A Network architectures from bibliography

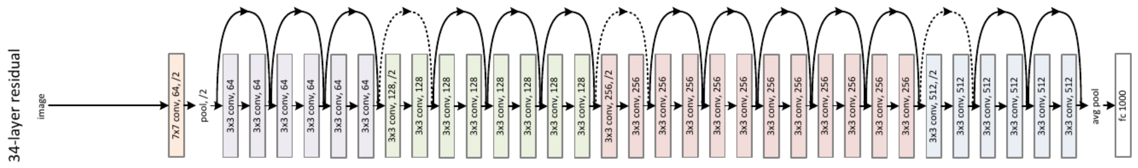


Figure 37: ResNet's architecture [4]

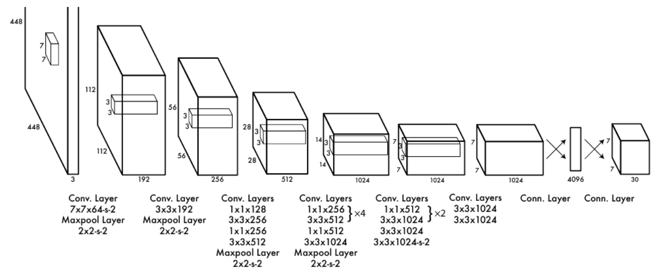

448
448
112
56
28
14
7
3
192
256
512
1024
4096
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 38: YOLO's architecture [12]

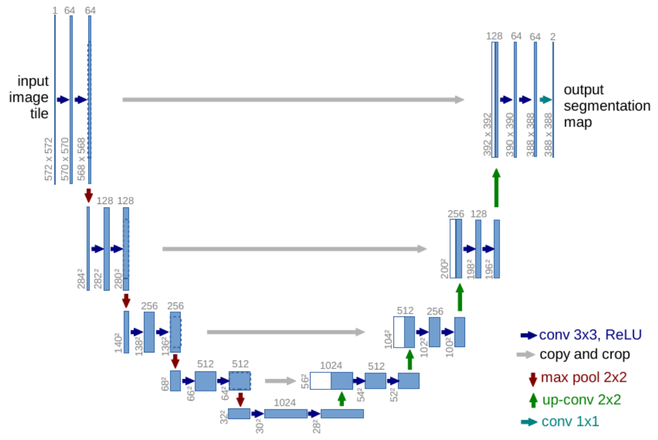


Figure 39: U-Net's architecture [13]

B Results

	Accuracy			Loss	
	Training	Validation	Test	Training	Validation
R1	99.22	98.73	0.99	0.021	0.045
R2	98.44	98.69	0.99	0.048	0.045
R3	98.83	98.18	0.98	0.024	0.057
R4	98.74	-	-	-	-
R5	96.48	96.84	0.97	0.097	0.093
All	97.66	98.42	98.50	0.090	0.051

Table 6: Results Matlab.

	Accuracy			Loss		
	Training	Validation	Test	Training	Validation	Test
R1	98.90	98.35	98.34	0.037	0.057	0.055
R2	98.72	95.78	95.95	0.041	0.160	0.142
R3	98.13	97.66	97.53	0.057	0.069	0.075
R4	98.97	97.97	97.56	0.034	0.061	0.069
R5	97.11	87.63	87.84	0.084	0.584	0.549

Table 7: First network's adaptation to Python.

	Accuracy			Loss		
	Training	Validation	Test	Training	Validation	Test
R1	98.83	98.12	97.80	0.040	0.064	0.071

Table 8: Network with convolutional sliding window.

	Accuracy		
	Training	Validation	Test
Conv-BN-ReLu	99.85	99.65	99.62
Conv-ReLu-BN	99.79	99.50	99.27
BN-Conv-ReLu	99.64	99.50	99.29

Table 9: Layer order test on R1.

	Accuracy		
	Training	Validation	Test
R1	99.85	99.65	99.62
R2	99.80	99.43	99.21
R3	99.83	99.53	99.46
R4	99.84	99.40	99.36
R5	99.77	98.62	98.37

Table 10: Results Network V1.

	Accuracy	F1 score
Max	97.37	98.67
Min	94.15	96.29
Median	95.99	97.96

Table 11: Network V1 on original R1 database.

	Accuracy		
	Training	Validation	Test
R1	99.94	99.50	99.51

Table 12: Results Network V2.

	Accuracy	F1 score
Max	98.61	99.24
Min	95.05	97.46
Median	97.44	98.60

Table 13: Network V2 on original R1 database.

	Accuracy	F1 score
Max	100	100
Min	97.77	98.5
Median	99.47	99.74

Table 14: Network V2 with average pool on original R1 database.

	Accuracy			Loss		
	Training	Validation	Test	Training	Validation	Test
All	99.91	99.22	99.46	0.002	0.047	0.035

Table 15: Results Network V2 for all Client X references.

	Accuracy	F1 score
Max	100	100
Min	97.67	97.18
Median	99.76	99.87

Table 16: Network V2 on original database.

	Accuracy		
	Training	Validation	Test
R1	99.83	99.78	99.68
R2	99.82	99.56	99.38
R3	99.83	99.44	99.48
R4	99.93	99.53	99.51
R5	99.77	98.81	98.57

Table 17: Results Network V3.

	R1		R2		R3		R4		R5	
	Accuracy	F1_score	Accuracy	F1_score	Accuracy	F1_score	Accuracy	F1_score	Accuracy	F1_score
Max	100	100	100	100	100	100	100	100	99.34	99.67
Min	98.22	99.1	99.05	99.52	99.3	99.65	99.21	99.6	94.12	96.97
Median	100	100	99.73	99.86	99.91	99.96	99.88	99.94	98.87	99.43

Table 18: Network V3 on original database.

	Accuracy		
	Training	Validation	Test
Client Y	99.92	99.84	99.71

Table 19: Results V3 on A0.

	Accuracy	F1 score
Max	100	100
Min	98.14	98.13
Median	99.17	99.48

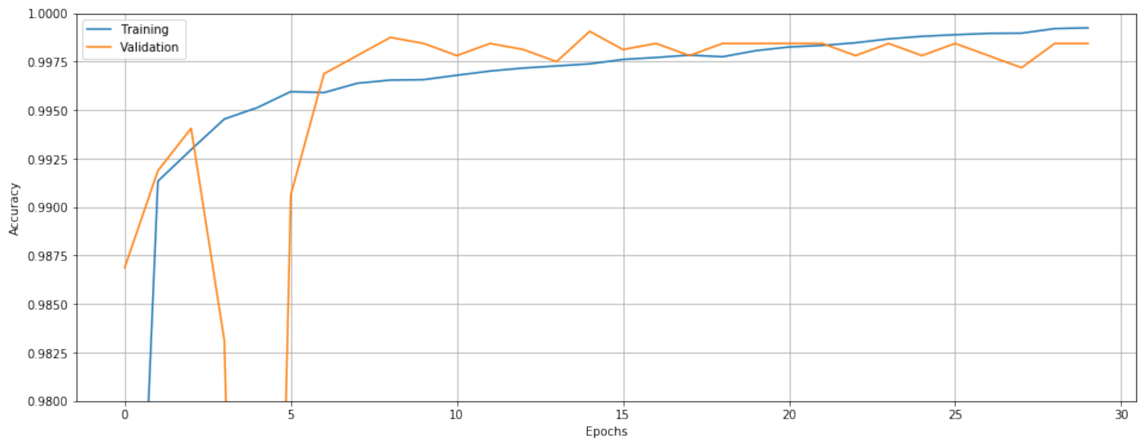
Table 20: V3 trained over A0 on Client Y database.

	Accuracy		
	Training	Validation	Test
R1	99.92	99.91	99.89
R2	99.91	99.78	99.87
R3	99.94	99.81	99.90
R4	99.92	99.87	99.87
R5	99.58	99.72	99.39

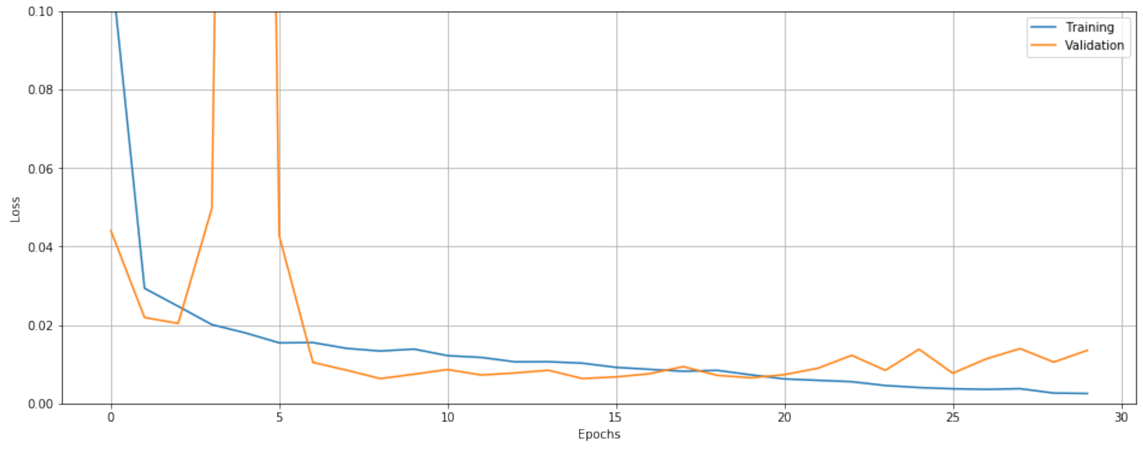
Table 21: Results V3 on D1.

	R1		R2		R3		R4		R5	
	Accuracy	F1_score	Accuracy	F1_score	Accuracy	F1_score	Accuracy	F1_score	Accuracy	F1_score
Max	100	100	100	100	100	100	100	100	100	100
Min	98.84	98.64	99.27	99.08	99.21	99.36	99.03	99.27	98.34	97.51
Median	100	100	100	100	100	100	100	100	100	100

Table 22: V3 trained over D1 on the original database.



(a) Accuracy.



(b) Loss.

Figure 40: Training and validation over the epochs for A0.

C Network Internal Scheme

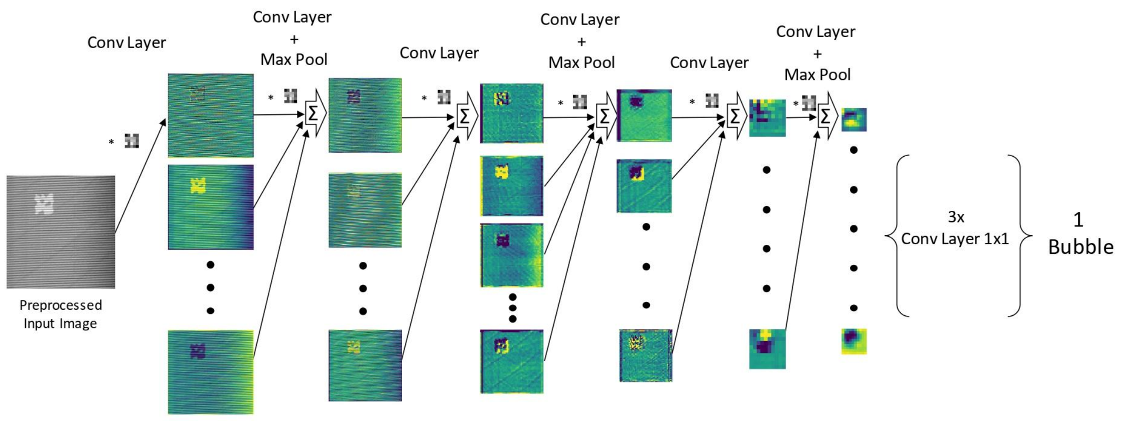


Figure 41: Visual of Internal Network Results - Bubble.

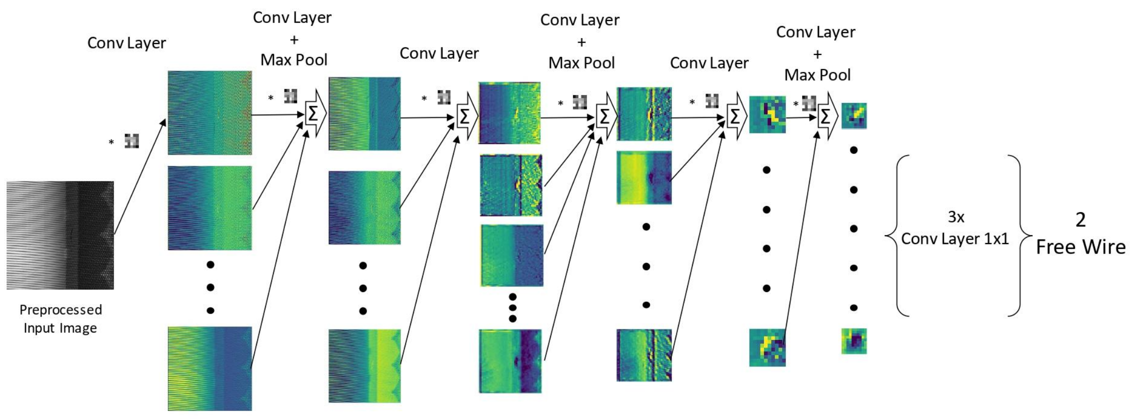
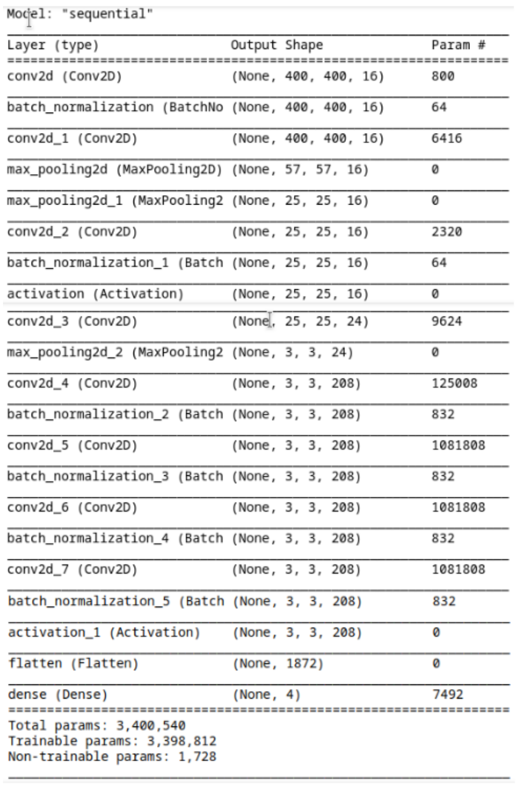


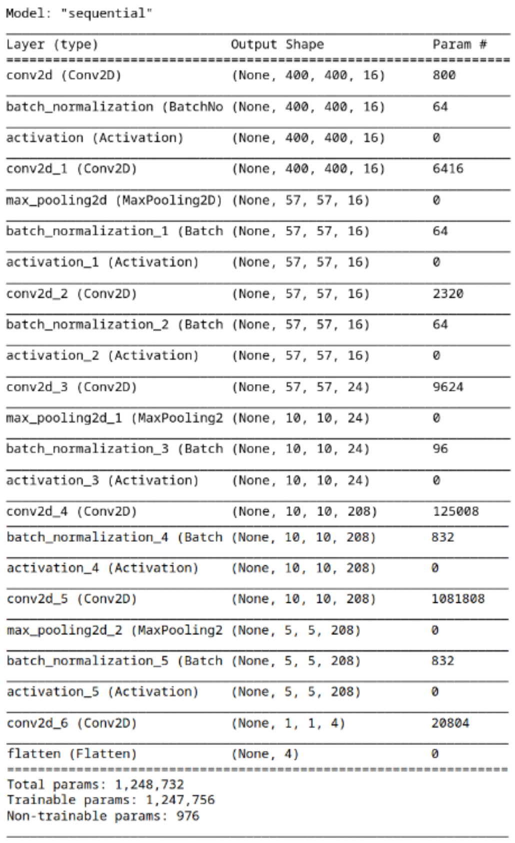
Figure 42: Visual of Internal Network Results - Free Wire.

D Networks architecture



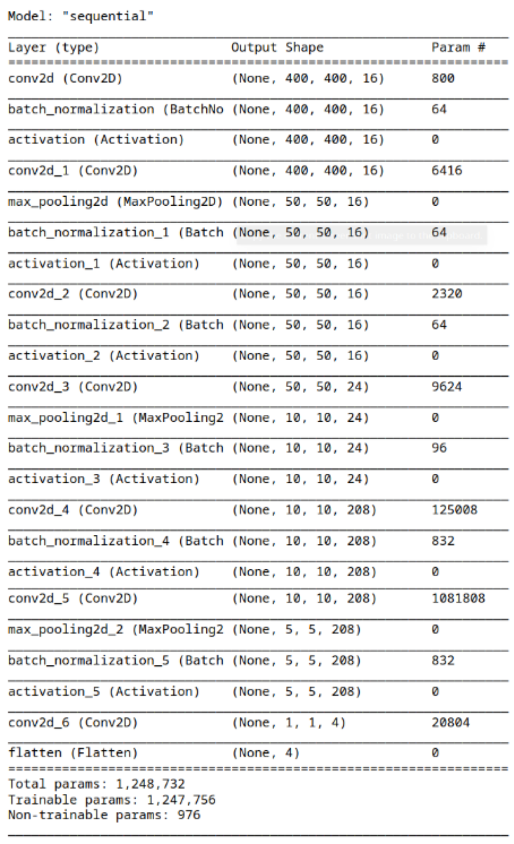
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 400, 400, 16)      800
_________________________________________________________________
batch_normalization (BatchNo (None, 400, 400, 16)      64
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 400, 400, 16)      6416
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 57, 57, 16)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 25, 25, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 25, 25, 16)        2320
_________________________________________________________________
batch_normalization_1 (Batch (None, 25, 25, 16)        64
_________________________________________________________________
activation (Activation)      (None, 25, 25, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 25, 25, 24)        9624
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 24)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 3, 208)         125008
_________________________________________________________________
batch_normalization_2 (Batch (None, 3, 3, 208)         832
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 208)         1081808
_________________________________________________________________
batch_normalization_3 (Batch (None, 3, 3, 208)         832
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 3, 3, 208)         1081808
_________________________________________________________________
batch_normalization_4 (Batch (None, 3, 3, 208)         832
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 3, 3, 208)         1081808
_________________________________________________________________
batch_normalization_5 (Batch (None, 3, 3, 208)         832
_________________________________________________________________
activation_1 (Activation)    (None, 3, 3, 208)         0
_________________________________________________________________
flatten (Flatten)            (None, 1872)              0
_________________________________________________________________
dense (Dense)                (None, 4)                 7492
=================================================================
Total params: 3,400,540
Trainable params: 3,398,812
Non-trainable params: 1,728
_________________________________________________________________
```

Figure 43: First network structure.



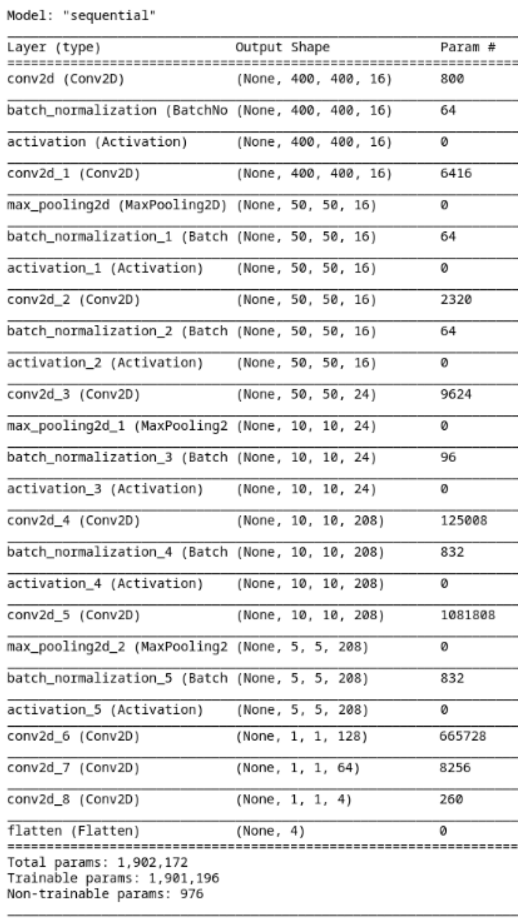
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 400, 400, 16)      800
_________________________________________________________________
batch_normalization (BatchNo (None, 400, 400, 16)      64
_________________________________________________________________
activation (Activation)      (None, 400, 400, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 400, 400, 16)      6416
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 57, 57, 16)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 57, 57, 16)        64
_________________________________________________________________
activation_1 (Activation)    (None, 57, 57, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 57, 57, 16)        2320
_________________________________________________________________
batch_normalization_2 (Batch (None, 57, 57, 16)        64
_________________________________________________________________
activation_2 (Activation)    (None, 57, 57, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 57, 57, 24)        9624
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 24)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 10, 10, 24)        96
_________________________________________________________________
activation_3 (Activation)    (None, 10, 10, 24)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 208)       125008
_________________________________________________________________
batch_normalization_4 (Batch (None, 10, 10, 208)       832
_________________________________________________________________
activation_4 (Activation)    (None, 10, 10, 208)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 208)       1081808
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 208)         0
_________________________________________________________________
batch_normalization_5 (Batch (None, 5, 5, 208)         832
_________________________________________________________________
activation_5 (Activation)    (None, 5, 5, 208)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 1, 1, 4)           20804
_________________________________________________________________
flatten (Flatten)            (None, 4)                 0
=================================================================
Total params: 1,248,732
Trainable params: 1,247,756
Non-trainable params: 976
_________________________________________________________________
```

Figure 44: Network V1 structure.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 400, 400, 16)      800
_________________________________________________________________
batch_normalization (BatchNo (None, 400, 400, 16)      64
_________________________________________________________________
activation (Activation)      (None, 400, 400, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 400, 400, 16)      6416
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 50, 50, 16)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 50, 50, 16)        64
_________________________________________________________________
activation_1 (Activation)    (None, 50, 50, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 50, 50, 16)        2320
_________________________________________________________________
batch_normalization_2 (Batch (None, 50, 50, 16)        64
_________________________________________________________________
activation_2 (Activation)    (None, 50, 50, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 50, 50, 24)        9624
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 24)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 10, 10, 24)        96
_________________________________________________________________
activation_3 (Activation)    (None, 10, 10, 24)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 208)       125008
_________________________________________________________________
batch_normalization_4 (Batch (None, 10, 10, 208)       832
_________________________________________________________________
activation_4 (Activation)    (None, 10, 10, 208)       0
conv2d_5 (Conv2D)            (None, 10, 10, 208)       1081808
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 208)         0
_________________________________________________________________
batch_normalization_5 (Batch (None, 5, 5, 208)         832
_________________________________________________________________
activation_5 (Activation)    (None, 5, 5, 208)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 1, 1, 4)           20804
_________________________________________________________________
flatten (Flatten)            (None, 4)                 0
=================================================================
Total params: 1,248,732
Trainable params: 1,247,756
Non-trainable params: 976
_________________________________________________________________
```

Figure 45: Network V2 structure.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 400, 400, 16)      800
_________________________________________________________________
batch_normalization (BatchNo (None, 400, 400, 16)      64
_________________________________________________________________
activation (Activation)      (None, 400, 400, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 400, 400, 16)      6416
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 50, 50, 16)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 50, 50, 16)        64
_________________________________________________________________
activation_1 (Activation)    (None, 50, 50, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 50, 50, 16)        2320
_________________________________________________________________
batch_normalization_2 (Batch (None, 50, 50, 16)        64
_________________________________________________________________
activation_2 (Activation)    (None, 50, 50, 16)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 50, 50, 24)        9624
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 10, 10, 24)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 10, 10, 24)        96
_________________________________________________________________
activation_3 (Activation)    (None, 10, 10, 24)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 10, 10, 208)       125008
_________________________________________________________________
batch_normalization_4 (Batch (None, 10, 10, 208)       832
_________________________________________________________________
activation_4 (Activation)    (None, 10, 10, 208)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 10, 10, 208)       1081808
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 208)         0
_________________________________________________________________
batch_normalization_5 (Batch (None, 5, 5, 208)         832
_________________________________________________________________
activation_5 (Activation)    (None, 5, 5, 208)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 1, 1, 128)         665728
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 1, 1, 64)          8256
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 1, 1, 4)           260
_________________________________________________________________
flatten (Flatten)            (None, 4)                 0
=================================================================
Total params: 1,902,172
Trainable params: 1,901,196
Non-trainable params: 976
_________________________________________________________________
```

Figure 46: Network V3 structure.