Robustness Analysis of a Convolutional Neural Network for Image Recognition in an Industrial Setting

Michele Zanchi

December 2018

Contents

Abstract

Machine learning is a set of techniques that allow to create models starting from data, without the need to explicitly code the rules. Thanks to machine learning it is possible to create systems that can solve complex tasks such as image recognition, language translations or autonomous driving. Machine learning can be adopted in an industrial context to support humans in complex and/or repetitive tasks.

When deploying a machine learning model to production there are two critical aspects to keep into consideration. The first one is the model robustness when dealing with unseen data and the second one is the interpretability of the model decisions.

When deployed to production the performance of the model is highly dependent on the data-set used during the training phase. If this is not representative of the real world, the model can show unexpected behaviors. This issue can occur in an industrial setting for two main reasons: environmental condition changes (e.g. seasonality) and data acquisition issues. This work describes a novel approach to assess the robustness for image recognition, when dealing with unseen images in production.

Machine learning models can be very accurate but their decisions are not trivial to interpret. As a rule of thumb, higher performance models rely on more complex mathematical representation at the expense of interpretability, and interpretability is crucial for business adoption and human acceptance and trust. Despite the power of machine learning solutions there are still some concerns about their application in mission critical scenarios. Understanding the reasons that triggered a model decision is one of main challenges of using complex machine learning models such as neural networks. This thesis proposes a novel approach to visualize the areas of interest in an image that drove the decision of the network.

This thesis describes the results of the activities conducted to improve a system for the au-

tomatization of an industrial crane, with particular focus on safety aspects. This project is realized in partnership with Tenaris R&D Data Science team and refers to a real industrial case. The results of this project are currently deployed and used in production.

Document Outline

This chapter briefly describes the structure of the document providing the basic concepts discussed exhaustively in each section.

- **Chapter 1** introduces the reader to the machine learning and to the case of study describing the main aspects: context, metrics, data-set and legacy network.
- **Chapter 2** provides to the reader the basic knowledge to the neural network world describing the architecture and the an intuition about how neural networks work. The chapter introduces the particular type of machine learning models, amply discussed in this document, namely Convolutional Neural Networks, which are state of the art for image classification.
- **Chapter 3** formalizes a novel robustness metric that can be applied to a generic machine learning model. The approach allows to measure the robustness of model against not nominal inputs and introduces an implementation methodology.
- **Chapter 4** describes a novel testing framework to assess the model robustness in perturbed environments. It is also reported a representative example of this analysis.
- **Chapter 5** explores the data augmentation technique, i.e. enriching the input domain with perturbed data to enhance model robustness. The chapter introduces also a novel technique to enrich the input data-set and to train the model without a complete time-consuming retraining.
- **Chapter 6** introduces the reader to a novel approach to visualize the focus area of a Convolutional Neural Network. The technique might help humans in the understanding of which image details triggered the model decisions.

- **Chapter 7** analyzes the impact of network architecture on the overall robustness, comparing the case study model with a simplified version of the model, obtained by pruning network neurons. The section will show us the importance of an accurate choice of the network structure and presents an automatic tool to reduce the model size.

- **Chapter 8** describes the implementation of software verification for a simple machine learning model. This study might help the researchers for discover critical input combinations that could drive the model to wrong critical outputs

- **Chapter 9** summarizes the results obtained in the work and describe how the model is now used in daily production.

Chapter 1

Introduction

Machine learning is an interdisciplinary subject in which statistical and modeling techniques are applied in order to allow the machines to *learn* themselves from the data. This field includes simple model, for example linear classifier, or very complex model like neural networks. This feature allows the data scientists to have not to explicitly describe the actions to achieve their targets: the machine tries to define its behavior from the data during *training phase*. Machine learning allows the researches to generate models able to solve complex challenge without having a complete knowledge on the problem on the condition to own a relative big number of samples and a discrete computation power.

The recent increase of the amount of the available data and the explosion of the execution power, thanks to the development of a particular hardware called Graphic Processing Unit, allowed the rapid growth of machine learning. The Graphic Processing Unit allows PCs to perform several matrix operations at the same time improving the computation power to a factor 5 ([8]). The explosion of machine learning made necessary major efforts on the analysis and study of the robustness of the models, in particular for the neural network. Neural networks are complex machine learning models used to solve varied tasks with a black box approach.

Today machine learning models can delivery human results in many tasks: images classification, games and others. However, machine learning researchers are not able to understand perfectly how the models take the decisions and in general, they see the models under a black-box perspective; this involves the researchers can't be confident in the network behavior in not nominal conditions. In general, it is not possible applying the

traditional white-box tests in order to improve the developer confidence on the models. In this scenario, it is possible applying only traditional coverage methods on input domains; but also in a very simple case, like the one presented in this work, the input domain is very ample and hard to explore. In particular, in image classification, there is the extra problem of the gathering of the data: it is the most expensive phase in all the development process and for production environment can be take a very long time. That is why researchers are used to generating synthetically the data-sets.
In this project, we want to formalize a simple black-box definition of robustness for image classification models and using it to test and to improve the robustness of a convolutional neural network. The work is divided into 5 steps:

- Formalize and analysis of the robustness of network under test.
- Using data augmentation to improve the network robustness.
- Using an attention analysis to improve our knowledge about the network.
- Verify the presence of adversarial examples for a logistic network.
- Using all the previous knowledge to deploy the best network for the case study.

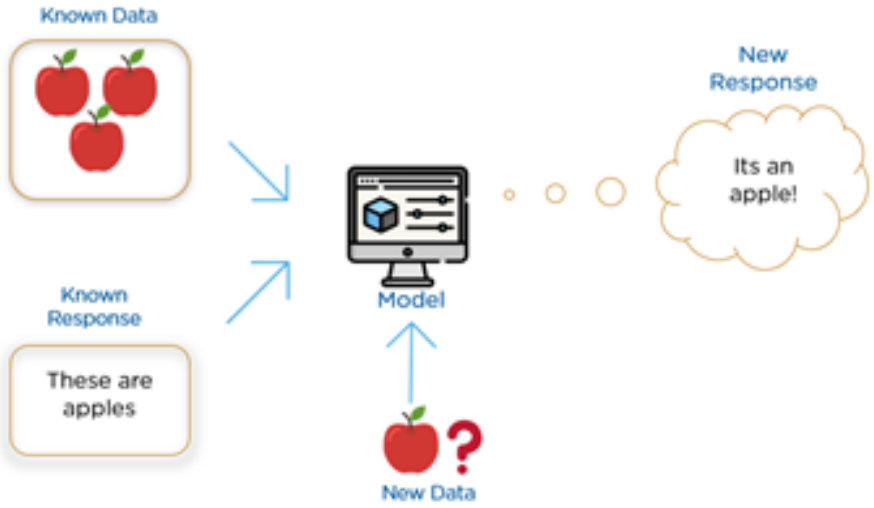


Figure 1.1: Example of machine learning task (courtesy from [1]).

1.1 Case Study and Industrial Setting

In this work, we will analyze the case study of a particular type of machine learning model: *Convolutional Neural Network*. This model is heart of an automatic pipeline used to command an industrial crane. During a stage of the production process, this crane has the task to lay down some semi-finished bars, called billets, on three different benches. The model is the main part of the pipeline because the behavior of all the system is based on its output. The crane performs the operation only in case of empty classification from the network. The project has the important challenge to improve the safety of the entire system.

1.1.1 Metrics

First of all, we define the metrics used in the work in order to measure and to evaluate the performance of the network in the tests. State of the art of classification tasks use the metrics of accuracy, precision and recall to evaluate the goodness of the model. The metrics represent the different ratio of positive and negative outputs. We classified the network's outputs according to our oracle as showed in Table 1.1.

Prediction	**Actual**	**Classification**
Full	Full	Positive
Empty	Empty	Negative
Full	Empty	False Positive
Empty	Full	False Negative

Table 1.1: Output classification.

Each metric provides us different information about the model behavior:

- **Accuracy** gives information about the precision of the model: we are able to know the rate of the corrected classifications but we are not able to know the type of error (1.1).

$$Accuracy = \frac{True\ positive + True\ negative}{Population} \tag{1.1}$$

- **Precision** gives information based on the quantity of false positive outputs. For this case study precision measures the efficiency of the model: in case of a full classification the crane doesn't perform any operation (1.2).

$$Precision = \frac{True\ positive}{True\ positive + False\ positive} \tag{1.2}$$

- **Recall** gives information about the quantity of false negative outputs. For this case study recall measures the safety of the model: in this case, the crane will move a new billet to a full bench ruining the billets already present on the bench (1.3).

$$Recall = \frac{True\ positive}{True\ positive + False\ negative} \tag{1.3}$$

1.1.2 Legacy System

A Convolutional Neural Network (CNN) is a particular type of neural networks that includes convolutional layers (for more information see the chapter 2); it represents the state of the art for features extraction and for images classification. The CNN under analysis is used in a real crane command pipeline to categorize the images in *empty* or *full* according to the presence or not of billets in the picture. The network takes as input images captured from 3 fixed cameras positioned over 3 benches. In particular, the network classifies an image as full if it is able to recognize the presence at least of one billet, otherwise, the image is classified as empty.

Each photo is pre-processed in order to obtain two smaller and simpler images that the network is able to analyze; the pre-processing consists of a series of steps:

- Conversion to grayscale.
- Normalization: scales the color values between the range [0,1].
- Equalization: distributes the lightning values in all the scale.
- Cropping on the focus areas. From each image, two smaller pictures are generated according to the fixed masks.
- Fixing the perspective.

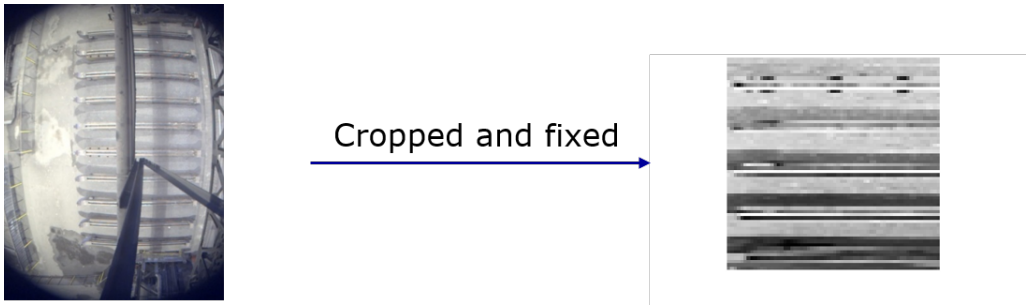


Figure 1.2: Sample of input and output of the pre-processing phase.

Network returns as output a probability used to classify the images. The prediction is compared with a threshold and reduced to 0 or 1: 0 represents an empty image while 1 represents a full one. The choice of the threshold is a tricky phase: a small threshold involves in more images classified as full and it has an effect on the efficiency of the system because increasing the number of false positives. On the other hand, a bigger threshold involves the number of false negatives and impacts on the safety of the system. From an experimental phase, it resulted that the best threshold for this problem is 0.5.

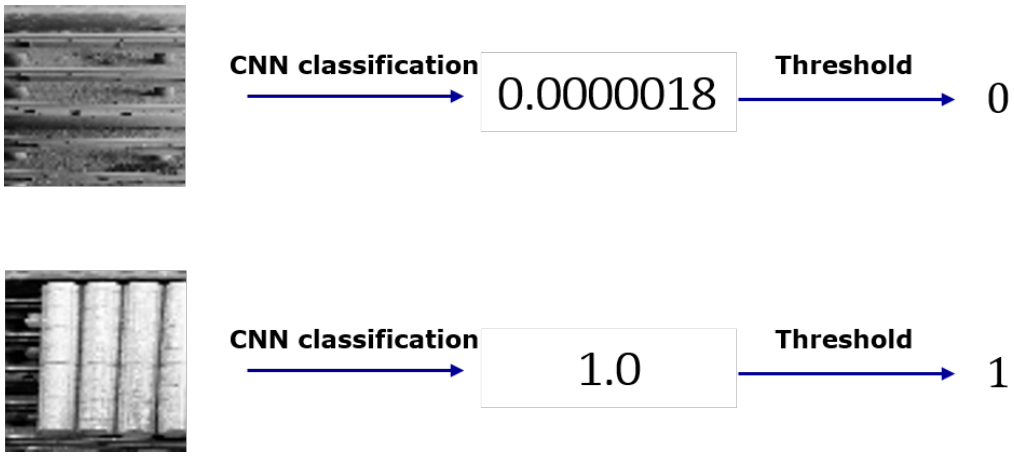


Figure 1.3: Example of classification

1.1.3 Data-set

The work is based on a data-set of 3057 images of three benches; the images are cropped and the perspective fixed before taking in input from the network. From each picture, two different crops are generated so the final data-set counts 6114 images. The photos have a dimension of (580x780x3), where the first number represents the width, the second the

height and the third the number of colors; they are cropped and fixed in (64x64x1); we can also notice the cropped images are in grayscale (1.4). The data-set was manually classified and each image has a label used as oracle during the training and the testing phase. The data-set is unbalanced: there are 4147 images classified as empty (67.8%) and 1969 images classified as full (32.2%).

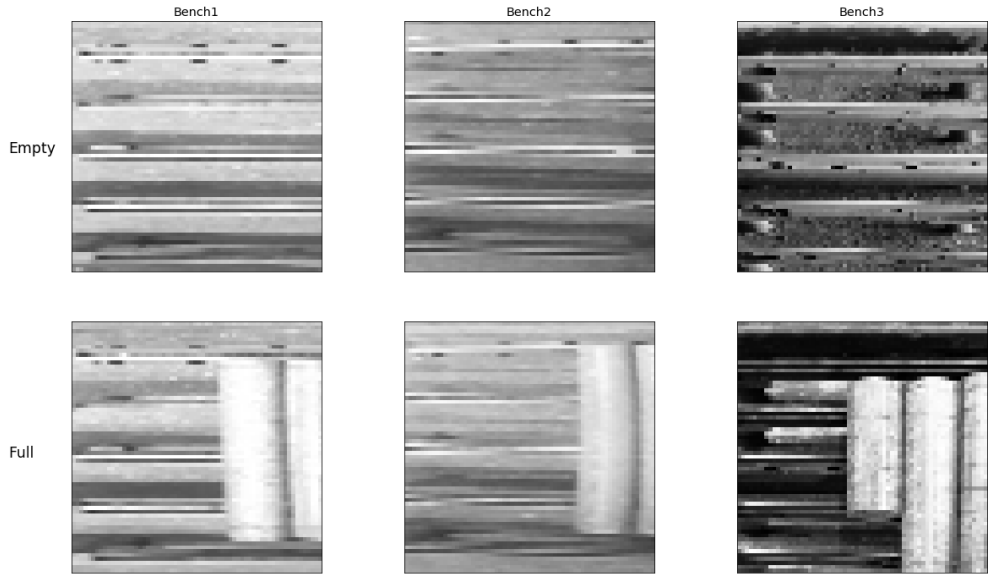


Figure 1.4: Example of input images.

1.1.4 Network

The legacy system uses a Convolutional Neural Network composed of two convolutional blocks necessary for the extraction of the image features: each block is composed of one convolutional layer and one max polling layer. After the convolutional blocks, there is one fully connected layer that performs the image classification. The first convolutional layer has twenty filters with shape 5x5, no padding and an unitary stripe. The second one has the same configuration as the first one with a higher number of filters: forty. Both the convolutional layers use RELU activation function. Max pooling layers have one filter with shape 3x3 and with an unitary stripe.

During the training phase is present also a dropout layer with a rate of 50% that helps to reduce the over-fitting and deploying the knowledge along the entire network.

Accuracy	Precision	Recall
99.80%	99.50%	100.00%

Table 1.2: Nominal metrics.

Table 1.2 shows the nominal metrics of the network obtained with the test data-set in nominal condition.

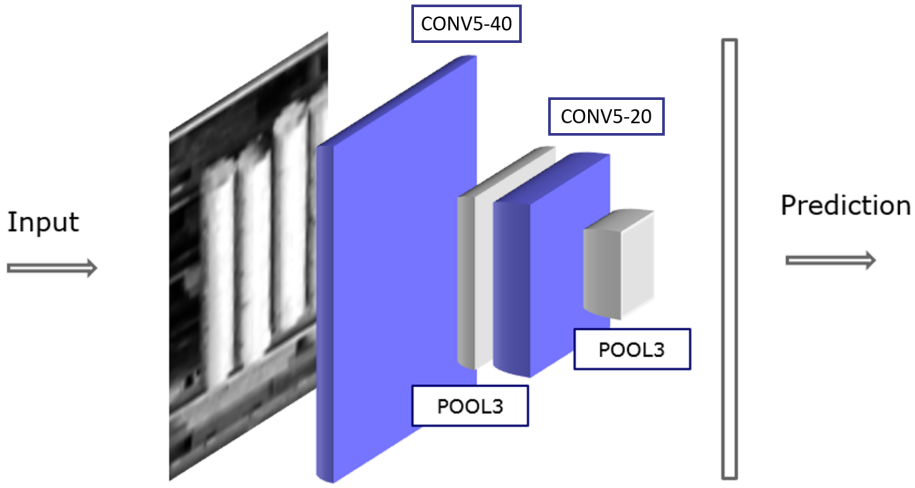


Figure 1.5: Architecture of legacy CNN.

1.2 Conclusion

The previous sections introduced the reader to basic aspects of machine learning, and how it can help solving industrial problems. Moreover they described the case study in terms of context, available data and desired results.

The next chapter will deepen the basic theoretical concepts of Convolutional Neural Networks which are a particular type of machine learning models specialized to solve computer vision problems.

Chapter 2

Theoretical Background

2.1 Introduction

This chapter provides to the reader some foundations about neural networks and in particular on Convolutional Neural Networks, which are state of the art models for image classification. Along the chapter we will present the mathematical structure of generic neural networks and we will describe how these models can be trained to solve specific tasks; once the reader is familiar with basic concepts we will provide intuitions on how Convolutional Neural Networks work.

2.2 Neural Networks

Neural networks are one of the machine learning tools used to solve countless different challenges. They are born in the 40s but only in the last decade, they became very used and famous in Artificial Intelligence due the massive used of GPUs. In fact, GPUs allows to speed up the training phase by about a factor 5 (benchmarks from [8]) thanks to its capabilities in matrix operations. They were inspired from the human brain: as a human brain, neural networks are composed from neurons, the atomic part of neural networks, grouped in layers. Each neuron performs a non-linear function on a weighted input: the inputs linearly combined with the weights W and a bias b, then the result is taking as input from a non-linear function (2.1).

$$z = f(WX + b) \tag{2.1}$$

The non-linear function is the main part of the neurons: without applying a non-linear function the result of each neuron is proportional to the input and as well the result of the network; in this way, the networks would be able to solve only a very small branch of problems.

There are many functions can be used in machine learning models, in particular, the most commons are ReLU, sigmoid, softmax or tanh chosen based on the type of problem. For example, softmax is very used in classification tasks.

Neural networks are composed of three different types of layers:

- **Input layer**: it's the first layer. It isn't usually considered as a real layer, it gathers the input from the outside system and it provides it to the next layer.
- **Output layer**: it is the last layer of the network, it gathers the data from the previous layer and it generates the output of the network.
- **Hidden layers**: they are the layers between the input and the output one. In the most simple network, no hidden layers are present; the number of hidden layers defines the type of the network.

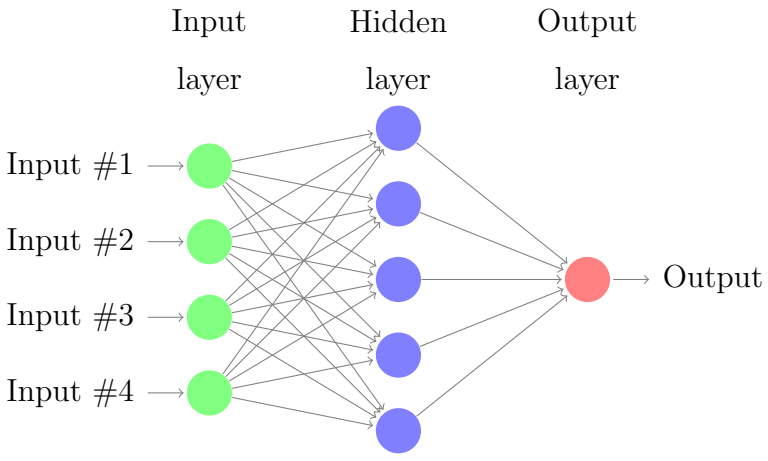


Figure 2.1: Example of a neural network.

The simplest network is the logistic one; logistic networks have the characteristic to not have hidden layers. These networks are used for their simplicity and for their

efficiency both the training and inference phases. The networks are usually very small and they can be stored in embedded systems with low computation power. Of course, their simplicity is also their weakness: they achieve weak results in complex challenges. In order to solve more complex tasks the state of the art defines deep learning: it is a particular technique where we add one or more hidden layers to the network. Each hidden layer elaborates the data gathering from the previous layer and it provides the results to the next one. This type of neural networks suffers the problem of the parameters explosion: every neuron of a layer is linked to all the neurons of the following layer and so on until the final output layer. Given two layers i and $i+1$ with respectively $\mathrm{N_i}$ and $\mathrm{N_{i+1}}$ number of neurons, the layer *i* has a number of parameters pair to $N_{\mathrm{i}}N_{\mathrm{i+1}} + N_{\mathrm{i}}$; we can simply image what happens to the overall number of parameters adding several hidden layers. To overcome this issue the researches define particular models for specific problems: for example for the image classification, it is used the convolutional neural networks capable to extract the features of the images, for speech recognizer we use the recurrent neural networks capable to use past data for a single prediction and so on. All the models are developed with the task to limit the number of parameters and of course to solve the specific challenge.

2.2.1 Training

The machine learning models have the particularity to learn from the data, so the programmer has not to explicitly describe the behavior of the model. Neural networks learn their behavior during the training phase: during this phase, the network tries to inference many examples taking in input and to fix its parameters comparing the predicted output with the actual ones obtained by an oracle. State of the art defines three different types of training:

- **Supervised learning**: the data-set contains the data and the relative labels that work as oracles. During the training, the network generates an output for each input and it tries to correct the wrong ones minimizing a cost function. This method is used in classification and regression tasks.

- **Unsupervised learning**: the data-set does not contain the labels and the network

has to deduce them from the data. It is used for clustering, dimensional reduction or association rules tasks.

- **Semi-supervised learning**: it is a mix of previous methods.

The training phase is performed with back-propagation method: it's a technique that fixes the weights minimizing a cost function. The cost function is non-linear and it represents the distance between the actual label and the predicted one. Many different loss (or cost) functions exist, the choice of what type of loss using is due to the characteristic of the problem. In our case, we use binary cross-entropy because it is the state of the art for binary classification (see section 2.4.2).
Another important aspect of the training phase is the data-set. In chapter 4 we'll analyze the testing of the data-set, in this section, we want to define how the data-set should be. The state of the art of machine learning describes dividing the data-set into three different and independent groups: training, validation and test set. Figure 2.2 shows an example of the use of the training and validation set during the training phase.

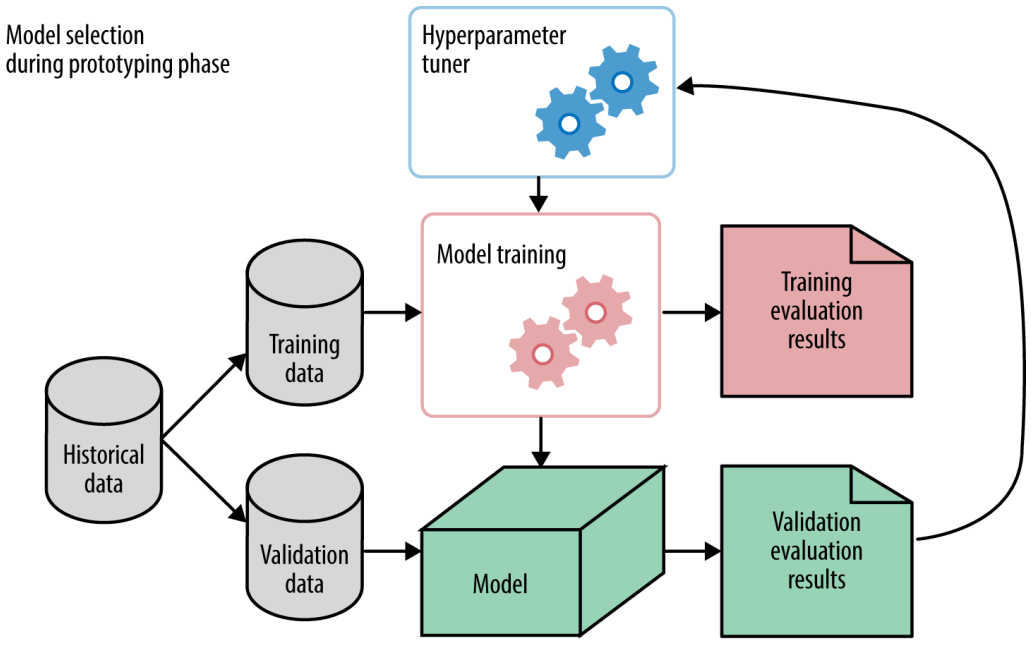


Figure 2.2: Example of the training phase [14].

All the sets should have the same proprieties and the same distribution of the original one and as well of the population. In general, the training set should be bigger than the other two, in this way the model has more information to define its behavior. The training

set is used during the training phase to update the network weights, the validation one is used to evaluate the capability of the network to generalize the problem and to define the optimum configuration of the hyper-parameters, while the test set is used to test and to calculate the nominal metrics of the final model. It is very important that the three groups are disjoint: this ensures to calculate some accurate metrics.

The training phase is an iterative operation divided into epochs: an *epoch* is a series of steps that is ended when all the data in the training set are used. Each operation uses a restrict number of data called *batch*. A big batch increases the speed of the training but it reduces the quality because it reduces the number of parameter upgrades. Each iteration performs a forward-propagation in order to generate the predicted labels for every data in the batch and then a back-propagation to upgrade the weights. The weights are minimized in based on the gradient, calculated on the loss function. The gradient measures the change of the loss function respect to a particular activation. The aim of the training phase is to minimize the loss function for the training set. Figure 2.3 shows the two steps of forward and back-propagation during an iteration in the training phase.

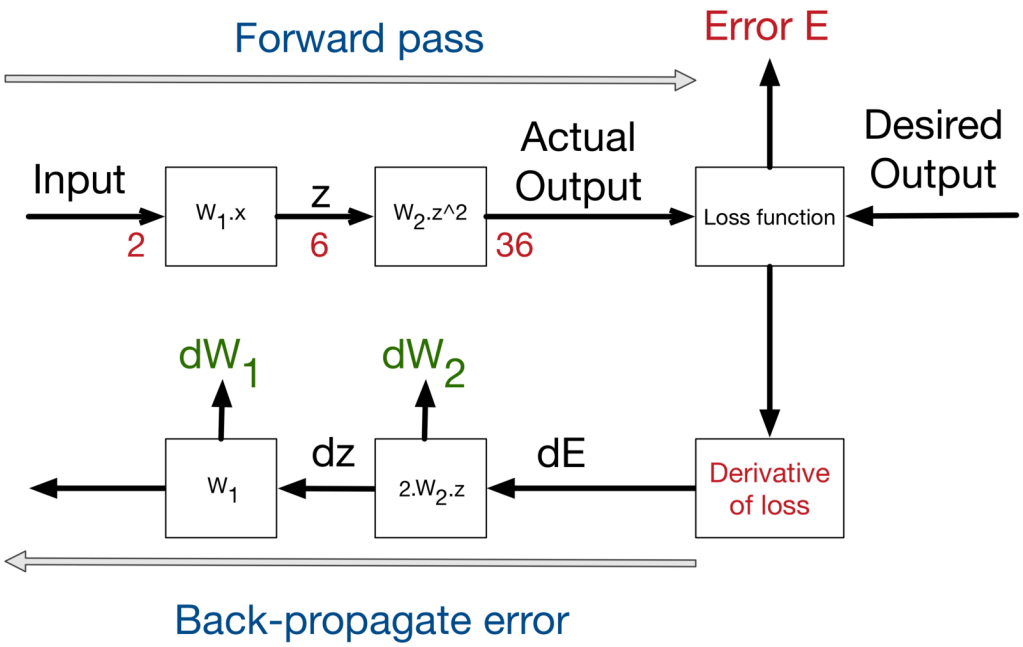


Figure 2.3: Example of back-propagation [9].

The training phase is one of the main steps in machine learning modeling because it defines the behavior of the model in the inference phase. The evolution of this phase is

defined from few but very important hyper-parameters:

- **Batch size**: we define the size of the portion of the training set to use in each weights upgrade.
- **Epoch**: number of iterations of the training phase. A great number of epochs should generate problems of over-fitting while a small number should generate a weak model.
- **Learning rate**: defines the modification rate for each epoch. A high value of learning rate speed-ups the training but it reduces the quality of the phase.

The development of a good model is not an exact science, but it is based on the state of the art of machine learning and on the experience of the researcher; so it is not possible to define the best hyper-parameters configuration uniquely.

2.2.2 Binary Cross-entropy

Binary cross-entropy is a loss function based on logarithmic used when the output of the model represent a probability. The function maps a real number into the probability space [0,1]. The function increases its value as the predicted label diverges from the actual one. The formula 2.2 represents the function expressed in analytic form.

$$Loss = -(y \log(p) + (1 - y) \log(1 - y)) \tag{2.2}$$

Where *p* represents the predicted output and *y* the actual one.

2.3 Convolutional Neural Networks

In the previous section, we analyzed the theory at the base of the neural network. In this section, we want to deepen to a particular type of networks: convolutional neural network or CNN. In adding to its capability of feature extraction, one of the reasons for the introduction of this type of network is the limitation of the number of parameters that allows an improvement of the efficiency and saving space. CNN introduces new types of layers:

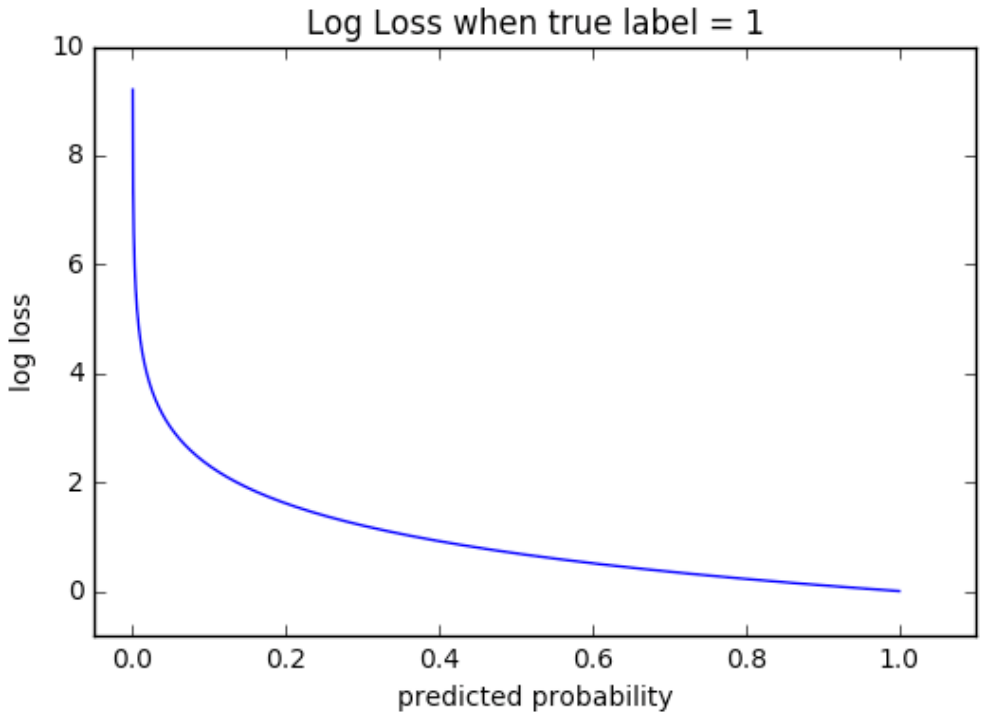


Figure 2.4: Binary cross-entropy where the true label is 1.

- **Convolutional layer**: series of filters used to extract specific features from the images.
- **Max-polling layer**: used to reduce the size of the image by applying a max operation. The operation is performed iteratively in small areas of the activation bounded from the filter.
- **Average-polling layer**: used to reduce the size of the image by applying an average operation. Like the max-polling layer also in this layer is defined a filter.
- **Deconvolutional layer**: particular layer used to inverse the operation of convolution.
- **Dropout layer**: it is a particular layer used to avoid the over-fitting and to distribute the knowledge along in all the network. It reduces the capability of the network to learn to discard some updates for each iteration. The user has to specify the rate of dropout.

The core part of CNN is obliviously the convolutional layer. This layer applies a

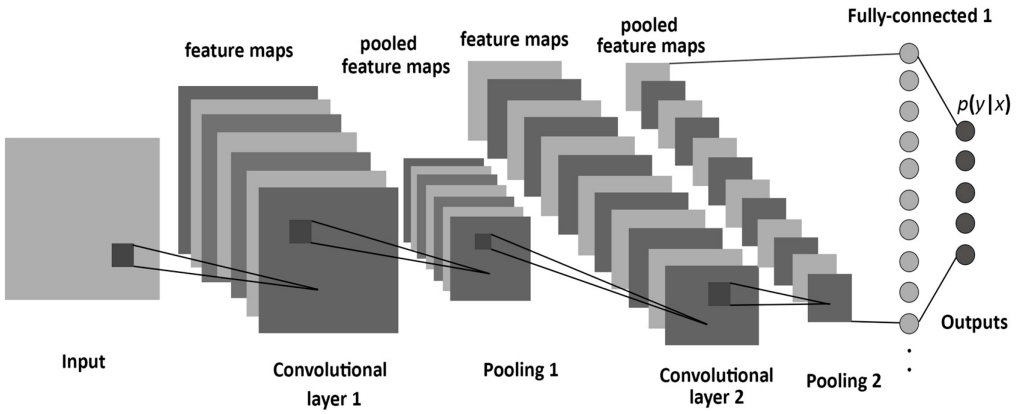


Figure 2.5: Example of CNN structure.

convolutional operation on the image with a series of filters. Each filter, trained during the training phase, is used to extract a specific feature: in the first layers the features are very simple, in the lasts they become complicated: the last convolutional layers aggregate the features from the previous layers in more complex activations. For example, the first layers are able to extract simple features like vertical or horizontal lines, the last ones are able to recognize parts of faces or other complex structures. The convolution transformation is a matrix operation where a sub-matrix with the same dimension of the filter is extracted from the input data, each element of the sub-matrix is multiplied with the corresponded element of the filter and all the results are summed in order to obtain an element of the new matrix, output of the convolution operation. Figure 2.6 shows an example.

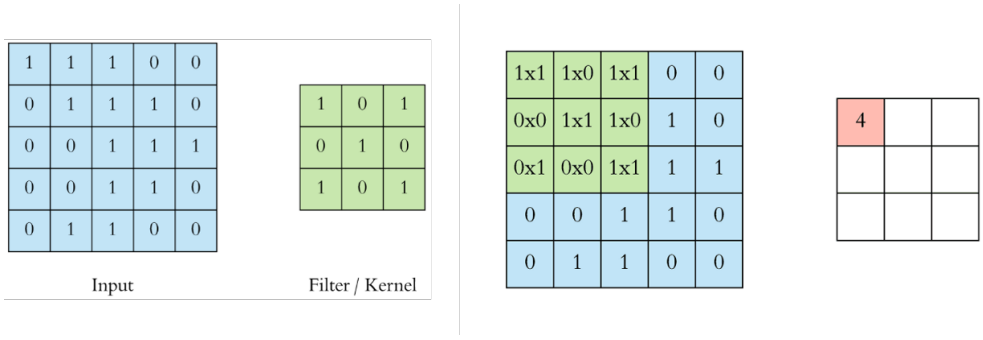


Figure 2.6: Example of a convolutional operation (courtesy from: [4]).

The convolution layers are defined by some parameters:

- **Activation function**: to each element of the output matrix is applied a non-linear

function specified from this field.

- **Stride**: specify the movement step between two following sub-matrix.
- **Padding**: extra rows and columns added to the input matrix to match the input size with the filter size and the parameters defined above. The padding can assume two value: *same* if the output matrix has the same dimension of the input or *valid* that means no padding.
- **Filter size**: the dimension of filters.

2.4 Conclusion

The chapter introduces the reader to basic knowledge on neural networks structure and how to train them to solve problems starting from data. As any machine learning models, data-set quality and variety has a strong impact on the model performance: in fact, if data from reality differs from the training data-set, the model can show an unexpected behavior. The next chapter will describe a novel approach to quantify how much a model is robust against unexpected samples, i.e. images with characteristics that strongly differ from one in data-set.

Chapter 3

Robustness

3.1 Introduction

This chapter presents a novel approach to quantify the robustness of a machine learning model against inputs that comes from a different domain than the nominal one used to generate the training set. This property is really relevant in industrial applications since the data acquisition from the field can present defects and deviations from the nominal data due to the impossibility to fully control the environment.

3.2 Definition

The focus of all the project is improving the safety of a legacy system through the analysis of the robustness of a convolutional neural network. State of the art of neural network does not define a simple definition of robustness, so before starting with the work it is necessary to provide this definition. In the great part of the work we consider the model as a black box, so also the robustness definition is black box. In this chapter, we will provide out definition of robustness for machine learning models.

Simply speaking we can think to the robustness as the property to maintain the behavior close to the nominal one with every input. As shown in the following chapter, we represent each test result with 3 graphs, showing the variation of accuracy, precision and recall of the increasing of the alteration intensity of a specific image feature (for example brightness).

For each alteration, we define an interval that represents the area under analysis for that test: of course, it is impossible to maintain a behavior close to nominal one in every situation and in particular in high perturbation cases. In this interval, we want to maintain an accuracy over a reference line that defines the acceptable bound for the metric. Our reference is an accuracy of 90%: a lower accuracy can not be considered acceptable.
We compute the area included between the accuracy limit and the alteration graph to obtain a significant value used to compare different networks in the same situation. Figure 3.1 shows an example of the calculus.

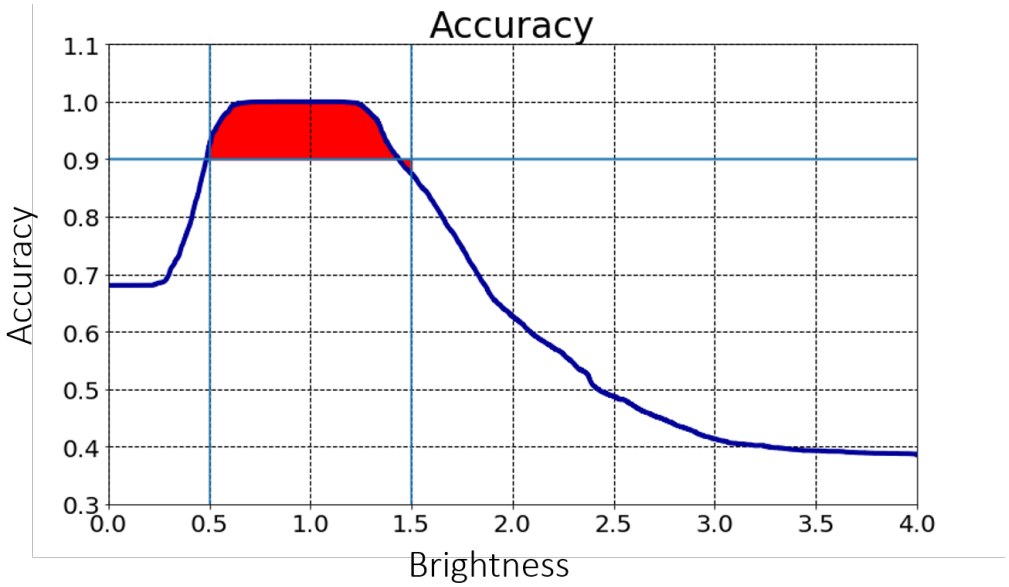


Figure 3.1: Example of the significant area. The red area is calculated with an integral.

It makes sense thinking the alteration values close to the nominal value are more probable than the far ones. In order to better represent the probability of the alteration intensity, we define a distribution that favors values close to the nominal case. We weight each accuracy value with its relative probability. Unfortunately, we do not have time series of the alterations and we can not evaluate the right distribution so we decided to use two simple probability distributions:

- **Linear**: the slope is automatically defined from the data.
- **Exponential**: we have to define the α parameter representing the slope; again we

don't have any data from the past, so we defined it based on the importance of test for us and our experience.

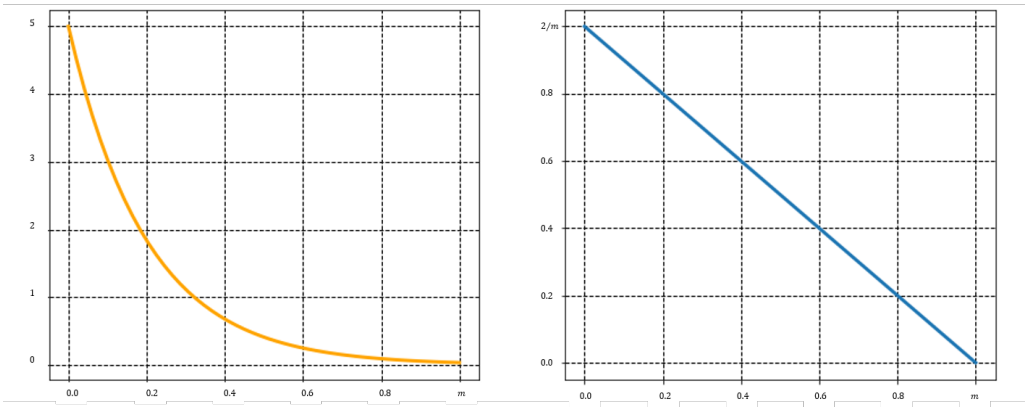


Figure 3.2: Linear (in blue) and exponential (in orange) distributions.

The area is calculated through a definite integral of the accuracy values weighted on the probability (3.1).

$$\int_{n}^{m} max(a(x) - 0.9, 0)p(x)\,\mathrm{d}x. \tag{3.1}$$

Where *m* represents the maximum alteration and *n* the nominal case, *p(x)* the probability for the specific perturbation value and *a(x)* the accuracy value in *x*. If the nominal case is inside the perturbation interval, the calculus will be the sum of two integrals. For example in the brightness case, it is possible both increasing and decreasing the value of this feature; in this case, the area will be the sum of the integral on the decreasing area and the increasing one. The area represents the robustness of the network for a given test case. Figure 3.1 shows the brightness example: the nominal case n is represented from the value 1, while the maximum alteration m is 0.5 for the left integral and 1.5 for the right one.

The next step is to summarize the different results in a single number representing the robustness for the network in all the situations. We can't sum or mean directly the values because each area value depends also from the considered interval: some intervals are significantly bigger than others. So we relate all the integral results with the maximum area obtaining for the respective test, in general referred to 100% of accuracy. Now we can mean all the previous results and obtaining a percentage that represents the quality of our

network: 0% means the network metrics are never over the reference lines while 100% means the model is optimum. Meaning all the result in a single real number maintains the possibility to compare different networks.
In case we consider some tests more important than the others, we can perform a weighted mean. It is also possible modifying the weight of the tests and analyzing how the robustness varies: in our case, for example, we consider the most important scenarios the perturbation of the brightness and the rotation but we are not able to define two specific weights for them. So, we defined the weights for the other tests and then we varied only the importance of brightness and rotation.

Alteration	**Weight** in %
Blur	10
Occlusion 1x1	5
Occlusion 2x2	5
Occlusion 4x4	10
Zoom	5
Sensor defect	10
Translation	5
Perspective	10

Table 3.1: Weights used in robustness analysis.

Table 3.1 shows the weights used for the robustness calculus. The remained 40% is devoted to brightness and rotation. Overall, the alterations presented in the table weight for the 60%.

3.3 Conclusion

The chapter described a novel metric to measure the robustness of a generic machine learning model. The method evaluates the behavior of any machine learning models on the variations of the work environment. The next chapter will present a testing framework for machine learning models that implements this novel metric.

Chapter 4

Test CNN Robustness

4.1 Introduction

The following chapter describes a testing framework for a generic machine learning model. For industrial settings, evaluating the model safety is a main task. As the behavior of machine learning models is influenced from the data quality, and there are no explicit coded rules, the methodology uses a black box approach. The framework analyzes the model behavior working outside of nominal conditions nominal conditions using perturbed data as input perturbed data and measuring the robustness from the outputs.
All the tests were designed to reproduce situations that could happen during the crane operations: i.e. modification in the brightness, little movements in perspective or partially occlusion of the camera view.

4.2 State of the Art

Software testing is that branch of information technology where the researchers join the state of the art of the field and their experience to improve their confidence to a software. Like the traditional testing, also in the artificial intelligence world, it is impossible to perform an exhaustive test on all the parts of the systems; in addition to it, in this particular field, there is not a solid state of the art in testing, but just few guidelines in aid to the tester. [6] describes a checklist that should be performed to test a machine learning model. The work defines a series of tests that should be performed to the model and to the data-set

during all the development phase. We can group all the step in three different classes according to the target of the test:

- **Data-set test**: it checks if the data-set is correctly generated in order to achieve the wished behavior. Data define the model behavior so a bad data-set has a big involve to all the system. In this phase, the tester has to check the correctness of the data labels, their relevance on the problem and the size of the data-set: a small set generates a poor model while too much information increases the needed time for the training. It is very important to verify that the distribution of the data in the data-set and their variability are representative of the population.

- **Test-set check**: it checks if the test data-set is well-formed. This ensures the metrics we got from the training phase are consistent. In this phase, the tester has to check if the distribution of the data in the data-set is the same of the population, if the batch can represent the population and if it does not contain elements of the training or of the validation set. These checks ensure to have consistent results from the tests.

- **Functionality test**: in general it is a black box test that evaluates the behavior of the network and if all the features of the data-set are learnt from the network.

An important aspect of testing is the meaning of the results: in other words what the variations of the metrics represent for our system. For example in our case results with a high number of false negatives impacts on the safety, critic part of the system, in the other hand high number of false positives impact on the efficiency. These considerations allow to classify the bugs in different categories of priority based on the impact to the system and to fix the most significant ones.

Other works focused on applying little alterations to the input of the CNN in order to try to generate a misclassification from the network. In the work [13] is explained a method to find the principal points of the image called SIFT points. These points have the characteristic to not change or in a very small way in the modification of perspective, size or scale. The authors implemented an algorithm that it is able to find SIFT points and to generate an adversarial example from them. The alteration phase is performed in two phases: in the first step a possible alteration is chosen and then another algorithm chooses the perturbation value. The alteration is performed inside a bounded space in order to

obtain adversarial examples closed to the original images. Figure 4.1 shows an example of the work: few white points overlapped to the image change the classification from a traffic light to in an oven.

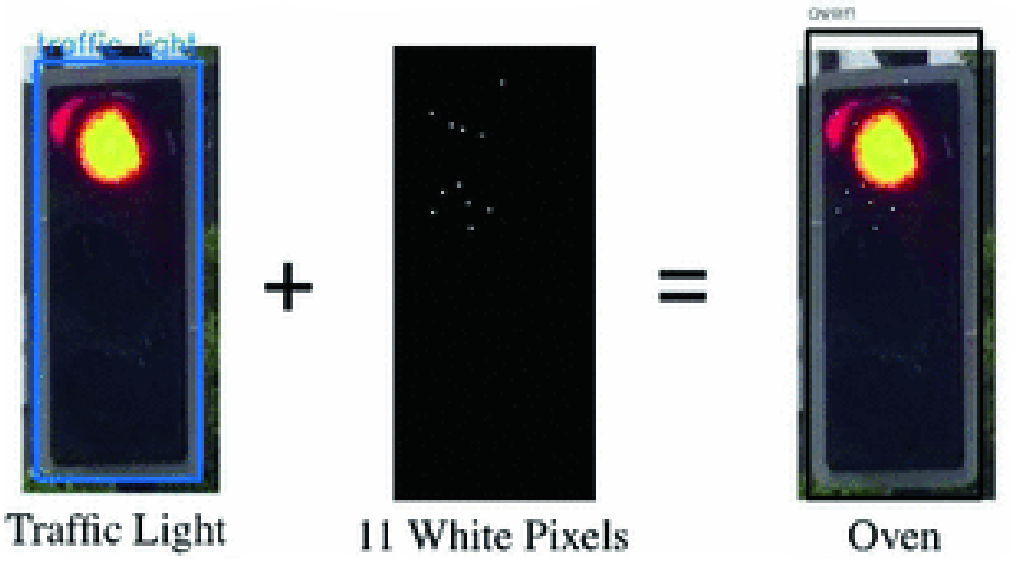


Figure 4.1: Example of misclassification from [13].

4.3 Test Framework

In the previous section, we analyzed some studies about the neural network testing. Starting from the state of the art we created our robustness testing framework.
In general, it is more complicated reproduce accurately the new conditions in the reality, in particular in industrial settings due to the elevated cost and elevated needed quantity of time. It is common use reproducing artificially the perturbed conditions, solution adopted also in this work.
Our testing framework is based on a black box approach: we simple generated some altered images and provided them to the network. The results are expressed through graphs that represent the evolution of the accuracy, precision and recall based on the alteration value.

The framework consists of a list of actions that the user has to follow in order to obtain a good testing phase. We already define which metrics we will use to compare the test

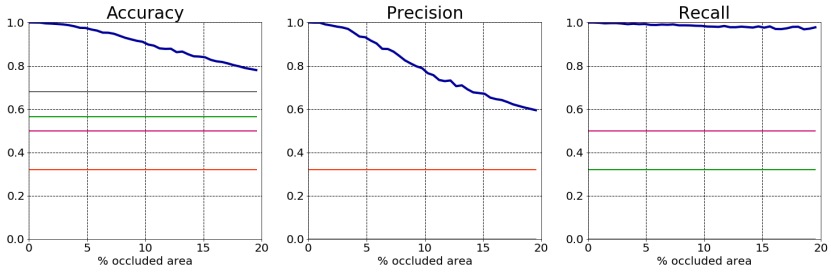


Figure 4.2: Example of a test result.

outputs, so the first step is to define the nominal behavior used as a landmark; of course, we can forecast that the perturbed images tests achieve weaker performance. We consider the nominal behavior the results obtained using the test-set; we used the complete data-set to generate and to perform the altered images tests.

In order to have more information from the tests and to help ourselves in the analysis, we defined 4 different benchmarks representing low-mark behaviors. The benchmarks are completely random and they use 4 different probabilistic distributions:

- **Uniform**: 50% probability to get an empty or a full classification.
- **Bayesian**: this distribution follows the proportion of empty and full images in the population. In our data-set about the 68% of the images are empty and 32% are full.
- **0-distribution**: in this case, the estimated prediction is always 0.
- **1-distribution**: in this case, the estimated prediction is always 1.

For each distribution we calculated the probability to get a positive, negative, false positive or false negative output: we compute a confusion matrix for each distribution and from that, we can easily compute the accuracy, precision and recall metrics. Table 4.1 summaries the metrics for each benchmark calculated in an analytic way.

We assigned a color for each benchmark. Figure 4.3 shows the legend valid for the entire project.

The next step is to decide which alterations perform on the images and in which range. Of course, it is important applying significant alterations in order to obtain consistent results. In our case, we performed two different types of tests: one could be applied to

cropped images and the other to complete images. We applied alterations of brightness, occlusion with different sizes of square and colors, blur and sensor defects (for example the presence of green lines) directly to the cropped images. The tests on complete images concern alteration on the view: translation (both horizontal and vertical), rotation, perspective transformation and zoom.

Benchmark	**Accuracy**	**Precision**	**Recall**
Uniform	50.0%	32.0%	50.0%
Bayesian	56.4%	32.0%	32.0%
0-distribution	68.0%	32.0%	0.0%
1-distribution	32.0%	32.0%	100.0%

Table 4.1: Benchmark metrics.

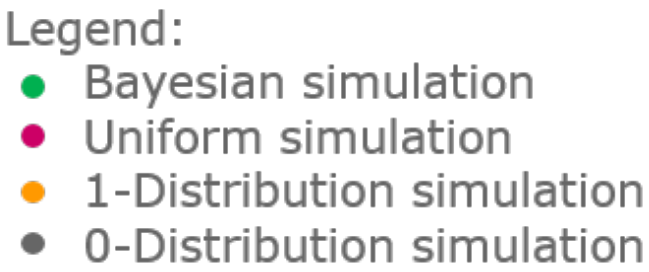


Figure 4.3: Graphs legend.

An important step is to define the range of the alterations. It's a tricky task: we want to cover the major part of the cases with the minimum effort. For each alteration is necessary to select a range capable to generate significant conditions (for example an image with the view outside of the zone of interest is useless) and choosing a step to scan the interval in a reasonable time. Finally, we could apply the transformations to the data-set and generating the new images.

The final phase of testing is the comparison of the results with the nominal behavior and the benchmarks. This phase allows us to determinate the robustness of our model in conditions not forecast during the development and the training phase and to highlight the

critic situations. In the next chapter, we will discuss how to overcome some of these situations, in particular for blur, brightness and rotation perturbations.
We can summarize the framework as a recipe:

1. Define the concept of robustness.
2. Define the metrics and eventually the benchmarks we want to use.
3. Select what alterations we want to reproduce.
4. Select for each alteration the range of modification.
5. Perform the tests and gather the results.
6. Analyze the results and highlight the critic situations.

4.4 Tests

In this section, we report all the test performed during the work and in particular, we will analyze a little better the test of brightness, blur and rotation.
In the previous sections we introduced our work method: we will follow the same procedure for every test and we performed the robustness analysis as explained in the theory chapter. In the next chapter, we will use a state art method in order to increase the robustness of the network for specific conditions and we will introduce a possible alternative in case of absence of the original training data-set.

4.4.1 Blur

The Blur filter is a Gaussian alteration that modifies the sharpness and the scale of an image. This modification is often used in image processing to reduce the noise of a picture, but at the same time, it also reduces the details of the image. The filter removes the high-frequency component and it is very similar to a low-pass filter; simply speaking the Blur filter for every pixel applies a Gaussian function including a defined neighboured specified from the round value: a great value of round means a low level of details in the

entire image. The alteration simulates the lost of the focus of the camera. The transformation is symmetric so it's not necessary to perform the perturbation for negative values. We chose to apply the modification in the range between 0 (original image) and 5 value of round with a step of 0.1.

The graph shows that Blur alteration has not particularly effective on the accuracy of the network also for elevated values of round: in general the accuracy remains around 90%. In chapter 6 we will explore a little bit the focus of the network in classification task: for now, we can think the network recognizes the transition from dark to clear color as a billet; from the example images, we can see that also with high alteration values this characteristic is maintained. For this test, the main result is the reduction of the precision metric. This involves in more false positive and a less efficiency of our system. Figure 4.5 summaries the result of the test.

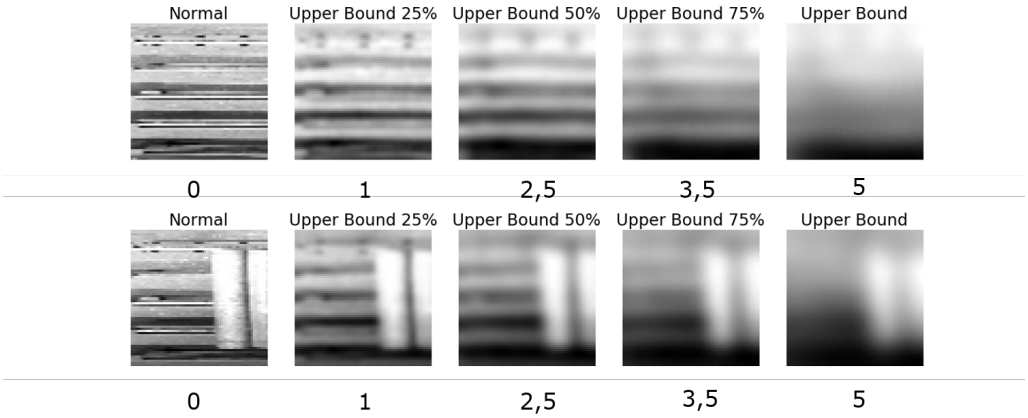


Figure 4.4: Example of Blur altered images.

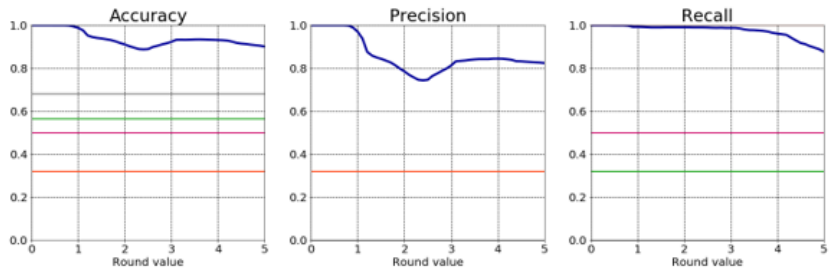


Figure 4.5: Results of Blur test.

4.4.2 Rotation

Rotation alteration simulates little rotations of the camera view around its center. The alteration is applied on the complete images before they are cropped and the perspective fixed. We couldn't apply the modification to the cropped images because the movement would generate black portions in the image. We rotated the camera view between an angle of $-10°$ and $10°$ with a step of $1°$.

Figure 4.7 shows how the alteration impacts on the network metrics. The accuracy decreases its values sensible also under 80%, unacceptable for our target. The network maintains our standard (accuracy over 90%) only in the range $-6°$ and $6°$ with a little exception near the $-4°$. We can also notice how the positive variations are more effective on the metrics respect the negative ones. Both the precision metric and recall one decrease their values involving in problems in safety and efficiency of the system.

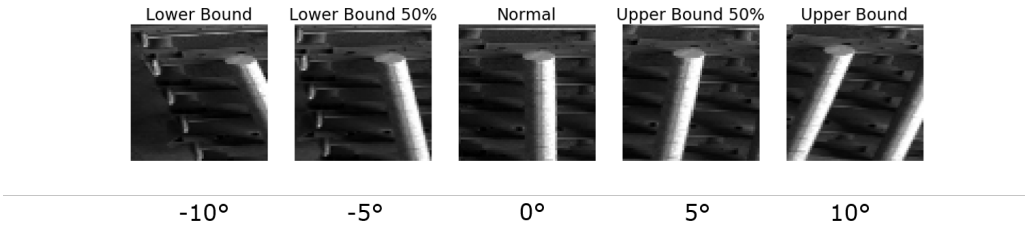


Figure 4.6: Example of rotated images.

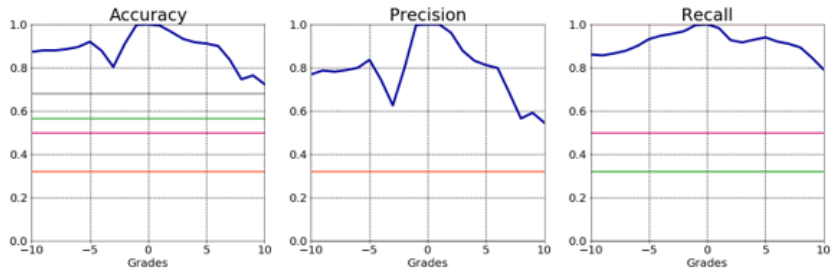


Figure 4.7: Example of rotation altered graph.

4.4.3 Brightness

Brightness is another important alteration to consider for us. The benches are positioned in three different places with different lighting and the network has to manage all the

situations. In addition to it, we have to consider the different lights along the day and the year. The test alters the brightness in the range 0-5: the values represent the percentage of brightness in the image relate on the original one; so 1 it is the original image, 0 it is a black image, 2 it is an image with double brightness and soon on. Figure 4.8 shows the scale of perturbation performed in this test. Under the example images, we showed five histograms that represent the distribution of the color values of the above images. We can see how the black image has all the pixels centered to 0 while the white image has the concentration around the 1.

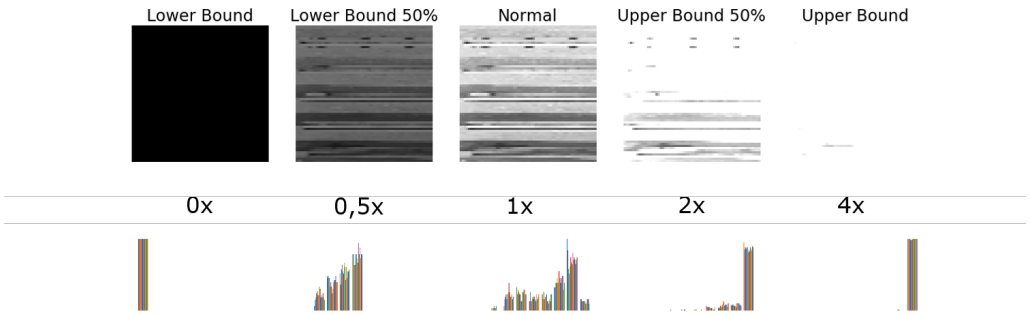


Figure 4.8: Example of brightness alteration.

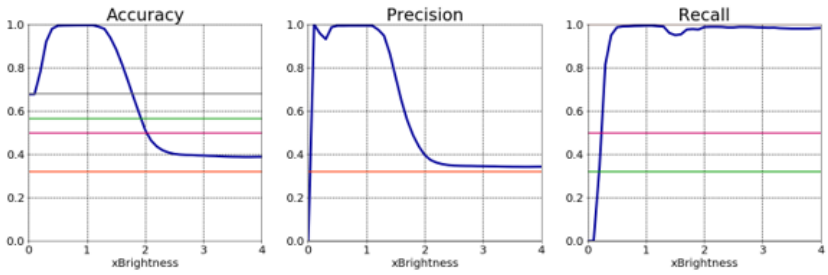


Figure 4.9: Example of brightness altered graph.

Figure 4.9 shows the results obtained in the brightness tests. We can see how the brightness has big involved in the network behavior: in both cases with low and high values, the performance is very bad. Figure 4.10 shows a range highlighted from the orange band where the alteration has no effect.

There are two particular cases shown in the graphs:

- With values of brightness closed to 0%, the network classifies all the images as *Empty*. We can see it from the accuracy to 68%, the same value of the percentage

of empty images.

- With values higher than 200% the network categorizes all images as *Full*.

These particular cases are very interesting and they have to be solved: in particular, the first scenario has a big involve in the safety of the network.

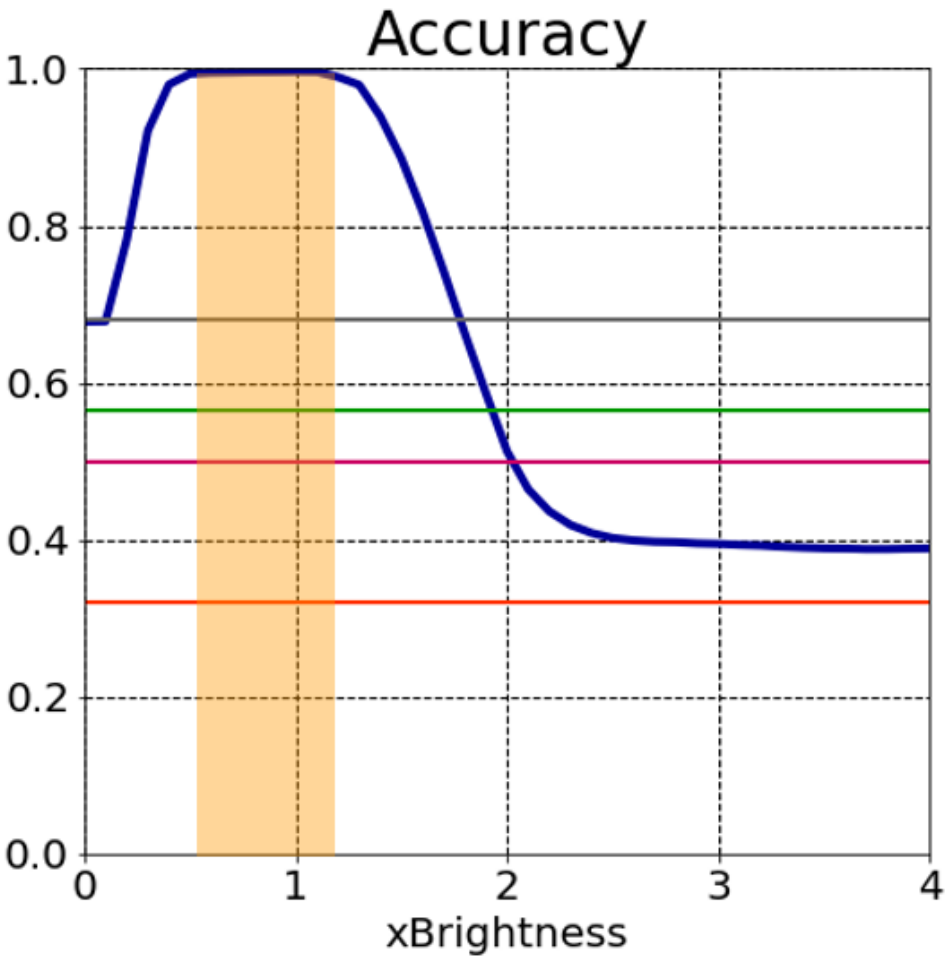


Figure 4.10: Brightness accuracy.

4.4.4 Other Alterations

In addition to the previous perturbations we performed also the following tests:

- **Sensor defects**: in this test we simulated the presence of defects in the camera view, in particular, the presence of green lines which are caused by errors in the

acquisition of the camera signal. We performed the test with 0, 1, 2, 3, 4 and 5 lines in random and not overlapped positions.

Figure 4.12 shows how the presence of green lines in the view can compromise the performance only with a high number of lines: we can see how with three lines the accuracy is closed to 90%.

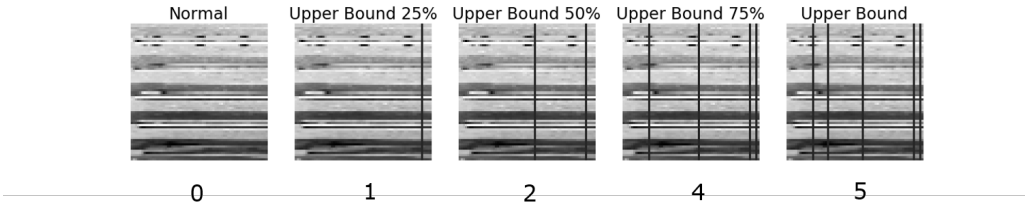


Figure 4.11: Example of green lines alteration.

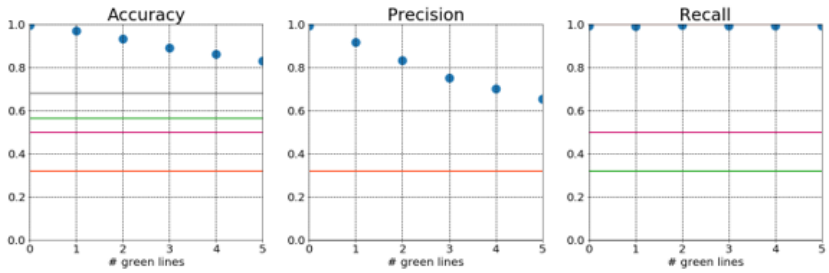


Figure 4.12: Results of green lines alteration test.

- **Zoom**: we simulate what happens if we crop the image with a different area: the new crop could be smaller or bigger than the original one. We explored the range between -10 and 10 with a step of 1: the values represent the numbers of rows and columns added or removed from the cropping area for each side. For example, the value 2 represents an erasing of 2 rows and 2 columns to the top of the image and to the bottom of the image. Figure 4.13 shows some examples with different values of zooms. The modification of the cropped area involves in a perturbation of the perspective of the final image. From the graphs of figure 4.14 we can notice how it is not a big problem having a smaller area of cropping, while a bigger cropping area has more involving to the network behavior: this effect is due to the presence of other billets in the safety part of the benches or to the presence of the building

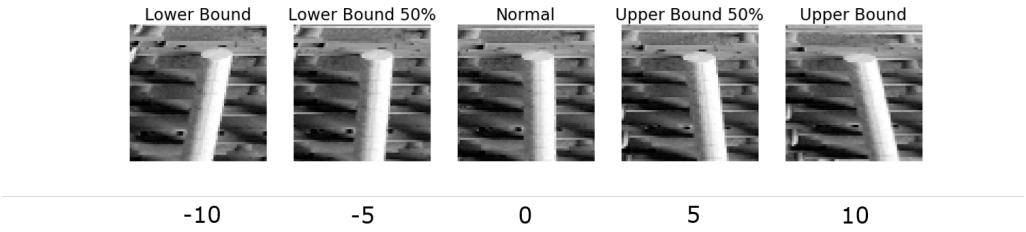


Figure 4.13: Example of zoom alteration.

structure in the image. In general, the alteration has not a big involve on the network behavior.

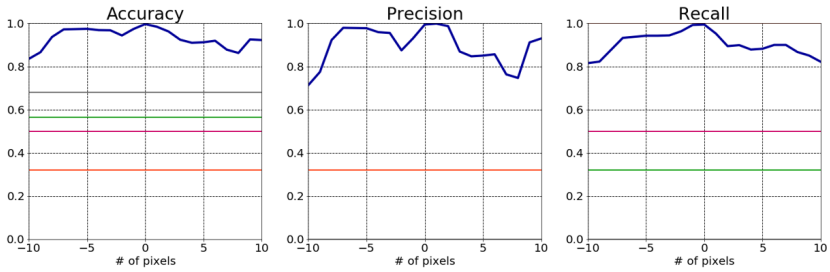


Figure 4.14: Result of zoom alteration.

- **Perspective**: the perspective alteration is a modification of the point of the camera view. In general, the alteration is used to map a 3-D object in a 2-D plan. Simply speaking the alteration transforms every rectangle to a trapezoid.
  We define the modification in the range 0 to 80 with a step of 5: the values represent the grade of the alterations and in particularly the Euclidean distance between the original top-right angle with the new top-right one. The alteration is performed both to the left and the right side. Figure 4.15 shows an example of the alteration with a perturbation applied on the right side, we let to the reader to image the specular alteration on the opposite side.

  The graphs show how the network is substantially affected by this alteration. There is not a considerable difference between the left and the right perturbation: both decrease the metric very fast, in adding to it in the left side we have a fluctuation of the precision metric.

- **Translation**: in this test, we modified the complete images translating for few pix-

els in vertical and horizontal direction. We performed the alteration between -20 and 20: the values represent the number of pixels of the movement, negative values are for movement to left and positive are for right. The same range has been used also for the vertical translation: in this case, negative values represent an upward movement while positive a downward one. Figure 4.18 shows an example of horizontal translation on the top and vertical on the bottom.

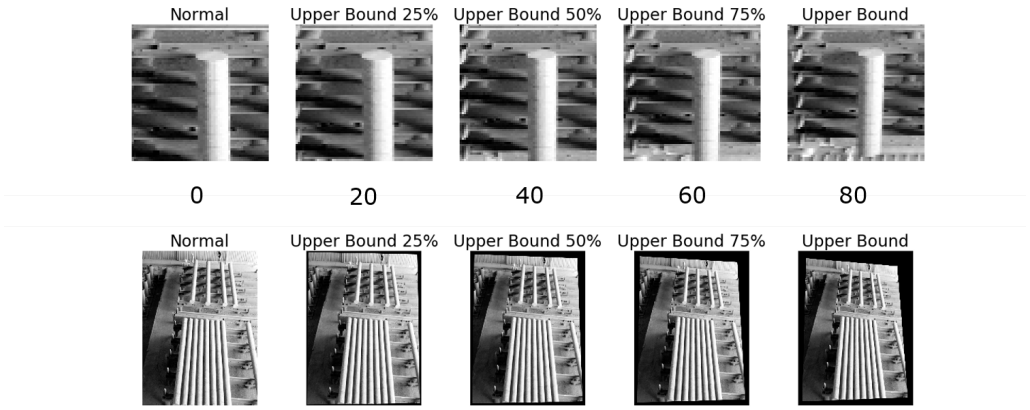


Figure 4.15: Example of perspective alteration to the right side.

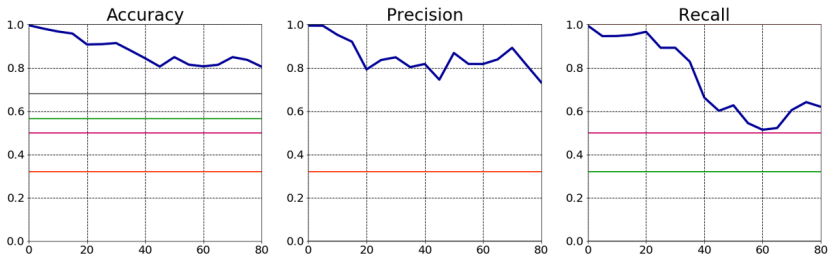


Figure 4.16: Result of perspective alteration to the right side.

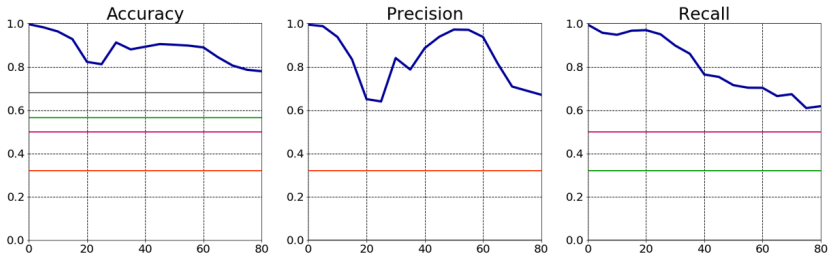


Figure 4.17: Result of perspective alteration to the left side.

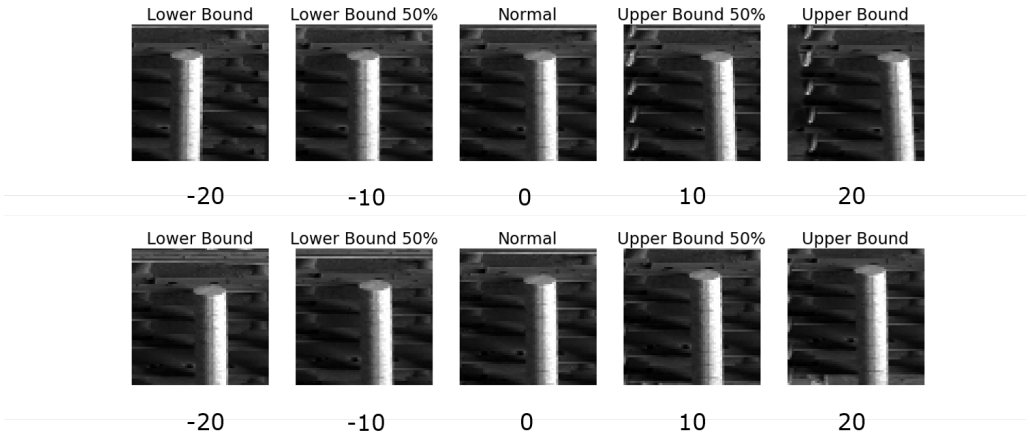


Figure 4.18: Translation examples: top horizontal and bottom vertical.

The two scenarios involve in two different results:

- Horizontal movement does not effect for little values while it decreases quickly the accuracy metric with values far to the 0. In particular, we have more problems with negative values: these values represent left movements and this can involve to include billets outside of the original focus area.

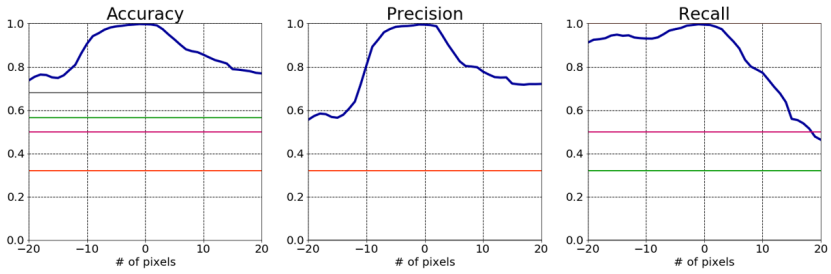


Figure 4.19: Result horizontal translation.

- Vertical translation has not a big impact on the metrics because in general, the billets are present along all the image and only particular cases are affected by this alteration.

- **Occlusion**: we simulate the presence of random occlusions on the camera view. We used squares with different sizes (1x1, 2x2 and 4x4) and different colors (black, white and Gaussian random gray). The tests are performed in range 0-20% (respectively 0-40% and 0-78%) of covered area. The square positions were selected

randomly minimizing the overlapping between the squares.

In general, occlusion has a big impact on the network behavior: black and white colors reduce the accuracy considerably also with the size 1x1 and 2x2, while the Gaussian one only with the 4x4. The 4x4 is a big problem with all the three colors. The Gaussian color has a little involved with the small sizes because the network recognizes variations between the light and dark parts of the images (the light parts represent the billets while the dark ones the empty space).

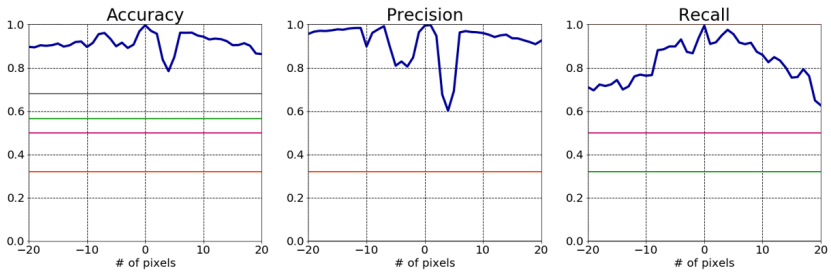


Figure 4.20: Result vertical translation.

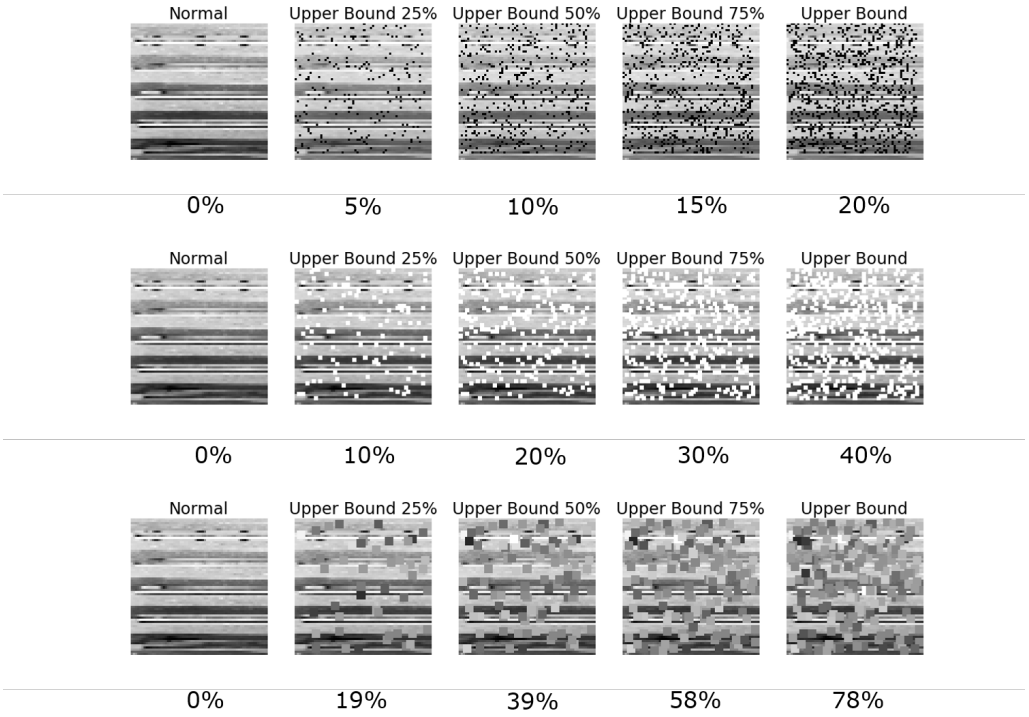
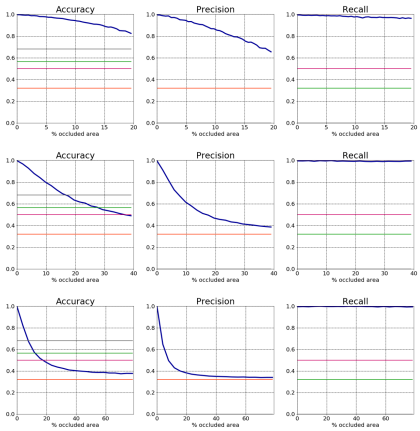
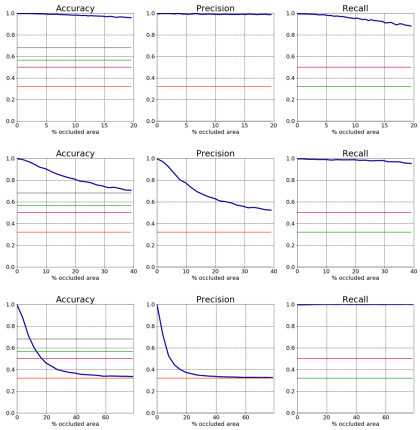


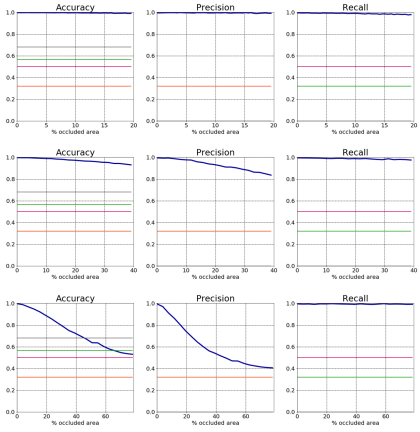
Figure 4.21: Example of occlusion alteration.



(a) Results of alterations of occlusion with black squares.



(b) Results of alterations of occlusion with white squares.



(c) Results of alterations of occlusion with Gaussian squares.

4.5 Robustness Analysis

In chapter 3, we formalized our concept of neural network robustness. In this section, we apply the formalization to evaluate the robustness of the legacy CNN. For each test, we chose a sub-range of values considered the most significant for this specific case. As explained in the theoretical section, we used two different distributions to weight the distance of the altered image from the nominal one, for the exponential distribution we chose to use an α value of 1.25 for all the tests except for brightness, rotation, occlusion and green line that we used a value of 0.5. It's useful to remind that a small α means a steep slope, so the values closed to the nominal case are very important in the robustness analysis. Table 4.2 summarizes the chosen interval and the results of the integration. We can notice the results are expressed in percentage: all the results are normalized to the max area (4.1).

$$Robustness_{\%} = \frac{area}{area_{\text{max}}} 100 \tag{4.1}$$

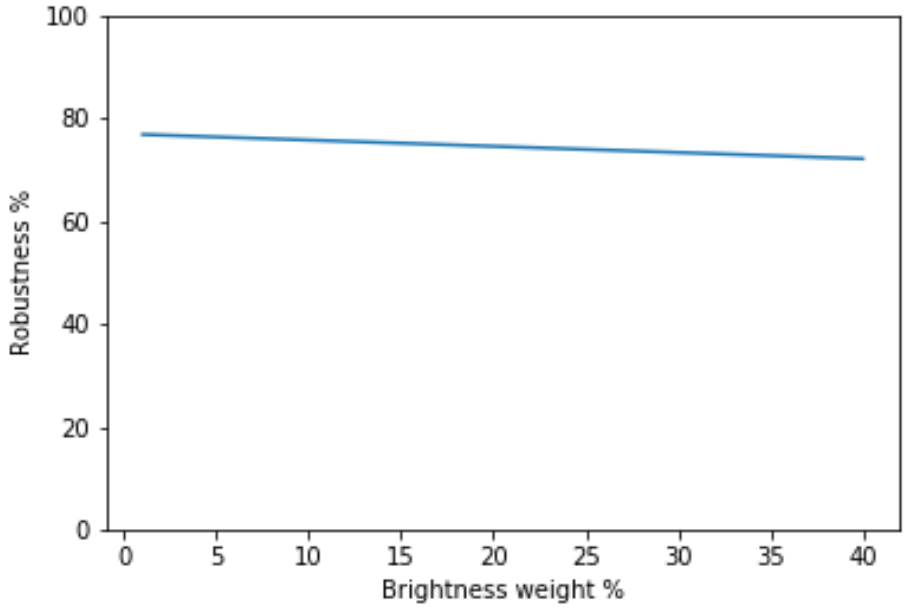
From the Table 4.2 we can notice that many tests got very good results, but some of those, in particular rotation and occlusion with a square size of 4x4, don't have a good robustness: in many parts, the accuracy is under the reference line fixed above. In the next chapter, we discuss how to improve the metrics in the cases of blur, brightness and rotation; we will not consider the occlusion because for our case is not a very common perturbation. Finally, we performed the analysis varying the weights of the brightness and the rotation tests. Figure 4.23 shows two bidimensional views of the analysis, one generated using the linear distribution and the other with the exponential one. Considering only two variable we can see how the better result of the brightness involves in the graph results: the optimal case is to not consider the rotation, situation not realistic. The robustness varies between 72% and 76% for the linear distribution and between 85% and 86% with the exponential one.

Alteration	**Interval**	**Robustness %**	
		Linear	**Exponential**
Brightness	[0.5, 2]	69.25	81.75
Occlusion 1x1	[0, 200]	99.76	99.91
Occlusion 2x2	[0, 50]	97.65	98.99
Occlusion 4x4	[0, 30]	47.65	56.53
Blur	[0, 2]	85.62	99.99
Green line	[0, 2]	87.09	88.64
Translation Horizontal	[-5, 5]	98.57	99.97
Translation Vertical	[-5, 5]	62.62	97.41
Rotation	[-5, 5]	57.18	78.79
Perspective Left	[0, 15]	88.76	97.08
Perspective Right	[0, 15]	95.27	98.65
Zoom	[-5, 5]	80.96	97.38

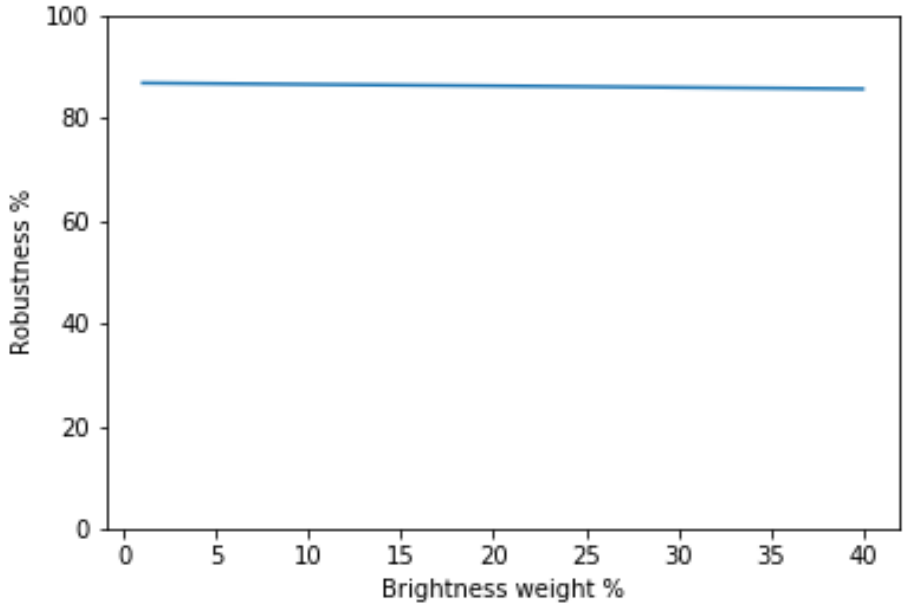
Table 4.2: Significant intervals and results.

4.6 Conclusion

The chapter described a novel black box framework able to perform various tests and to measure the robustness of a generic machine learning model. The methodology is generic enough that can be implemented in different settings, using a different set of tests or with a different type of model. In industrial setting safety represents a fundamental property, the next chapter will analyze two methods to improve the model robustness when it is working outside its nominal conditions.



(a) Generated with linear distribution.



(b) Generated with exponential distribution.

Figure 4.23: 2-D graphs of global robustness.

Chapter 5

Data Augmentation

5.1 Introduction

The previous chapter introduced a novel test framework for a generic machine learning model and we presented an example implementation for a convolutional neural network. This chapter will analyze two methods to improve the robustness of a convolutional neural network. The methods enrich the training data-set adding data of perturbed environment improving the training phase with non-nominal conditions. One method implements the state of the art of data augmentation while the other implements a novel method that enriches the model behavior using only the perturbed data.

5.2 State of the Art

Chapter 4 introduced us to the world of neural networks testing and it gives us some useful information about which tests are important performing. In particularly in the previous chapter, we spoke about the size and the variance of the data-set. The argument is crucial to obtain good performance: training a neural network means setting the parameters (weights and filters) to precise values that allow having a particular output with a given input. It is very important to balance the structure of the network based on the dimension of the data-set: if we have an ample number of parameters and a small input we probably fall into the problem of the over-fitting and the network will not be able to generalize the information learnt during the training phase for new data.

Data augmentation is a particular method used in machine learning to increase the size and the variance of a data-set. The method consists in few little alterations of the images like translation, rotation, brightness, crop and others applied to all the data-set. In general, it is recommended to not exceed the thirty images from one, otherwise the network specializes itself for the altered images. [2] explains in a simple way the problem and a technique for data augmentation.

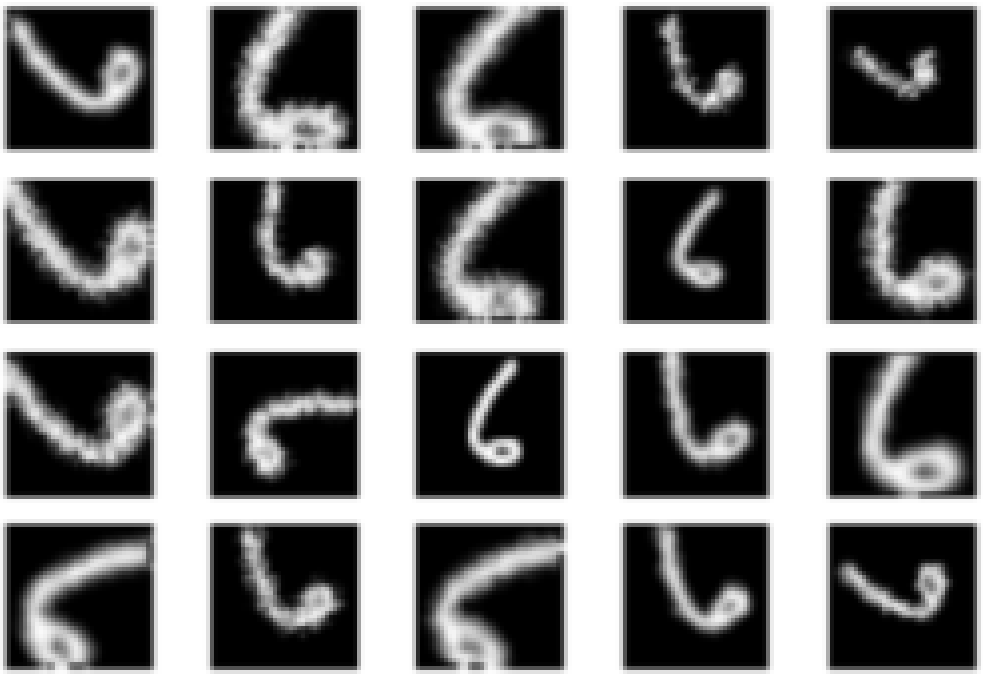


Figure 5.1: Example of data augmentation from [2].

Data augmentation requires a re-training phase each time we want to increase our data-set and, of course, it is necessary to have the original data-set. A different approach is incremental learning: it tries to increase the capabilities of the network without retraining all the network every time and without using the original data-set. [12] proposes a method with which is possible adding classes to an existent convolution neural network and so improving the domain of the network. The method requires the modification of network architecture. The new structure has two different branches: one is the frozen legacy network and the other is a new and not trained network. The authors train only the new branch of the model using only the new images: the output of the two branches are used to calculate a cross-entropy loss and to update the weights of the trainable branch. Figure

5.2 shows the structure proposed in the article.

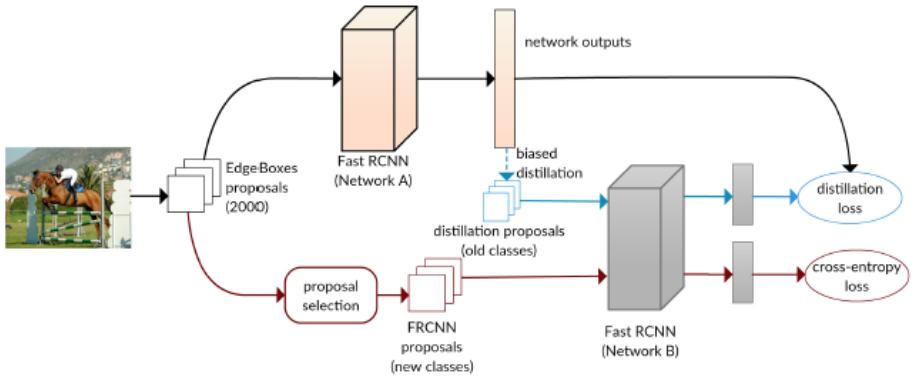


Figure 5.2: Improving learning architecture from [12].

We can think the new branch is trained to classify the new image types and it uses the support of the original branch for the old types. This method allows the researchers to modify an existent network adding new classes without retraining all the system: in general one of the most expensive phases.
In our case, we do not have to add a class but we need to enrich the training data-set of the one class presented in the model. In the following section, we will introduce three models with the intention of overcoming this issue.

5.3 Models

In this section, we present some models used as tentative to improve the robustness in particular conditions without using the original data-set. For relevance and visual reasons, we chose to consider blur, brightness and rotation alterations as examples. It is not possible retraining the original model without causing a catastrophic forgetting using only the new data-set. Machine learning state of the art defines catastrophic forgetting as the loss of the previous knowledge learnt during the original training phase. Retraining the model only with the altered data specializes the network only for the new images. To overcome this issue we added to the original model a new branch. We froze the old part and we trained only the new parts. The problem with this solution is how to combine the old CNN output with the new one. We propose three solutions:

- **Max layer**: in parallel to the CNN we added a copy of the model and we joined the two output in a max layer (Figure 5.3).

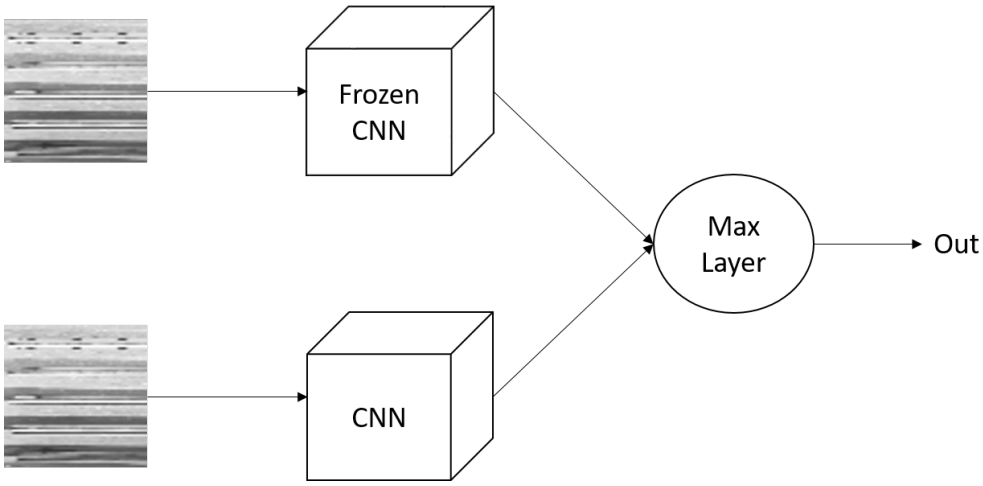


Figure 5.3: Max layer model.

- **Fully connected layer**: in parallel to the CNN we added a copy of the model and we joined the two output in a fully connected layer (Figure 5.4).

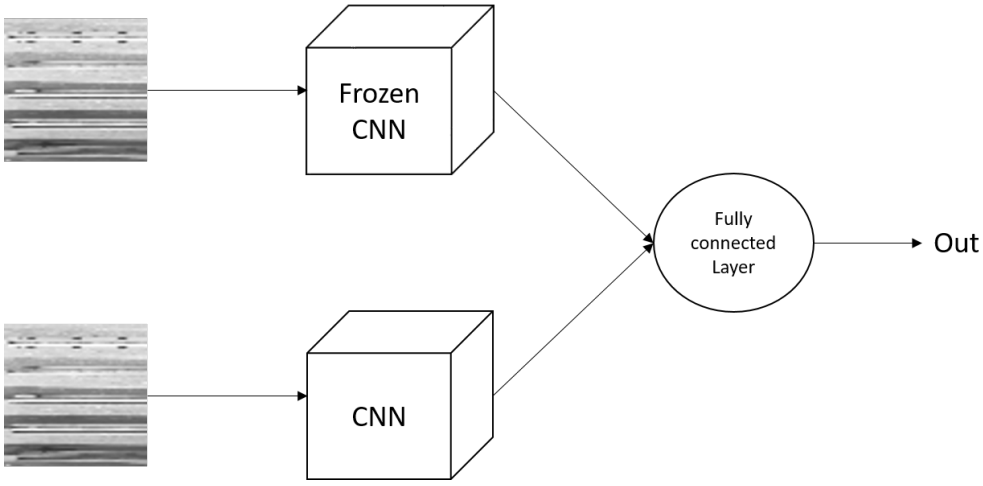


Figure 5.4: Fully connected layer model.

- **Prompter**: this is the most complicated solution, the output of the frozen CNN is used as an input of the first fully connected layer of the new CNN (Figure 5.5). The difference from the previous one is about the position of the information merging: in the first case the merging is applied at the end of the two networks and the two

outputs have the same importance; in this case, the output of the frozen network is joined to the output of the last convolutional layer of the new network: the frozen model has less influence to the entire system.

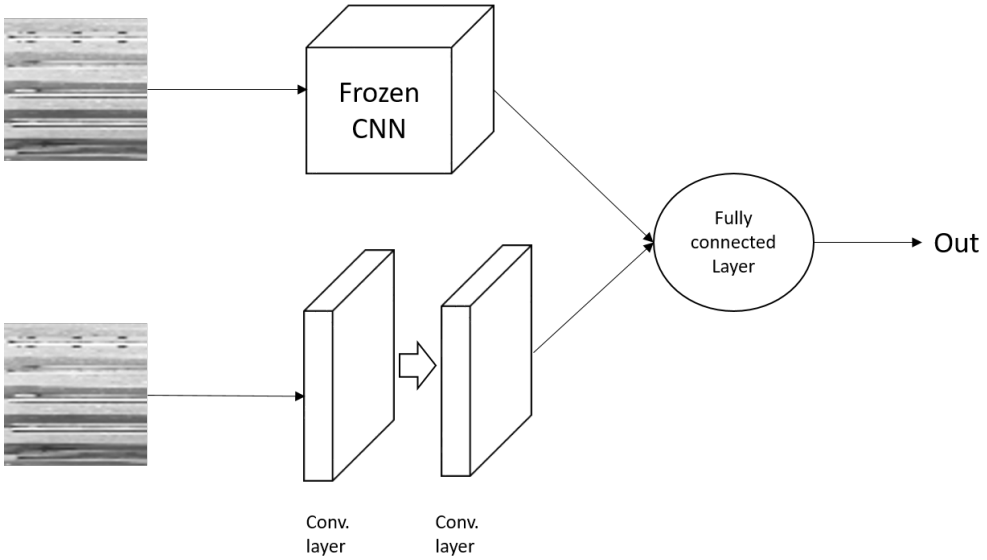


Figure 5.5: Prompter model.

Before starting training and performing tests, it is necessary to create the new data-sets. The image alterations are performed in the same way done during the testing phase; we just increase the steps in order to have a smaller data-set: it is important to avoid that the new network specializes itself only on the new inputs. For the blur alteration, we used a step of 1 between round values of [1, 5], for the brightness we used a step of 0.5 in interval [0.5, 2.5] (1 excluded) while for the rotation we used step of $1°$ in the range $[-2°$, $2°]$.

5.4 Results

In this section, we analyze the results obtained by applying the data augmentation method and the incremental learning one. The tests were performed with the framework introduced in chapter 4.

In order to analyze the result is important to perform two different tests: first of all, we have to perform the test in nominal condition, we want to be sure the new data-set does not

sensible decrease the nominal metrics, then we have to check the behavior in the altered conditions.

5.4.1 Blur Results

Table 5.1 summarizes the results of the test in nominal conditions using the models trained with the data-set enriched with blur images. The new data-set is composed of 30570 images altered with different grades of intensity with blur perturbation. The new data-set has the same distribution of original one: about the 68% of images are labeled as *empty* and the 32% as *full*.

Test	**Accuracy**	**Precision**	**Recall**
Nominal	99.80%	99.50%	100.00%
Enriched data-set	99.98%	100.00%	99.95%
Fully connected	100.00%	100.00%	100.00%
Max	99.90%	99.90%	99.80%
Prompter	99.92%	99.90%	99.85%

Table 5.1: Summary models trained with blur data-set in nominal conditions.

Table 5.1 shows us that the state of the art method and all the models used are possible candidates for our aim because their metrics are better than the nominal ones. The final step is to verify if the models improved the altered behavior. Figure 5.8 shows the results for every model under test.

From the graphs, we can notice how the enriched data-set method, the *fully connected* and the *prompter* models solved the problem. Interesting is the situation of the *max layer* model: in this case, we did not obtain any improvement for the problem. The network was not able to train itself during the training phase. Figure 5.6 shows the changing of the loss value in the training phase: the loss is stable around the value 0.3 for all the phase.

5.4.2 Rotation Results

Table 5.2 summarizes the output of the tests in nominal conditions using the models trained with the data-set enriched with rotated images.

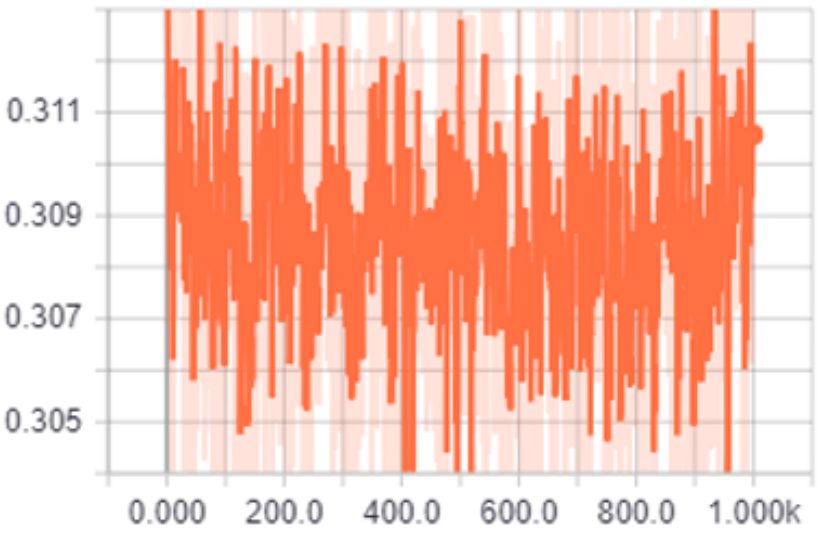


Figure 5.6: Graph loss function during the training phase of the max model.

Test	**Accuracy**	**Precision**	**Recall**
Nominal	99.80%	99.50%	100.00%
Enriched data-set	100.00%	100.00%	100.00%
Fully connected	100.00%	100.00%	100.00%
Max	99.98%	100.00%	99.95%
Prompter	99.97%	100.00%	99.90%

Table 5.2: Summary models trained with rotation data-set in nominal conditions.

Also, in this case, all the methods are candidates for our task. Figure 5.9 shows the results for the rotation tests. From the graphs, we can observe that the max solution does not solve the problem like in the blur case. Figure 5.7 explains the reason for this behavior: in a very close way to the blur case the network is not able to update its weights.

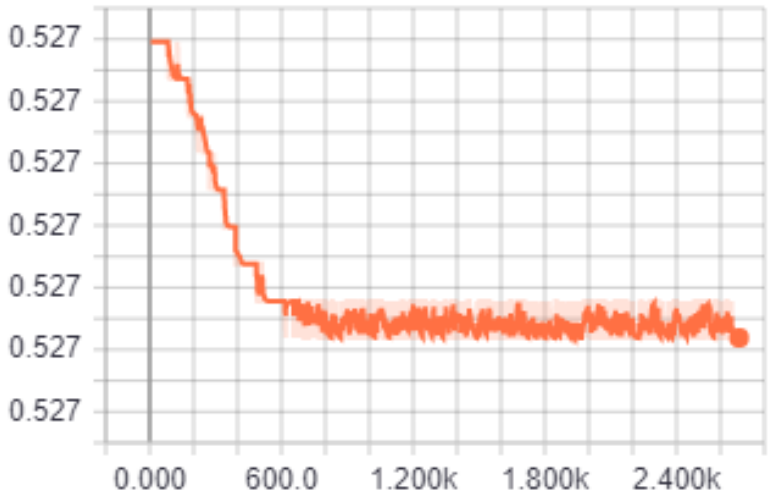
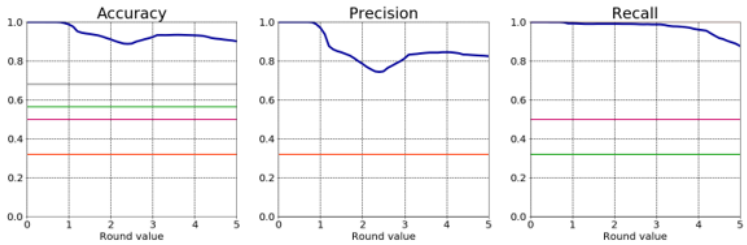


Figure 5.7: Graph loss function train phase max model.

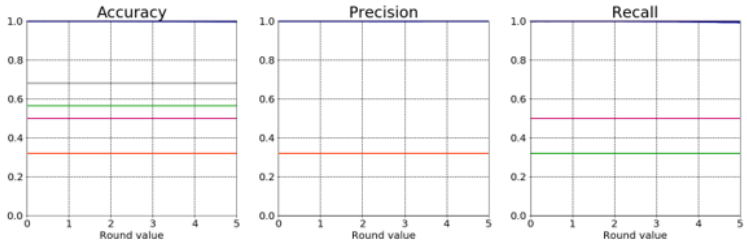
The other solutions have very similar behaviors: in the range $[-2°, 2°]$ the problem is solved while increasing the absolute value of the alteration the solutions just attenuate the effect. That means rotation has a complicated effect on the network and what it learnt during the training phase is not enough to generalize the behavior. We can also notice how the negative alterations are less effective than the positive ones, this is could be explained from the data-set composition: some images altered with negative values are present in the data-set.

5.5 Conclusion

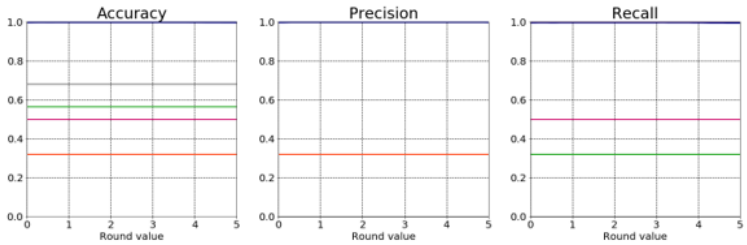
In the previous sections, we improved the robustness of a convolutional neural network in defined conditions enriching the nominal data-set with perturbed data. The results showed that data augmentation technique is the best option when the training data set is available. If training set is not available (e.g. the model was pre-trained and downloaded), the chapter described two valid alternatives for robustness improving.
The next chapter will present two methods to visualize the convolutional neural network attention and to understand how this type of network works.



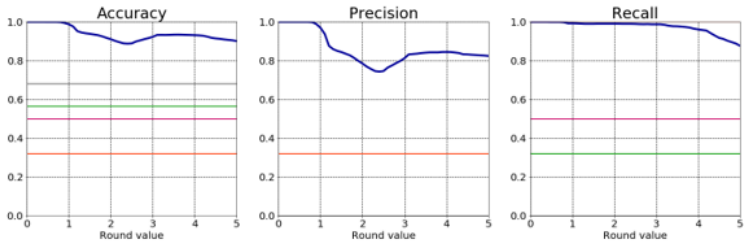
(a) Nominal.



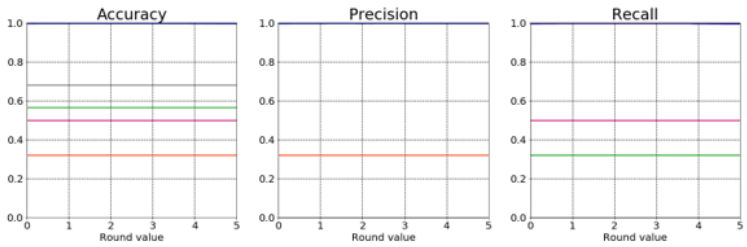
(b) Enriched.



(c) Fully connected.

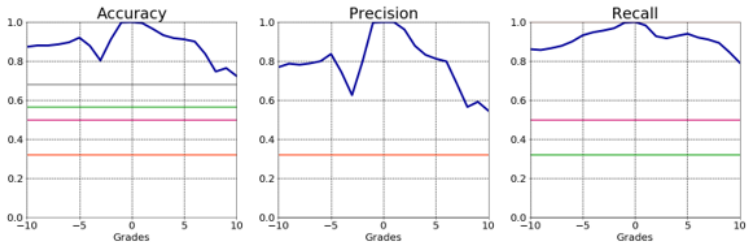


(d) Max.

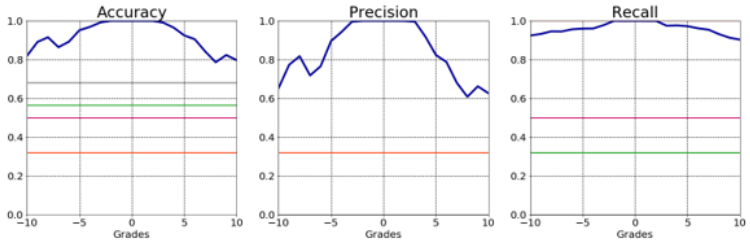


(e) Prompter.

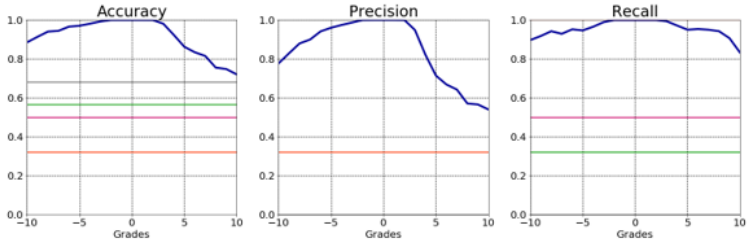
Figure 5.8: Results blur data augmentation tests.



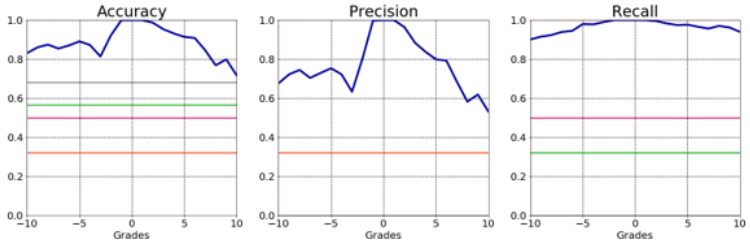
(a) Nominal.



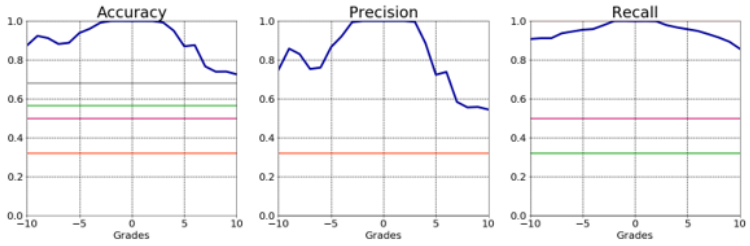
(b) Enriched.



(c) Fully connected.



(d) Max.



(e) Prompter.

Figure 5.9: Results rotation data augmentation tests.

Chapter 6

Attention

6.1 Introduction

The previous chapters analyzed the model with black box studies, limiting the researches only to provide input and gather output.
As the machine learning models implicitly define the behavior from the data, researchers usually see them on a black box approach. This chapter will highlight the focus of a convolutional neural network during the classification process in order to understand its functioning. The final results would be a special image, called heat-map, that highlights with different colors the focus of the model on the image.

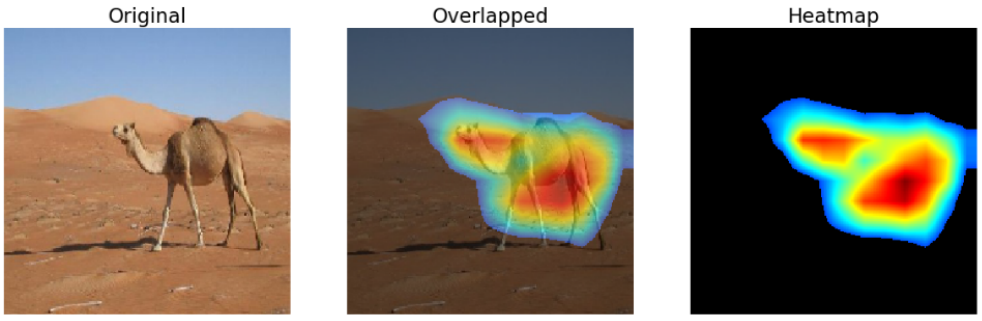


Figure 6.1: Example of attention of a CNN.

6.2 State of the Art

Due to its complexity, the study of neural networks is in continuous development. Only a few works were developed on this particular field.

Zhou et al. [15] propose a simple and efficient method to obtain for a given input an heat-map image that represents the focus for the CNN classification. The work is based on the Class Activation Map (CAM): it is a method that is able to extract the focus of a given CNN highlighting the most important pixels of given in image for the network.

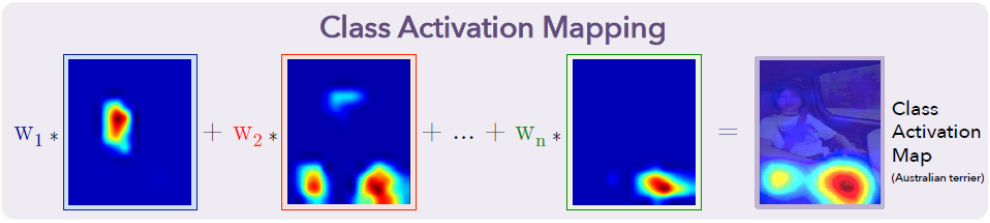


Figure 6.2: Example of application of CAM. [15]

This method requires the presence of a Global Average Polling (GAP) followed to fully connected layer that classifies the image. A GAP is a particular layer that summaries each activation in just a real number applying a simple mean to the matrix. We can see a GAP layer as a normal average layer with the particularity to have a filter of the same dimensions of its input.

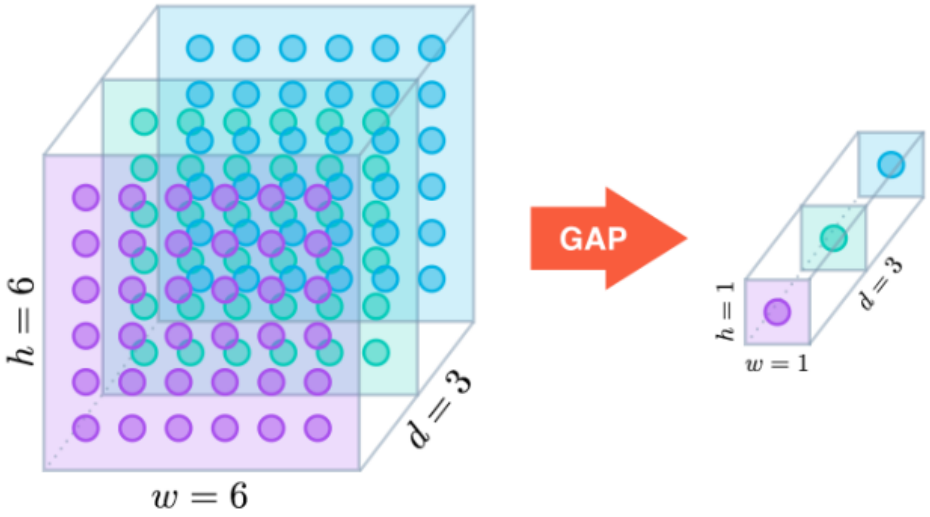


Figure 6.3: Gap layer [3].

Simply speaking CAM method weighted the activations of the last convolutional layer with the weights of the fully connected layer, the result has to expand to the input image dimension with an interpolation method. The final output is the heat-map of the input image. An exhaustive explanation GAP layer and CAM method is present in a [3].

Of course, if this configuration is not present in the network it is necessary to modify the architecture and to retrain the network. Figure 6.4 shows an example of a network structure where it is possible applying CAM method.

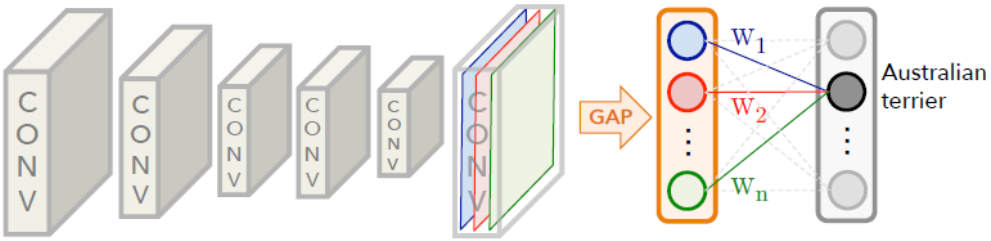


Figure 6.4: Example of structure of CAM. [15]

We can summarize the steps to implement CAM method in the following considerations:

- CAM can work only in particular neural network structures: we need at least one convolutional layer, a GAP layer and finally a fully connected layer. If the network under exam is different it is necessary to modify the network.
- In the case we modified the structure of the network it is necessary to retrain the model with the new configuration. The output of the new network is an evaluated classification of the input.
- We gather the weights of the final fully connected layer.
- For each input image, we gather the activation maps of the last convolution layer.
- It is necessary to re-size the activation maps to the original input size (in general the convolutional networks use polling layer to reduce the size of activations).
- We linearly combine the weights with the resized maps in order to obtain the heat-map.

The first two steps are the main limitations of the method: the necessity to modify and to retrain the network are two of the most expensive phases during the all development process. In the following section, we introduce a framework that allows us to overcome these limitations.

6.3 Framework

At the end of the previous section, we briefly mentioned the limitations of the CAM method. The limit of the network alteration is not a high stumbling block for machine learning researchers: they have just removed all the layers after the last convolutional one and replaced them with a GAP layer and a fully connected one. The most important difficulty is the retraining phase. We can just think the case when the network is bought from an external company or one when we do not own the original data-set; in these situations, it is impossible to retrain the network and so applying the CAM method. Another situation when CAM method is not applicable is when the network is trained with a very big data-set that requires a big effort.

Our framework suggests a simple and efficient way to estimate the heat-map for a given input and for a given convolutional neural network without the operation of modification and retraining of the model. The framework requires the network has at least one fully connected layer and one convolution layer. In this way, both the previous limitations are resolved.

This approach generates a new problem: how can we estimate the weight for the heat-map? The CAM method reduces the activations in single real number and it trains the weights ad hoc in a new training phase.

Figure 6.5 shows the simplified structure of our CNN where the activations are just flattened and taking as input from the fully connected layer. In order to estimate a single weight for each activation, we just calculate an average on the weights relative to that activation. In this case, it is possible because we have only one fully connected layer after the convolutional layer and there is a relationship one to one from the pixels of activations and the weights.

In case there are more than one fully connected layers at the end of the network it is

necessary to weight each point of one activation to all the nodes of every following fully connected layer. Figure 6.6 shows a simple example of multi-layer fully connected.

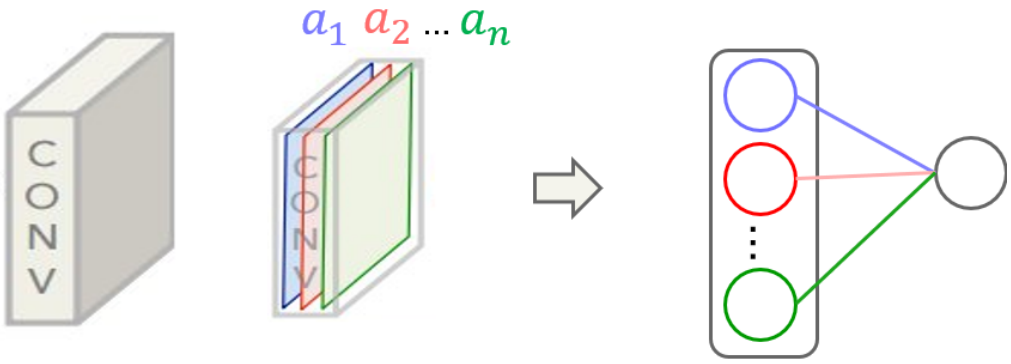


Figure 6.5: CNN where we applied the heat-map framework.

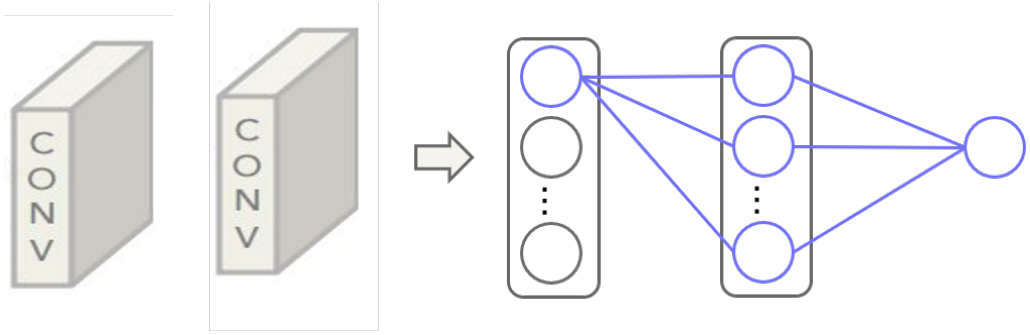


Figure 6.6: Example of structure of CNN with multi fully connected layers.

We highlighted with the color blue the expansion of a single point of one activation inside the network. In order to calculate the weight for the heat-map, in this case, we developed a tricky solution. The basic idea is to calculate the importance of the output of every node on the output of every node of the following layer. Repeating the procedure for each couple of fully connected layers in the network tail. At the end combining all the weights referred to the nodes of the start layer. Let's an example of the calculus of the weights for a couple of layers. Given two layers L and $L+1$, for each node i of L we calculated the product between the $output_{\text{i}}$ and the $weight_{\text{j}}$, where j represents a node of the layer L+1 and the $weight_{\text{j}}$ is the weight for the $output_{\text{i}}$ in the node j. The result is divided for the $output_{\text{j}}$. The final result represents the weight of the $output_{\text{i}}$ to $output_{\text{j}}$. So for each node, we apply the formula (6.1).

$$Weight_{\text{i-j}} = \frac{output_{\text{i}} weight_{\text{i}}}{output_{\text{j}}} \tag{6.1}$$

This solution generates for each layer a matrix with dimension $(N_{\text{L}} x N_{\text{L+1}})$ where N represents the number of neurons for a given layer. The last matrix is reduced to a column vector selecting the class of our input, simply speaking we choose the node of the layer $L + 1$. Starting from the last couple of layers we multiply the matrix for the vector and then we perform a sum for each row. We repeat the operation for each fully connected layer and at the end, we obtain a vector of weights used to the calculus of the heat-map. The precision is the limit of the framework: in particular, in the case of multi fully connected layers the framework does not consider the non-linear functions and the calculated weight is just an estimate to the real value.

6.4 Results

In this section, we present the results we obtained with the framework. The framework implementation is very easy to use: the user has to create the instance of the class and to generate the heat-map. The class requires only the trained model, the class of the image expressed as an integer and the input image; it returns the heat-map and the overlapped image.

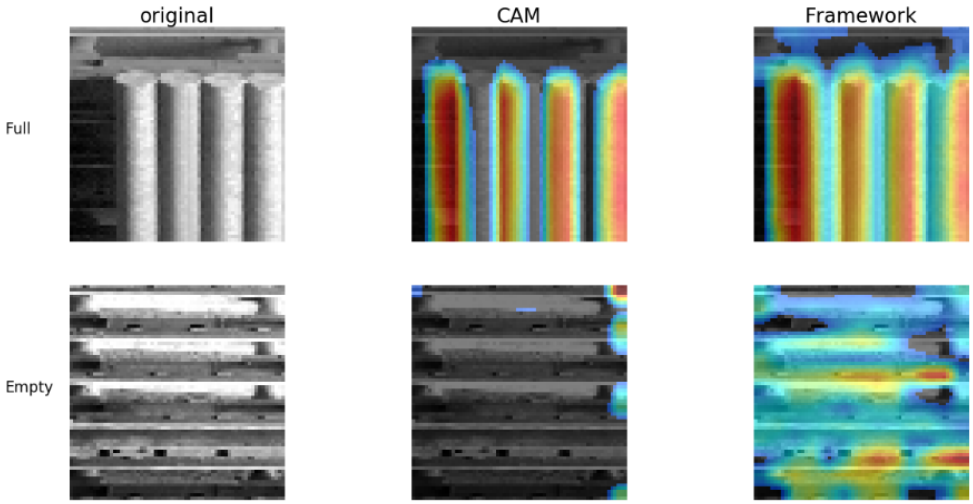


Figure 6.7: Examples of heat-map generated with CAM and framework.

Figure 6.7 shows the difference between the application of the CAM method and our

framework. We can notice the CAM is more selective than the framework: the reason is very easy, CAM applies an average operation on the activations and it maintains only the principal information. However, the effort needed to apply CAM method is not justified from the more accuracy.

The CAM method shows how the network takes the decisions: Figure 6.8 highlights that the network recognizes the great difference of the dark and light colors: we can see that the network focus is on the edge of the billet while it does not see anything of the center. In general, it extracts and recognizes the vertical lines of the input. In the images classified as empty there is not a general scheme: sometimes the network recognizes the horizontal bar of the benches in other cases it focuses in different areas.

This little analysis provided us some important information about the functioning of a CNN.

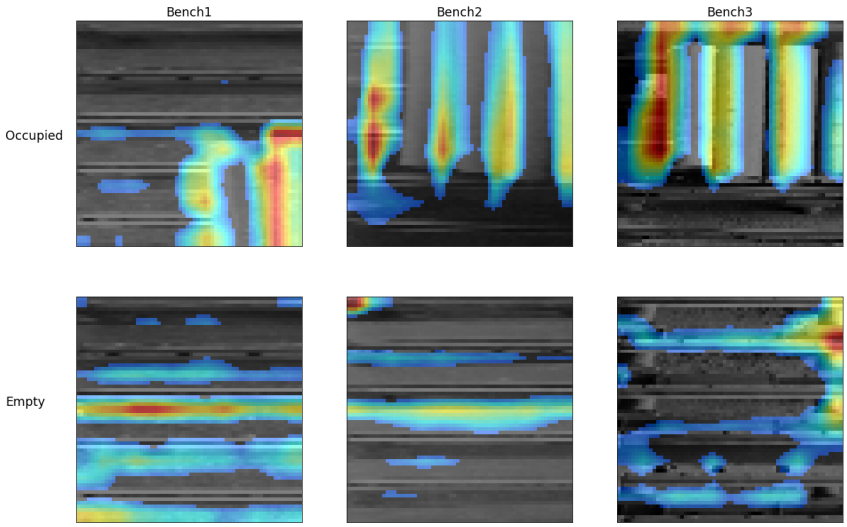


Figure 6.8: Examples of heat-map generated with the framework for the three benches.

We tried to use the new knowledge in order to code a simple script that is able to modify an image in order to have a misclassification. The basic idea is to start to modify the images from the max-values point of the heat-map and then moving in the neighborhood. To generate an adversarial example we tried to confound the network: from the heat-map, we saw the CNN focuses on the vertical line in order to find the billets; to misclassify

a full image in an empty one we colored with dark points the max-valued pixels of its heat-map and the max-valued points of an empty image heat-map; we tried to hidden the billets. In the other scenario, from an empty classification to a full one, we colored with white points the max-valued pixels of a full image heat-map (we chose an heat-map from the same bench); in this case, we tried to *create* a fake billet.

Figures 6.11 and 6.12 shows some adversarial examples generating with this simple method: we can notice that not with all the images it was possible to cheat the network in a limited number of iterations. For this test, we limited the number of iterations to 500 and for each iteration, we modified one pixel plus a small neighborhood (in particular a square 3x3 and 5x5 around the selected pixel). We performed different attempts with different sizes of modification area for each iteration and with different full heat-map images for the case empty-full. It is possible to play with this parameters in order to obtain the minimum alteration that generates an adversarial example, but in general, it is often possible to see the alteration. We can notice that it is more simple generating an adversarial example starting to an empty image.

6.5 VGG16 Test

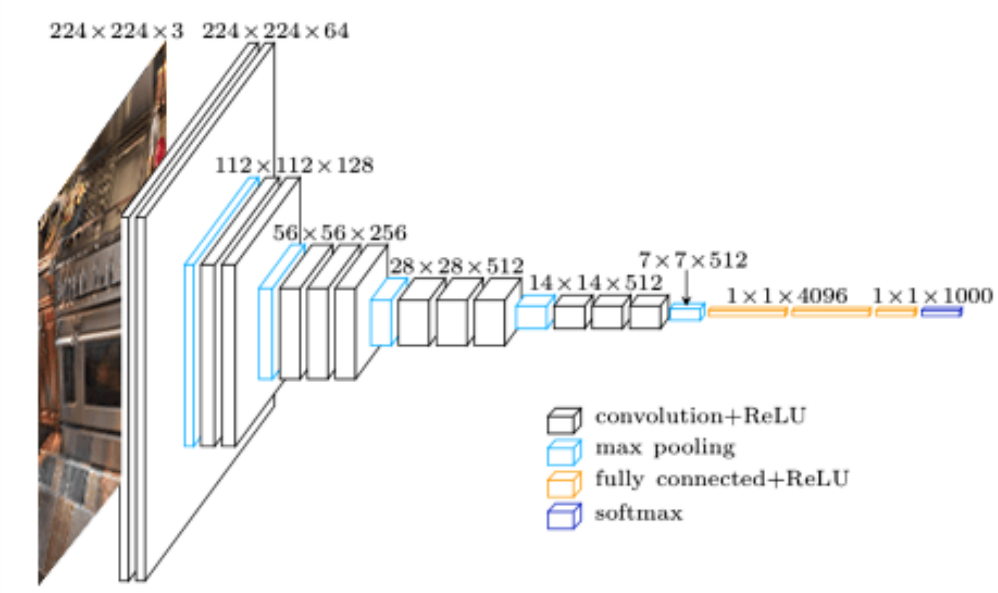


Figure 6.9: Structure of VGG16 [11].

In order to test our framework in the *real* world, we tried our framework with a more complex model like VGG16. The trained model is part of the Keras application models, so we just downloaded the trained network from Keras repository. The network structure is very complicated: we have a series of convolutional layers with a growing number of filters (64, 128, 256, 512). After every convolutional layer is present a max polling that divides in half the image dimension. At the end we have three fully connected layers, the last one has 1000 of nodes representing the classes of the network. The model is able to accept different sizes of images in input, the minimum size is 32x32x3.

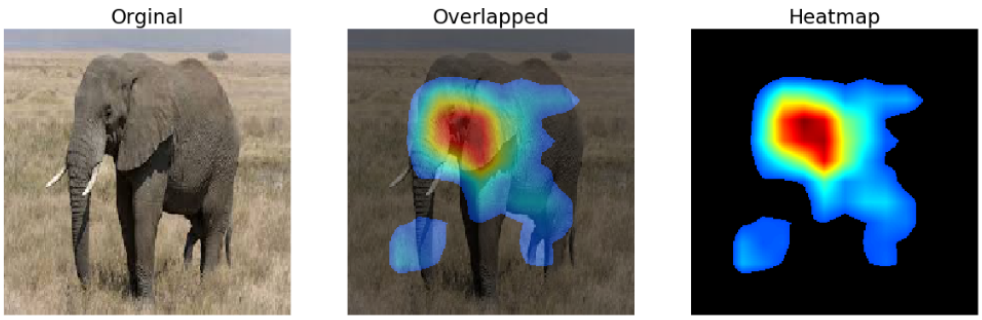


Figure 6.10: Heatmap from VGG16 generated with the framework.

6.6 Conclusion

The previous sections described two methods to visualize the focus of a generic convolutional neural network. The heat-maps allow us to understand the internal functioning of the model and provides hints to support the generation of some adversarial examples. The analysis shows the complexity to cheat our model improving our knowledge and confidence on it. The next chapter will analyze the structure of the model applying two techniques to simplify the network architecture and measure the impact of this reduction in terms of robustness.

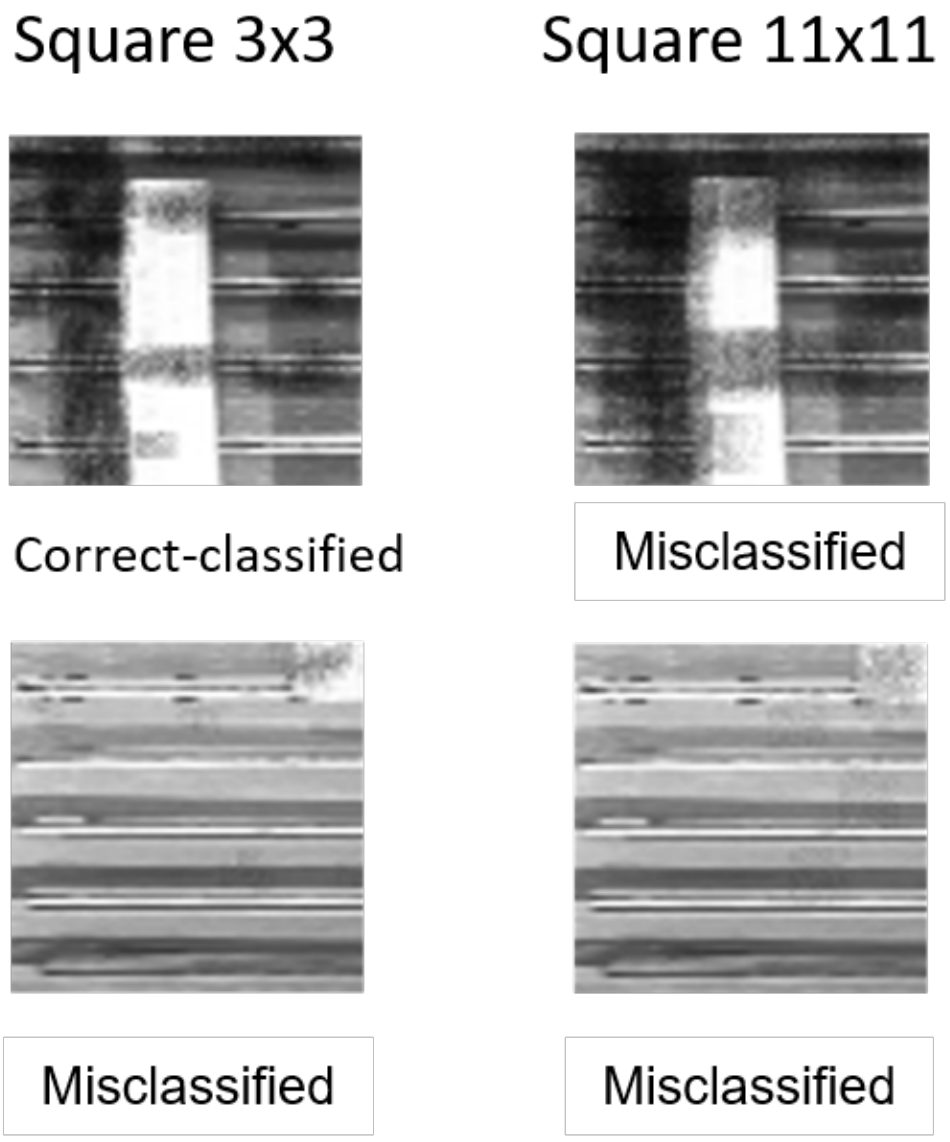


Figure 6.11: Examples of adversarial examples full-empty with 2 different size of area.

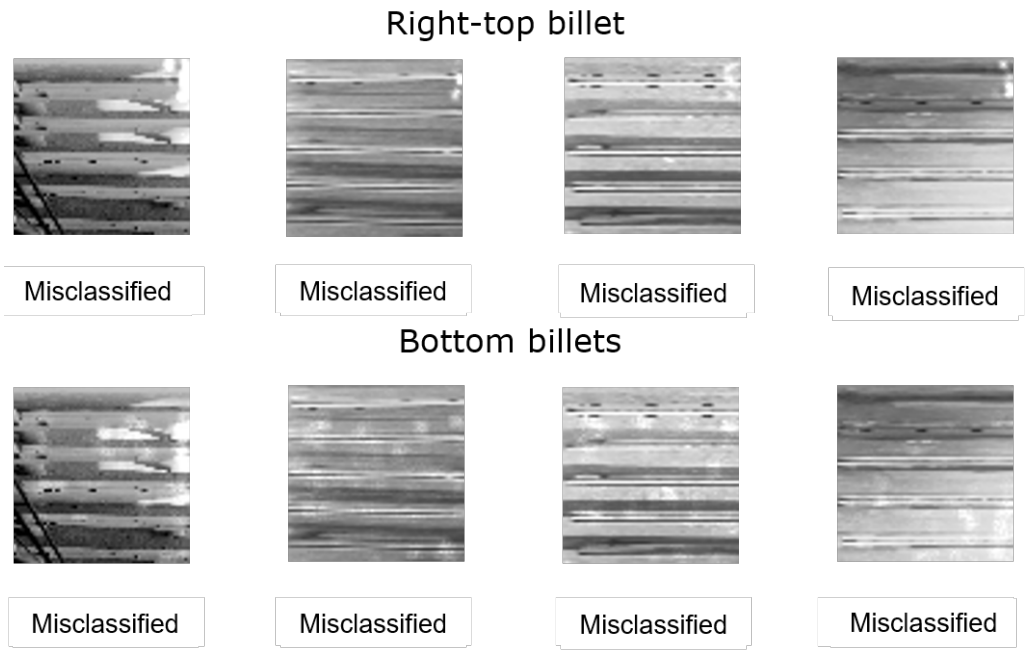


Figure 6.12: Examples of adversarial examples empty-full with 2 basic images.

Chapter 7

Robustness of Pruned CNN

7.1 Introduction

Network architecture fulfills an important role in defining potential performances of the model: when they are too simple, the models shows a bias, i.e. inability to fully extract patterns from data. When the architecture is too complex, the model exhibits the tendency to over-fit, i.e. the inability to generalize patterns. Moreover, the larger is the network, the higher is the computing power demand for training and for inference. Therefore one of the hardest tasks for data scientists and machine learning engineers is to tune the right network size.

This chapter provides techniques to estimate the impact of network size reduction on the robustness metric. custom structure gives to the network the capability to generalize from the training data, so the architecture choice influences the overall system.

7.2 State of the Art

In the previous chapters we could see how to measure and improve the robustness of our network, but what happens if we use a smaller network? We can image two situations: if the network is well-formed the new network will be light and fast but also less robust; in the other hand if we start with an oversized network for the problem we could reduce the over-fitting and so improving its performance.

A pruned network is a reduced model that all the useless neurons or filters have been

removed. The idea is to apply a state of the art method in order to reduce the number of parameters of the network with the target to generate a light and fast network. This an important field of machine learning: complex problems are solved with big and rich architecture, but this makes difficult running the model in embedded or low power systems. Pruned methods allow the researchers to reduce the size of the networks removing the least important filters; of course, it's reasonable expected the process decreases the performance of the model. The state of the art of network pruning is composed of many different approaches that they can be grouped into two classes:

1. Methods that require the retraining of the network. These methods just decide how many filters or neurons removing: it does not matter which filters will remove because the network has to be retrained at the end of the process.

2. Methods that do not require the retraining of the network. In this case, the methods directly modify the structure and the values of the filters.

[10] proposes a series of methods to prune a convolutional neural network. All the methods are based on an iterative process that removes the least important filters for the final output; the difference between the methods is the heuristic criterion used to define which filters are essential to the classification and which not. Some examples are:

- Number of zeros: if a filter has a number of zeros major than a defined threshold the filter is removed.

- Activation: removes the filters with low values of activations. In order to define a lower bound, the article calculated the mean or the standard deviation of activations for each layer.

- Mutual information between the filters.

- Taylor expansion: we expand with Taylor method the difference between the nominal cost function and a final one obtained removed a selected filter.

7.2.1 Taylor Expansion Heuristic

In this section, we analyze better the Taylor expansion heuristic described to [10] and we will implement it to reduce the size of the network. The method evaluates the importance of each filter on the final loss function with a gradient analysis and it removes the least important filters. The method returns the minor number of filters that satisfy the constraint to the loss: it necessary to set the maximum absolute alteration we want to bear on the pruned network.
The method is based on an optimization of the function:

$$|\Delta C(h_{\mathrm{i}})| = |C(D, h_{\mathrm{i}} = 0) - C(D, h_{\mathrm{i}})| \tag{7.1}$$

The equation defines the difference between the function cost calculated with all the parameters and without the parameter h_i. We can rewrite the cost function with $h_i = 0$ through a Taylor expansion of the first order:

$$C(D, h_{\mathrm{i}} = 0) = C(D, h_{\mathrm{i}}) + \frac{\delta C}{\delta h_i} h_i + R \tag{7.2}$$

R represents the remainder of the expansion, we consider the value as negligible. So our difference is now define as:

$$|\Delta C(h_{\mathrm{i}})| = \left| C(D, h_{\mathrm{i}}) - \left(C(D, h_{\mathrm{i}}) + \frac{\delta C}{\delta h_i} h_i \right) \right| = \left| \frac{\delta C}{\delta h_i} \right| \tag{7.3}$$

With the equation (7.3) we demonstrated the difference between the two cost functions is exactly the gradient of h_i. Once we defined the maximum variance of the loss and we can apply the method to our network. Finally, we retrain the network with the new hyper-parameters.

7.3 Heatmap Method

In the previous section, we describe one of the many heuristic methods to reduce the size of a given convolutional neural network. In this section, we propose a simple and fast method to reduce the number of the filters of the last convolutional layer using the heatmap framework. In the framework, described in chapter 6, we extracted the activation maps from the last convolutional layer and we weighted them according to the output.

The basic idea of our pruning method is to use this weighted activations to find those filters are less used in the layer, and then we remove those filters.
The method is very simple and intuitive, but it can be applied only on the last convolutional layer. For this experiment we used only the validation set: in this way we are sure to not influence the results with images already seen from the network. To apply the method we calculate the weighted activations for every image with the heat-map framework. Then we calculate a global mean of all the weighted activations: this measure represents our threshold. For each filter, we compute the mean of the respective activations. The filters with a mean minor of the threshold are removed. Then we retrain the network with the new number of filters.

7.4 Results

In the work, we just applied the two described methods and we obtained two different networks. We have to precise we started intentionally from an oversized model in order to compare the results with the pruned ones. We performed the robustness analysis to compare the networks.
Table 7.1 shows the results of the pruning process with the different methods. We can see a sensible reduction of the number of filters: this could mean the original network was too large for the problem. The second row defines the best configuration obtained in iterative mode: retraining the network many times with different values of hyper-parameters.

Method	**# Filters 1° conv.**	**# Filters 2° conv.**
Nominal	20	40
Best iterative model	10	20
Taylor expansion	12	21
Heatmap method	20	26

Table 7.1: Result pruning phase.

We notice how the Taylor method from the state of the art is the most efficient and it is very close to the best model obtained by iterations. The main difference between this

method and an iterative one it is the time: the traditional method requires to retrain the network many times while with the pruning method just one time. The heat-map method reduction is applicable only to the last convolutional layer, in general the biggest one. The heat-map method, despite its simplicity, has good results on the last convolutional layer, unfortunately, it is not possible to expand to other layers.

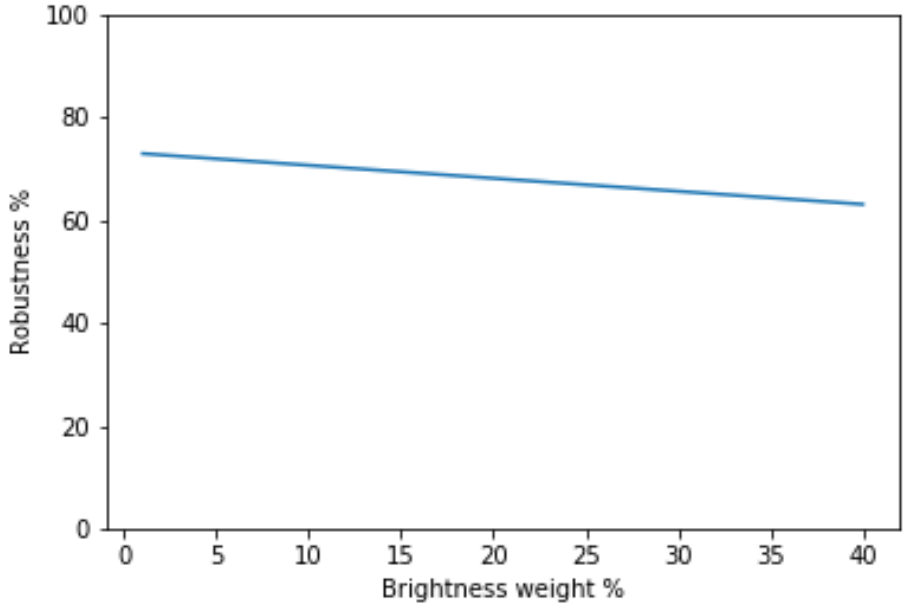
The aim of this chapter is that to analyze what happens to the network when the user applies a reduction technique. We performed the analysis only for the Taylor pruned model. Table 7.2 shows the results for the linear and exponential distribution for the network.

Alteration	**Interval**	**Robustness %**	
		Linear	**Exponential**
Brightness	[0.5, 2]	78.69	93.79
Occlusion 1x1	[0, 200]	92.13	98.85
Occlusion 2x2	[0, 50]	73.23	86.34
Occlusion 4x4	[0, 30]	41.15	52.47
Blur	[0, 2]	80.53	100.00
Green line	[0, 2]	38.89	45.83
Translation Horizontal	[-5, 5]	97.49	99.85
Translation Vertical	[-5, 5]	67.61	96.97
Rotation	[-5, 5]	53.46	76.03
Perspective Left	[0, 15]	91.89	98.50
Perspective Right	[0, 15]	95.19	99.16
Zoom	[-5, 5]	76.50	97.92

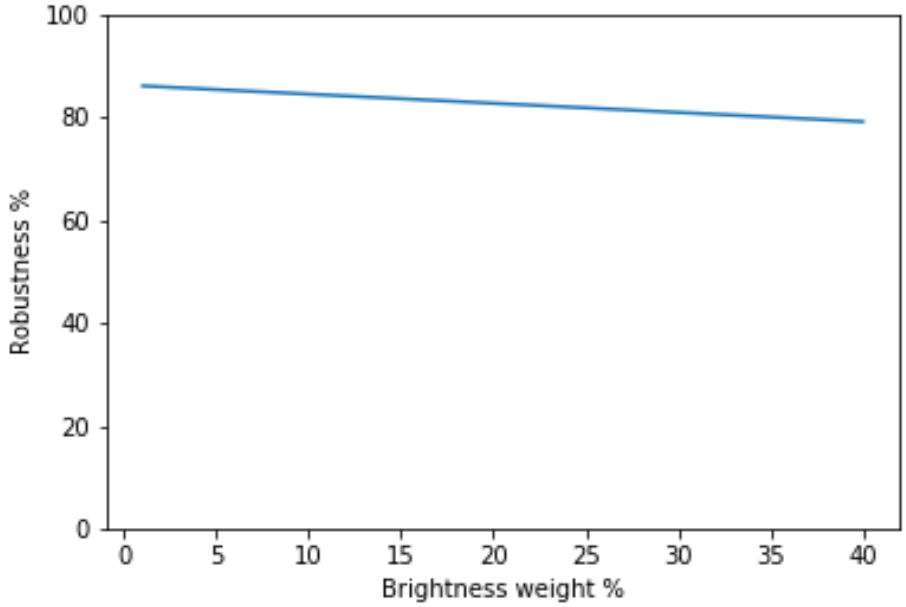
Table 7.2: Significant intervals and results from Taylor expansion pruned model.

By comparing Table 4.2 with Table 7.2 we can see many values are changed: some of them increased their accuracy, for example, the brightness that earned almost 10 percentage points, others decreased their values, for example, the green line alteration. The general robustness for both the distributions is smaller than the nominal robustness, but the reduction, maintaining the same nominal behavior, positively affected the size of the

network and also some perturbations, in particular those that alter the entire image (i.e. brightness, translation and perspective): this means the nominal network was oversized and for those specific alterations it was not able to generalize.



(a) Generated with linear distribution.



(b) Generated with exponential distribution.

Figure 7.1: 2-D graphs of global robustness.

7.5 Conclusion

The previous chapter analyzed the robustness on the variation of the case study network structure. As the data quality, also the network architecture influences significantly the model behavior. The chapter shown that the case study network suffered of over-fitting and it was unable to generalize the knowledge learnt from the training data-set. In adding to the state of the art method we proposed a novel pruned method based on the network attention.

In the previous chapters, we analyzed and improved the robustness of a convolutional neural network but we can't guarantee the completely safety in all the conditions. The next chapter will discuss a particular form a testing called verification that allows us to be sure of the presence or not of critical situations.

Chapter 8

Verification

8.1 Introduction

Software testing increases the confidence of the researchers on a software but it can not ensure the completely absence of adversarial examples due the impossibility to perform an exhaustive search. The state of the art of software defines a technique that allows an exhaustive search in bounded conditions: software verification. This chapter will present a novel method to verify the presence of adversarial examples in the simplest neural network: logistic network.

8.2 State of the Art

Software verification is an important branch of testing that allows the user to have the 100% of confidence that its software is correct. In general, software is too big and its domain is too large to be complete explored and neural networks do not make exception. It is possible to workaround the issue using software verification. The state of the art of traditional testing includes the branch of software verification: the aim of this field is to have a complete confidence in the system. In general, there are two methods to verify a software:

1. **Symbolic execution**: it is a particular execution of the code where the tester uses symbols to represent the input domain. The main advantage of this method is to drastically reduce the input domain.

2. **Model-based testing**: the software is converted to a simpler model that is able to perform all the functions of the real code. The aim of this method is to prove mathematical proprieties increasing the confidence of the tester on the software.

The state of the art of neural network verification is generally based on symbolic execution that helps the researchers to reduce the input domain and the execution time. Symbolic execution is very complicated also for little code: the execution of the code is performed replacing the real values with symbolic formulas solved at the end of the execution process.

[5] defines a method to verify a neural network based on a reduction of the input domain and a symbolic execution. The method defines a formal definition of safety: for [5] a neural network is considered safe if it was not possible to find an adversarial example classified rightly from humans. The authors define a list of possible perturbations limited in a bounded input domain. The method researches the smallest perturbation, inside the limits, that generates an adversarial example. The process is applied to a single layer and repeated for all the layers of the network. So bounded the input space, the network is considered safe if no negative examples were found.

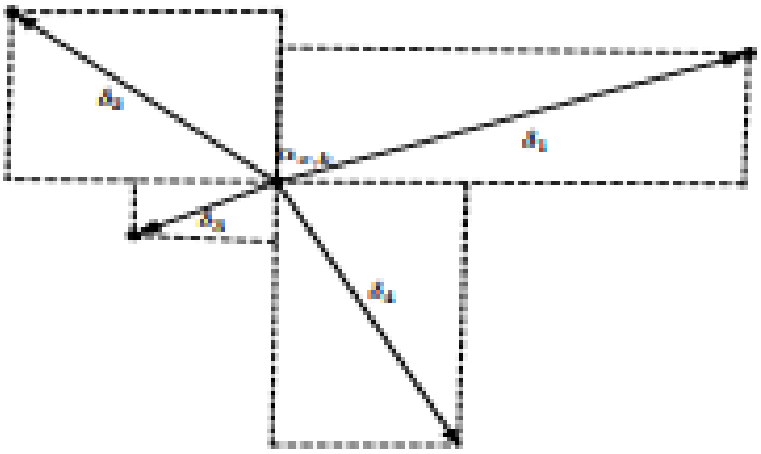


Figure 8.1: Example of bounded alterations [5].

The limits are necessary to reduce the space of the possible activations. To a single activation is possible to apply different perturbations at the same time: the evolution of the

image perturbations is represented through a *ladder* 8.2. To compare the original image and the new one, it measured the distance between the two activations calculated as the greatest sum of the all alterations.

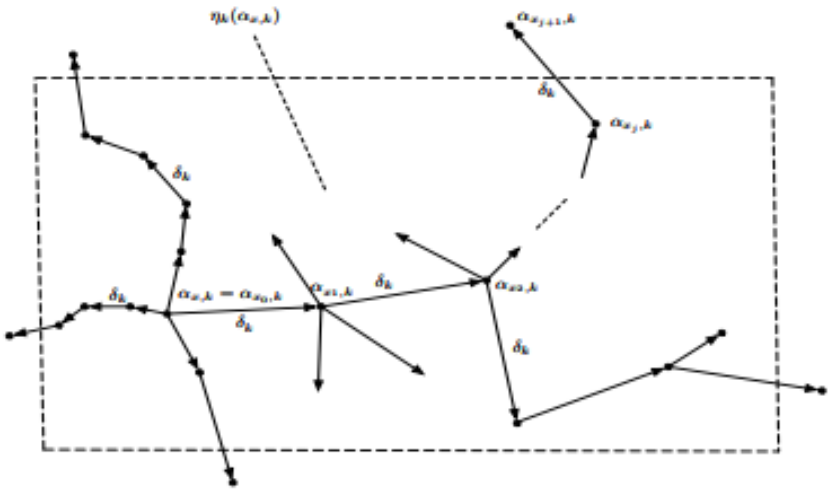


Figure 8.2: Example of a ladder.

The implementation of the method requires two agents: one agent chooses the manipulation and the other the value of the perturbation: positive or negative. In order to have an exhaustive exploration in a reasonable time, it is necessary to use a symbolic execution reducing, in this way, the input variance. Figure 8.3 shows some results got from [5].

Figure 8.3: Example of [5] results.

[7] suggests a different approach to neural network verification: the authors used the Satisfiability Modulo Theories (SMT) plus the operation (+, -, ·, $\leq$, $\geq$) generating a new domain called Simplex and they modified it to incorporate also the non-linear function RELU. The method is an iterative process that generates an adversarial example modifying the activation values and maintaining verified all the theories. Also, in this case, the authors limit the activation values in an upper and lower bound.

8.3 Verification on Logistic Network

In this section, we propose a different approach to verify logistic neural networks. A logistic neural network is a simple model composed only to two layers: input layer and the output one. We chose this type of network in order to focus only on the method and not on the complexity of the network. Verification allows us to increase the confidence to our network: rewriting the model in a closed form we can perform every mathematical operation and prove many properties. The basic idea is to calculate in a bounded domain an adversarial example for each image and checking if the example is well-formed: during the pre-processing phase, every image is normalized in the range [0,1] so all the pixels in the calculated image has to be in the range [0,1]. In case we are not able to find any adversarial examples the network is safe for that specific conditions.

The model is very simple so we can rewrite it in one equation:

$$Prediction = f(WX + b) \tag{8.1}$$

Where *W* is the vector of weights, *X* is the image and *b* is the bias. *f* is the non-linear function, we are working on a classification task and so we used the sigmoid function, state of the art for this field.

$$f = \frac{1}{1 + \exp(-(WX + b))} \tag{8.2}$$

Once writing the model as a simple equation we can apply our method. The method is:

- We change the prediction in order to modify the network classification; in our case, we fixed it very close to threshold (0.4 for the full images and 0.6 for the empty ones).
- We explicit the variables we want to modify. We can apply the method for one single pixel of for a small group of them.
- We calculate the values of our variables.
- We check if the new variables have values in the range [0,1]. In this case, we found an adversarial example.

We implemented the method modifying one and two pixels at the same time. With a little effort is possible to extend the method to a bigger group of pixels.

8.3.1 Example of Alteration of One Pixel

In this section, we present an example of our verification method modifying one pixel at a time. We rewrite the equation as following:

$$x'_{\mathrm{h}} w_{\mathrm{h}} + f^{-1}(WX + b) - x_{\mathrm{h}} w_{\mathrm{h}} - f^{-1}(Prediction') = 0 \tag{8.3}$$

where and x'_{h} is our variable, then we explicit the x:

$$x'_{\mathrm{h}} = -\frac{f^{-1}(WX + b) - x_{\mathrm{h}} w_{\mathrm{h}} - f^{-1}(Prediction')}{w_{\mathrm{h}}} \tag{8.4}$$

Now we have to choose which pixel we want to modify. In this simple case, it is possible to try every combination and taking the pixel with the minimum difference from the original one. We just compute the variable value and we check if the result is in the interval [0,1]. Figure 8.4 shows some examples of adversarial examples.

Modifying only one pixel for every image we found only 9 negative examples (about 0.15% of the data-set has at least one adversarial example).

Bench	**Position**	**Image**	**Actual predict**
Bench 1	Bottom	62	0.64
Bench 2	Bottom	15	0.67
Bench 2	Bottom	16	0.77
Bench 2	Bottom	20	0.74
Bench 2	Top	154	0.78
Bench 2	Top	265	0.61
Bench 2	Top	301	0.85
Bench 2	Top	329	0.81
Bench 2	Top	392	0.78

Table 8.1: Images with an adversarial example.

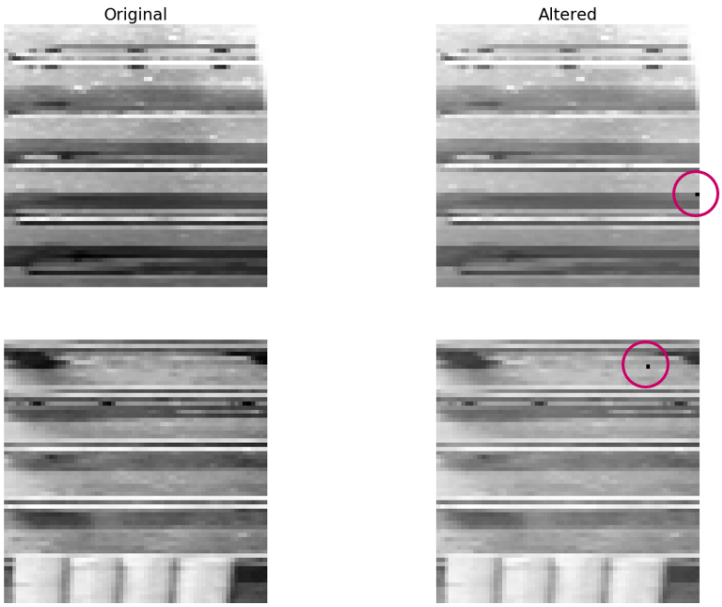


Figure 8.4: Examples of adversarial images with 1-pixel modified. The magenta circle identify the modified pixel.

Table 8.1 shows the images that it was possible to find an adversarial example, the first thing we can notice that all the images are classified as full in the nominal situation. The modification of only one pixel alters very few the image and also a simple network like the logistic one is robust on this attack. We have seen in chapter 6 what type of alteration is necessary to generate misclassification on convolutional neural network for this simple problem: it is not possible to find an adversarial example modifying only one pixel.

8.3.2 Example of Alteration of Two Pixels

In this section, we present an example of our verification method modifying two pixels at a time. We rewrite the equation as following:

$$x'w_{\mathrm{x}} + y'w_{\mathrm{y}} + f^{\text{-}1}(WX + b) - xw_{\mathrm{x}} - yw_{\mathrm{y}} - f^{\text{-}1}(Prediction') = 0 \qquad (8.5)$$

In the previous section, we tested all the pixel in order to find the best one to modify, in this situation that approach is too expensive: each image has a dimension of 64x64x1

so in total it has 4096 pixels, every pixel can be combined with other 4095 one; so for each image, we have more than sixteen million of possibility. We thought a smarter approach: we get a negative example if the values of the two chosen pixels are in the range [0,1]. Of course, we can choose the value of one pixel and calculate the other one as consequence of the first. We decided to vary the y and to calculate the x. Equation (8.6) allows us to calculate the x.

$$x' = -\frac{y'w_{\mathrm{y}} + f^{-1}(WX + b) - xw_{\mathrm{x}} - yw_{\mathrm{y}} - f^{-1}(Prediction')}{w_{\mathrm{x}}} \tag{8.6}$$

In order to maximize the possibility of a negative example we have to minimize the term:

$$\left| \frac{y'w_{\mathrm{y}} + f^{-1}(WX + b) - xw_{\mathrm{x}} - yw_{\mathrm{y}} - f^{-1}(Prediction')}{w_{\mathrm{x}}} \right| \tag{8.7}$$

We choose the y that maximizes the product $y\ w_y$ and then we try with all the possible x. Now we chose the two pixels that we want to modify but we did not choose the value of the y yet. One of the most important thing in adversarial example generation is creating examples close to the nominal images that a human person is able to correctly classify. So we want to select the two values that reduce the distance between the altered and the original image. The computation of image distance is a very big field: there are a lot of methods that we can use. We chose to use the Manhattan distance for its facility of implementation and computation (8.8).

$$Distance_{\text{Manhattan}} = |y - y'| + |x - x'| \tag{8.8}$$

Minimizing the Manhattan distance between the two images we get the values of the pixels.

The modification of two pixels improves the results of one pixel test but as above we are able to modify only full images. We obtained an adversarial example for about 0.30% of the data-set (8.2).

This approach is a white-box method that allows the user to be sure if his network is vulnerable to adversarial examples or not. Of course, it is possible to improve the method increasing the number of modified pixels for each iteration with a little extra effort.

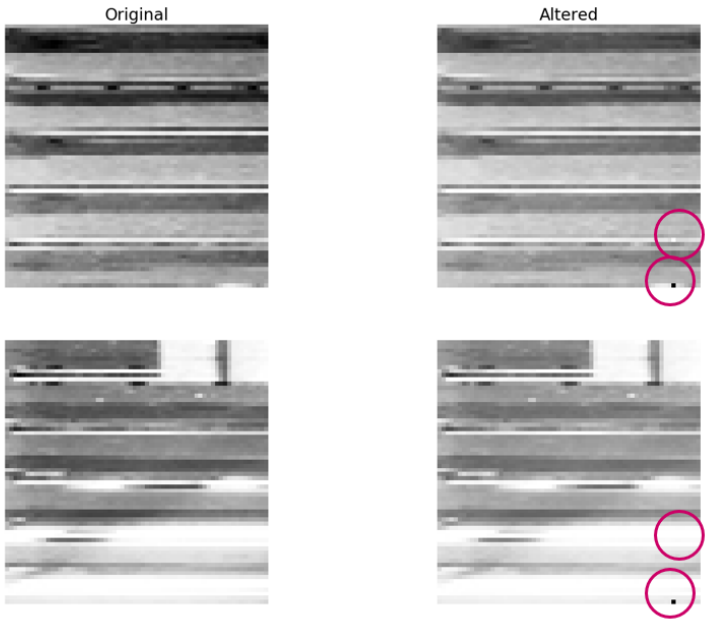


Figure 8.5: Examples of adversarial images with modification of 2 pixels.

8.4 Verification of CNN

In this section, we present a tentative of implementation of the previous analytic method to a more complex network as a CNN. As shown in chapter 6, it requires a certain effort to cheat a CNN, in particular our network: the case study is very simple and the network has just to classify the images only in one class. Our network, as shown in chapter 7, was bigger than the necessary size and also the data-set contains a lot of very similar images. Of course networks with thousands of class being able to classify many types of images are simpler to cheat. Figure 8.6 shows an example of adversarial example on a complex convolutional neural network. We can notice how simply adding some noise it's possible to change the classification.

In general, it is not possible to write a CNN model in analytic way without a big effort, also for a very simple model. The problem is the convolutional and the polling layers that introduce matrix operations, very expensive to write in an equation. We decided to apply the method only on the last fully connected layer and then rebuild the image from the altered activations. Using deconvolution layer and a simple trick for the pooling layer

Bench	Position	Image	Actual predict
Bench 1	Bottom	10	0.84
Bench 1	Bottom	62	0.64
Bench 2	Bottom	15	0.74
Bench 2	Bottom	16	0.77
Bench 2	Bottom	20	0.74
Bench 2	Bottom	265	0.78
Bench 2	Top	151	0.85
Bench 2	Top	153	0.87
Bench 2	Top	154	0.78
Bench 2	Top	155	0.64
Bench 2	Top	175	0.82
Bench 2	Top	265	0.61
Bench 2	Top	301	0.85
Bench 2	Top	328	0.74
Bench 2	Top	329	0.81
Bench 2	Top	362	0.83
Bench 2	Top	385	0.79
Bench 2	Top	392	0.78

Table 8.2: Images with an adversarial example.

it is possible rebuilding the input. In order to reverse the pooling layer, we start from the original activation in input to the pooling layer and we adapt it to the new output modifying only the elements that would make the output different.

8.4.1 Deconvolution Layer

Deconvolution layer is very similar to a convolution one. The layer performs a convolution operation using the transpose of convolutional filters. Of course, this changes also the equation to calculate the size of the result of the operation (8.9).

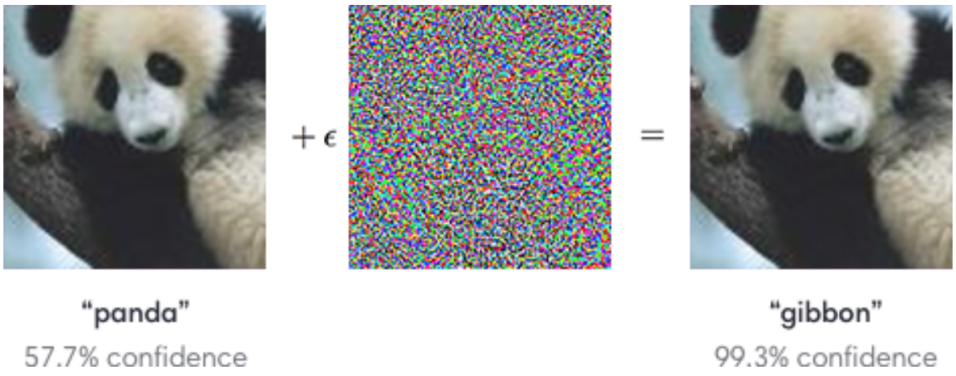


Figure 8.6: Example of adversarial example.

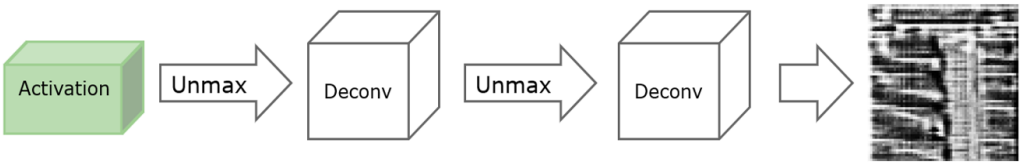


Figure 8.7: Generation image from activations.

$$Output_{\text{size}} = stride(input_{\text{size}} - 1) + filter_{\text{size}} - 2padding \tag{8.9}$$

The deconvolution process has a big problem: during the convolutions process the data are reduced and mixed based on the size of the filters and the strides. It is not possible to rebuild exactly the original input from a convoluted data; the best result obtained is a low-resolution image.

In general, it is possible to reduce this effect by extending the image through an interpolation. This allows us to have more information to rebuild the original image and to reduce the data mixing. Figure (8.10) explains better the concept: without resizing the image the deconvolutional layer adds automatically some padding, formed of 0s, while with the resizing padding is not necessary.

8.4.2 Method Implementation

In this section, we try to implement the method used with the logistic network with a CNN. Of course, it is not possible to describe in analytic way all the convolutional network so we decide to rewrite only the last fully connected layer, applying the verifica-

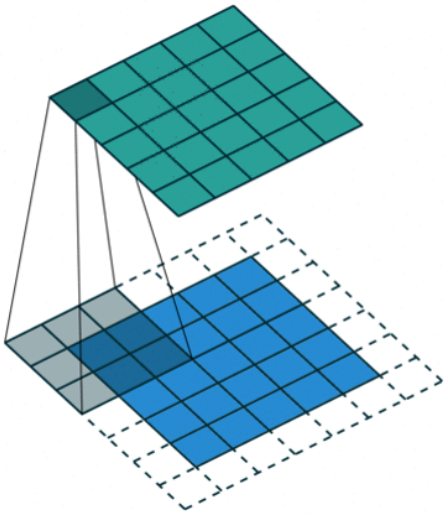


Figure 8.8: No resized image.

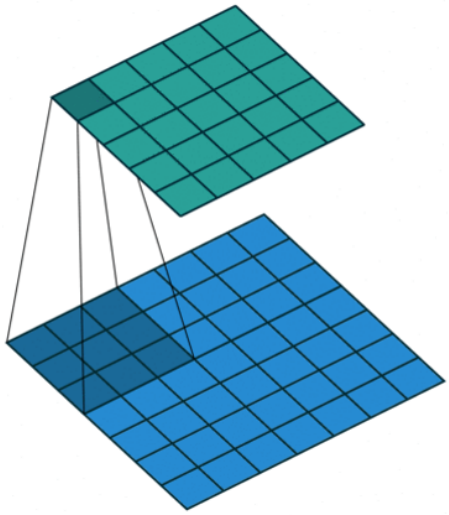


Figure 8.9: Resized image.

Figure 8.10: Difference between resized and no-resized images.

tion method and rebuilding the image using deconvolutional layer and the tricky for the polling layer starting from the altered activations. In the previous section, we quickly describe how a deconvolutional layer works and how to improve the results of that layer. The resized technique mitigates the mixing effect and provides more information to the deconvolution layer; it is not able to resolve the problem. Figure 8.11 shows the best deconvolutional results obtained during the work. We can see the big difference between the original and the new image: the deconvoluted image is a low-resolution original one. To increase our result we try to overlap the original image with the built one obtaining a discrete result. This solution has a negative effect on the perturbation: the overlapped mitigates the alteration.

If we try to evaluate the new image with the network, we do not get the wished result: the classification is still correct just with a less precision. The main problem with this implementation is that the alteration performed on one or a small group of pixels is deployed in all the image during the rebuilding phase and at the end, it is not perceptible from the network. In addition to it, the overlapping of the original image reduces further the perturbation. The final result is a not altered image. It is not possible applying the method only on the last layer of a CNN. Future works can be extended the method to a bigger group of layers.

Figure 8.11: Example of a deconvolutional image.

8.5 Conclusion

The previous sections described a novel network verification method. The technique is not applicable to all the neural networks, but just on the simplest ones due the difficulty to write the complex models in analytic form. For logistic network we shown a scalable method to prove the presence of adversarial examples and we proved the difficulty on the convolutional neural networks.

The next chapter will summarize results obtained in this work, the model deployed in the industrial environment the open issues on the topic.

Chapter 9

Conclusions

9.1 Introduction

The aim of this project was to improve the safety of an automatic command line of an industrial crane analyzing the robustness of the convolutional neural network deployed in the system. During the work, we explored many important topics: i.e. testing phase, robustness improving, network attention, network reduction and verification. All the steps gave useful information that we can be used to define a more robust model.
In this last chapter, we will describe the development process of the model deployed in the production environment. In the final section, we will briefly discuss possible future works on the topic.

9.2 Results

The project gave the opportunity to gain useful knowledge about us to learn very useful knowledge about convolutional neural networks that allows us to train a more robust model.
Testing phase provided us information about the behavior in perturbed situations and highlights some critic cases that must to be managed. For example, the network is not able to manage images completely black or white. From the robustness analysis of the legacy network, we defined the main issues in our case study are the variations in lighting, the little rotations and the lost of the camera focus. These perturbations may be common in an

industrial setting due the harsh environment and in adding to it, they impact significantly on the model safety. For these perturbation we applied the data augmentation technique enriching our data-set. The final data-set counts more than 80.000 images, it maintained the original distribution of empty and full images. Remembering the results in chapter 7 it makes sense use the alternative methods only in case it is not possible use the original data-set, so we use the state of the art method for data augmentation.

From the pruned method analysis, we understood the importance of an accurate chosen of the structure and the dimension of the network. In particular, from the results of chapter 8, we saw that the legacy network was oversized for the original problem. As explained in theoretical background chapter, it's impossible to define a standard configuration for the hyper-parameters. We performed an iterative process training different network structures varying the number of filters in each convolutional layer and/or adding a new convolutional layer. However, the tests showed that the best model is the original one, this time, trained without dropout. We have to remember that the original data-set counted about 6000 images while the enriched one more than 80.000, this explains why the original network architecture fits fine the new data-set.

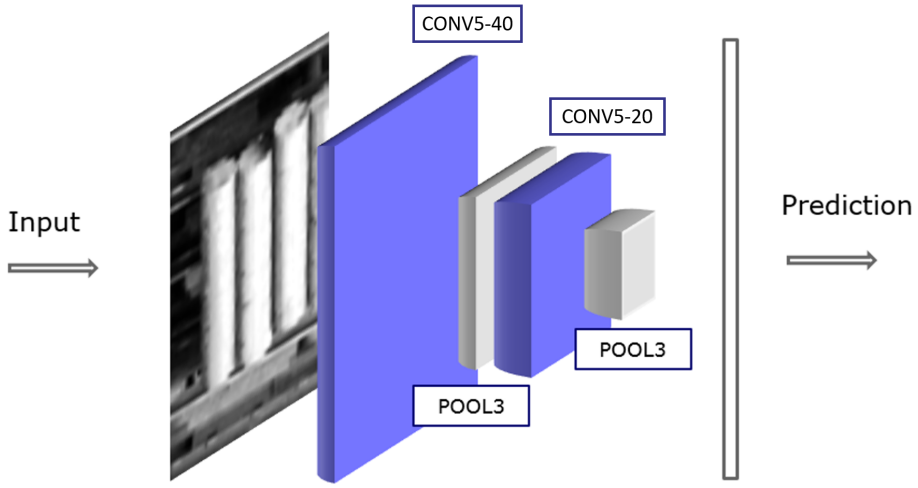


Figure 9.1: Structure of deployed model.

Figure 9.1 presents the architecture of the deployed model. We trained the model several times in order to find the best configurations of the hyper-parameters. Table 9.1

Accuracy	**Precision**	**Recall**
99.54%	99.28%	99.27%

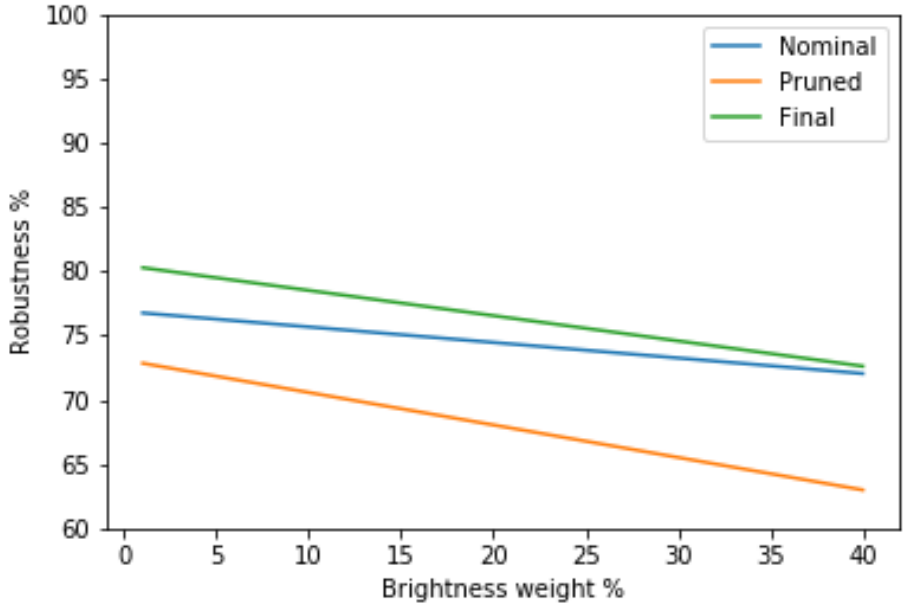
Table 9.1: Deployed model metrics.

presents the metrics of test performed with the testing data-set. We can notice that the nominal metrics lightly decrease their values. We performed the robustness analysis on all the situations described in the chapter 6. The results are described in Table 9.2.

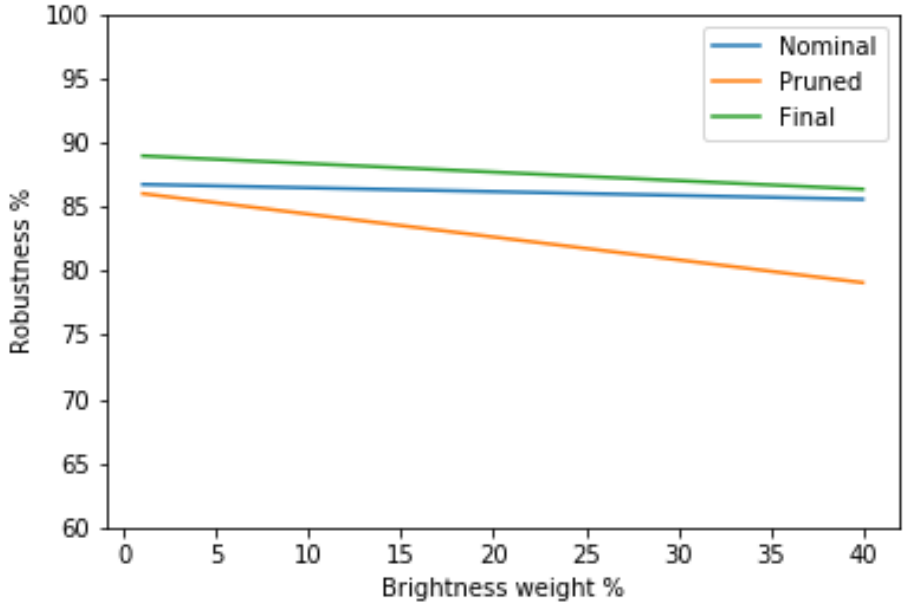
Alteration	**Interval**	**Robustness %**	
		Linear	**Exponential**
Brightness	[0.5, 2]	98.07	99.41
Occlusion 1	[0, 200]	87.61	97.66
Occlusion 2	[0, 50]	56.08	70.82
Occlusion 4	[0, 30]	29.84	43.29
Blur	[0, 2]	99.85	100.00
Green line	[0, 2]	68.19	72.12
Translation Horizontal	[-5, 5]	84.49	99.46
Translation Vertical	[-5, 5]	66.76	97.73
Rotation	[-5, 5]	78.45	92.77
Perspective Left	[0, 15]	82.25	97.47
Perspective Right	[0, 15]	85.42	98.35
Zoom	[-5, 5]	42.22	91.12

Table 9.2: Robustness analysis results.

From the graphs we clearly see we improved the network robustness. The robustness is in the range [73%, 80%] for the linear distribution and in the range [86%, 89%] for the exponential one. From the raw, values we can see how we increased the robustness of blur, brightness and rotation at the expense of the nominal situation and the other alterations: the enriched data-set improved the considered cases, but it has the negative effect to lose part of the capability of generalization.



(a) Generated with linear distribution.



(b) Generated with exponential distribution.

Figure 9.2: Comparison of robustness analysis of nominal, pruned and deployed models.

9.3 Open Issue and Future Works

In this work, we analyzed the world of convolutional neural network into an industrial setting. As we said in the introduction chapter machine learning is a very dynamic and fast world and new work opportunities born everyday. In adding to it, we explored only a tiny part of the machine learning, so a lot of challenges are still opened.

The work described in this document allows us to deploy in the production environment a more robustness and as consequence, more safety model. One open issue concern the position of the camera: in all the work we consider the camera view fixed for hypothesis and we can manage some little movements. What happens in case a big movement? Object detection, with its capability to change the focus areas of the network, can be the new challenge to improve the robustness in great movements of the camera view.

Another interesting topic is concerned the verification: in this work, we described a methodology to verify a simple model as a logistic network, but we demonstrated that our method is not adapted to more complex models as convolutional neural networks. An interesting issue will be studying an alternative technique to verify complex models writing them in mathematical form.

Acknowledgements

This project took eight hard but very gratifying months of work in partnership with the Unibg professor Angelo Gargantini and the Tenaris R&D Data Science team. The work allows us to develop a more robust machine learning model, currently used in the production environment, and provides some useful frameworks and methodologies that may be used in other machine learning projects.
I should like to say special thanks to my family, my friends and all the people that unconditionally support me in my journey started 5 years ago. Special thanks to my supervisor, prof. Angelo Gargantini, that provided this opportunity and helped me along all the work. Lastly, I wish to offer a special word of thanks to all the Tenaris R&D Data Science team, Vincenzo, Andrea, Jacopo and Carlo, that accepted me in their group and aided me with their experience in all the phases of the work.

Bibliography

[1] What is the difference between supervised and unsupervised learning algorithms? *Quora*, 2016.

[2] Raj Bharath. Data augmentation — how to use deep learning when you have limited data—part 2. *Medium Corporation*, 2018.

[3] Alexis Cook. Global average pooling layers for object localization. 2017.

[4] Arden Dertat. Applied deep learning - part 4: Convolutional neural networks. *Towards Data Science*, 2017.

[5] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. 2017.

[6] Arbon Jason. Testing ai: Supervised learning. *Linkedin*, 2018.

[7] Guy Katz, Clark Barrett, David Dill, Kyle Julian, and Mykel Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. *Computer Aided Verification*, 2017.

[8] Andriy Lazorenko. Tensorflow performance test: Cpu vs gpu. *Medium*, 2017.

[9] Assaad Moawad. Neural networks and back-propagation explained in a simple way. *Medium*, 2018.

[10] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *ICLR*, 2017.

[11] Agnès Mustar. Build vgg16 from scratch: Part i. *Machine Learning Journey*, 2017.

[12] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. *IEEE*, 2017.

[13] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. *Feature-Guided Black-Box Safety Testing of Deep Neural Networks*. Springer, Cham, 2018.

[14] Alice Zheng. Evaluating machine learning models. *O'Reilly*, 2015.

[15] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. *IEEE*, 2018.