POLITECNICO DI TORINO

Master degree course in Computer Engineering
Data Analytics

Master Degree Thesis

Deep Learning and Computer Vision based approaches for Airbags deployment analysis by means of user-defined ROIs

Supervisors
Prof. Tatiana Tommasi

Candidate
Matteo ABBAMONTE
ID number: s277483

Italdesign Giugiaro tutors
Luca Jozzo
Nikolas Vinci
Andrea Raballo

APRIL 2022

Summary

Advances in Computer Vision technology paved the way for the development of production pipelines which proved to perform similarly to human operators in terms of accuracy and better then them in terms of time, when applied on repetitive analysis. These performances increase with the introduction of Deep Learning algorithms, which narrow the margin with human level behavior.
This Thesis work focuses on Deep Learning approaches for a Computer Vision application in the Passive Safety checks field. In particular, the description of a tool aimed at checking the correct deployment of airbag devices is provided, delving into the Deep Learning-based detection engine. Experiments on the input data to the Machine Learning model are made, focusing on highlighting weaknesses and biases of the current approach.
Based on the collected evidences, two proposals for approaches to the problems are introduced. While both are based on Deep Learning implementations, the first solution is an updated and reworked version of the previous methodology. The second approach is far from the logic behind the previous ones and is accompanied by the presentation of integration modules and algorithms that ensure the compatibility with the pre-existing data and application. After a detailed description of the aforesaid alternatives, a test suite implementation is reported and then employed for a fair comparison between the default approach and the introduced ones.
The first introduced solution exploits the generalization capacity of the default model, with only few modifications in the field of the training process and output format. On the other hand, the second proposal offers a completely new horizon of solutions aiming at an improvement in the working speed and the simplicity of integration of a simpler Deep Learning task.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Video Media employment

Since the birth of video media and the dissemination of their use at affordable costs, organizations, companies, scholars and other categories of people started employing it in order to add abstraction and speed to some of the processes that, until then, used to impose physical presence to be completed.

- Abstraction factor is given by the reduction of direct involvement of the operator in the actions related to the collection of information for the main goal. By means of video media, these information can be collected asynchronously and with more elastic timing or, if they must be collected in real-time, with a bigger picture corresponding to multiple situations at once (an example could be video surveillance).
- Speed factor can be seen as the capability of an operator to deal with accelerated sequences of actions in an altered time state.

These same two factors can be evaluated even with the introduction of more complex and automated ways for accomplishing the same goal as before or more indirect and abstract ones.

1.2 Computer Vision and Machine Learning

Over time, with the raise of the computational power in everyday devices, more and more technology was implemented as orchestrator for the information collected by means of video media hardware. This led to the application of the Computer Vision coupled with Machine Learning approaches for both the same goals as before and for new challenges such as data mining and event predictions.

- In this case, the Abstraction aspect is given by the further indirection layer that is introduced by means of the computer automation. This takes the place of the previous human operator which is moved to an upper layer of orchestration, enabling the chosen methods to *learn* from the selected data and to extract the desired information from the environment.

- How the processes speed benefits from the introduction of Computer Vision and Machine Learning is straightforward, since the improvement can be obtained by relying on the greater speed that computer-based systems are able to achieve compared to the human being in repetitive actions.

1.3 Thesis domain and collaboration

This thesis work focuses on a study executed over the possible improvements that can be made regarding an existing tool powered by a strong Computer Vision and Machine Learning background component. The tool is owned by the Italdesign Giugiaro company, with whose collaboration it was possible to draw up an analysis of the current workflow of the system, its weaknesses and the main improvements that the current study should address. The described application is based on a patented system [18]

The main problems highlighted by the company can be clustered into two specific domains:

- **Performance in terms of robustness**: the described tool is widely used in the field of a service provided to third party airbag manufacturers. Being the application based on Computer Vision it must be as able as possible to work in situations and standards that are also very different from each other. Given that, the main focus of this collaboration is the improvement of the detection robustness.
- **Performance in terms of time**: even though the tool is not supposed to work with real time video streams, its time performances are fundamental since the amount of data to be analyzed can be huge. Moreover, the application must represent a way to speed up and automate a series of operations that, otherwise, would have to be carried out by a human operator.

The described problems statements are approached in the next chapters, the analysis is distributed in the following way:

- **Tool description**: the existing tool is described from the architecture point of view, focusing on the backend section, which contains the actual ML-based pipeline of the app itself.
- **Detection of biases and problems**: some tests results are presented, considering both expected and non-expected input, in order to highlight the weaknesses of the pipeline that is currently working in production.
- **First approach: Data pre-processing and decolorization**: a data pre-processing step performed in order to achieve a better generalization from the model used in the current version of the tool is described, accompained by a comparison with the previous version.
- **Second approach: Classifier with forced attention**: another approach with respect to the current one is presented, together with the description of the architecture of the selected model, the pre-processing phase of the dataset and the training algorithm.
- **Test suite and comparison**: a set of automated and manual tests is described, useful for the comparison of the approaches exposed in the Thesis work.

Chapter 2

Tool description

As announced in Section 1.3, the tool has been adopted by Italdesign Giugiaro company in the field of a service provided to various airbag manufacturers worldwide. The application is called ***AI^{rbag}***, in order to recall the link between the object of the analysis made by the tool and the Machine Learning based backend (which reference is represented by the "base" of the expression in the name, that stands for Artificial Intelligence).

AI^{rbag} is part of the improvements given by the digitalization and automation of the processes actuated in Italdesign's business units. As a company operating in the Automotive field, one of the main lines in IG's Research and Development operations is Passive Safety. The conjunction between this research line and the digitalization and automation process led to the need for a new tool (among others) that automates the process of analysis of videos depicting the deployment of test airbags in certain conditions. These conditions may consist of either a fixed frame or the copy of a vehicle interior, both including a steering wheel or another container equipped with an airbag.

In order to be as consistent as possible with the official aim of the original application, in the following, a quote from the patent description [18] is reported:

> In summary, the present invention is based on the principle of carrying out a method to verify the correct deployment of an airbag device by means of a computer, programmed to select or identify at least one investigation region in at least one image analysis area, where “select” means an embodiment wherein the region is chosen a priori at a given position in the image, “identify” means an embodiment wherein the system automatically recognizes in the image a relevant area, regardless of its position in the image. In the investigation region, at least one portion of the region representative of the air chamber of an airbag device deployed at one or more measurable points in times since the triggering of the deployment (inflation) of the air chamber is identified. The investigation region is a two-dimensional region or a one-dimensional line.

In the next section, an overview of the tool is provided, starting from the description of how the application works and which operations are possible, then focusing on the architecture of the application, on its Machine Learning backend and on the description of the actual Neural Network model and training process.

2.1 App functioning

In order to make an introduction to the jargon used within the application domain, in the following some acronyms or words are highlighted:

- A **Job** is an object that contains all the information required to activate the pipeline of operations (called **Run**) on a single or a set of videos. In particular, a Job includes a name, the video input path that is used once that they are loaded, the video output path and a *mode* chosen between the two provided in the UI. Each Run is saved for each Job and a recap is showed in a dedicated page.

- **ROI** is the acronym for *Region Of Interest*: it corresponds to each single polygonal section that a user can select on a set of frames belonging to a video.

- In the context of this Thesis work, any reference to the word **Detection** should be interpreted as the operation of recognition of the presence of an airbag in a user defined set of *ROIs*.

The main functions and features offered by the described tool are reported in the following flow of operations, where each step is deliberately described abstractly, since the interaction with the graphical interface is not the main topic of this Thesis.

The first fundamental operation that is allowed to the user by the UI is either the creation of a new *Job* or the selection of an older one from a dedicated interface, that reports the latest *Runs* and their corresponding *Jobs*. In the case the user creates a new *Job*, she has to load a new video or set of videos and provide all of the details reported in the acronym list, otherwise the video should be already available in the interface, showing the elements and options regarding the selected *Run* among those available for a *Job*. For each new *Job* or *Run* for an existing one, the user has to select or modify some *ROIs*. In particular, there are at least two necessary *ROIs* to insert, corresponding to the one that highlights the name and the temperature condition of the video sample and to the one that contains the increasing timestamp for each frame of the video. These *ROIs* will be analyzed by means of a *OCR* (Optical Character Recognition) tool if fired by the detection algorithm. In addition to these *ROIs*, a user can select any number of polygonal regions within the images borders; the aim of the selection is to let the algorithm perform the *Detection* of the airbag through either the whole video or a set of user-defined frames. After the *Run* command is activated, all the information is sent to the backend that performs the detection and returns a set of details that are collected in a report presented in the UI. In particular, for each polygonal *ROI*, the report contains the frame that fired the detection (together with some other frames around it), the timestamp and the frame number of the detection itself. Moreover, the report contains the name of each video (corresponding to the characters recognized by the *OCR* system) and some other information regarding each airbag deployment, such as the frame where the bag reaches its maximum extension in case the selected Run category is *Passenger* instead of *Knees*.

2.2 App architecture

With the aim of keeping the tool as compatible as possible with various environments (such as a clustered architecture), each stack is deployed by means of some Docker containers. The following subsections dive into the details of the Airbag stack and the Keycloak stack, describing each of the containers that are part of them.

2.2.1 Airbag stack

The *Airbag stack* contains the actual client-server application code, including a database container dedicated to the backend work. In the following, each relevant container is presented and described.

mongo container

The first container in the dependencies chain is the *Mongo container*, that is based on an official image[1] from the MongoDB repository on DockerHub and it works as a simple and fast database service for the server section in the backend of the application. More precisely, it handles all the information regarding each *Job* and *Run* belonging to every user of the tool.

ms-airbag container

Working as a middle layer between the MongoDB database, the client interface and the actual ML-based pipeline of the application, the *ms-airbag container* implements all the APIs that enable the communication with the clients. This communication includes the run command that is activated by the client (equipped with all the information needed for the ML-based pipeline to perform the detection), the dismiss command, the updates coming from the ML-based pipeline regarding the analysis status, the history of the Jobs for each user.

One of the main set of operations mediated by the *ms-airbag container* is the file management: the described section of the application has access to the storage through some ad-hoc mounted volumes. The storage contains a couple of *input* and *output* folders and a *Job* folder: the former two contain all the input videos and the output data of the analysis, grouped by user and Job name, the latter one works as a temporary storage for the intermediate data generated for each run of a Job.

The described container is based on a custom Docker image and the actual server code is written in Java language.

fe-airbag container

The UI of the application is provided to the client by means of the *fe-airbag container* that enables the work of the front-end section of the tool. The relevant interfaces are in the form of an history report regarding the Jobs and runs after the login page, a summary regarding the available resources on the cluster, a page containing a form that has to be

[1]https://hub.docker.com/_/mongo

filled for a new Run. The latter interface is used in two cases: a new Job is created with the loading of a video or an older Job is used as base for a new run. In the context of the described page, the form should be filled in order to provide the name for the Job and for the input and output folders; moreover, the interface provides the possibility to load a video or a set of videos in the same page and, for each of them, the user can add new ROIs by manually drawing polygons on the preview of the video and adding names and colours to distinguish them. After pressing the run command and waiting for the process to complete, a cell is added to a successful Runs list. After this condition, if the cell corresponding to the result is selected, a modal pops up, showing the details contained in the report of the analysis.

nginx-video **container**

As well as the *ms-airbag container*, also this section has access to the storage containing the input, output and Job folders. As a difference with the aforementioned section, the *nginx-video container* is used as a load balancer that reflects the operations of the clients, in order to complete the loading process of new videos or get information and results regarding the Jobs.

2.2.2 Keycloak stack

The actual application code does not include any implementation related to an authentication process. Indeed, this task is delegated to the *Keycloak* server, working as Identity and Access Management solution (as reported in the official website of the service[2]). In particular, the *Keycloak* server provides the entire login process and interface, by using cookies technology for the token exchange with the client. The actual server is deployed in a container and it is based on an image from DockerHub[3].

The *Keycloak* server works by using a dedicated database service based on *MySQL*, which runs in another container, based on an official DockerHub image[4].

2.2.3 AirbagDetection container

Every reference to the ML-based pipeline in the description of the *ms-airbag container* in Subsection 2.2.1, regards the detection process implemented in the *AirbagDetection container* (that is called *Detection Server* from now on for simplicity). Communication between this container and the one running the main server (*ms-airbag*) is made possible through a further API layer. The Detection Server code is entirely written in Python language, hence the RESTful compatibility is provided by a Flask service[5] running a light and custom Web-Server. The detection process consists of a pipeline of operations, including the estimation of the airbag position (made by a Convolutional Neural Network)

[2]`https://www.keycloak.org`

[3]`https://hub.docker.com/r/jboss/keycloak`

[4]`https://hub.docker.com/_/mysql`

[5]`https://flask.palletsprojects.com`

and the actual ROI content evaluation (made by means of some thresholds in a subsequent algorithm). A detailed explanation of the mentioned pipeline is provided in the next section.

2.3 Video analysis pipeline

The main aim of this Thesis work is to improve the *Detection* performances. Hence, since the practical implementation of this process is entirely contained in the Detection Server, this section provides a detailed description of the sequence of operations aimed at the generation of the information needed for the creation of a complete report, starting from an input video.

Video frames extraction

The first step of the pipeline is the reception of a new Job object from the Main Server (running in the *ms-airbag container* and described in Subsection 2.2.1) that contains all the information needed to perform the detection, including the position of the video (or set of videos) in the storage. If the Detection Server is busy with other Jobs, the current one is queued in order to be processed as soon as possible.

Each video belonging to a Job is subjected to a frame extraction phase: more specifically, by means of the FFmpeg framework[6], all the frames belonging to the video are isolated, renamed by ascending order and saved to a dedicated temporal folder in order to be used as input for the subsequent phase.

Airbag segmentation

The folder that is filled by the previous step should contain an ordered sequence of RGB images, corresponding to the frames of the input video. With this premise, each generated image is taken as input for an inference process made by a previously trained Deep Convolutional Neural Network. The DCNN performs a *binary segmentation* process, meaning that the resulting images are composed only of pixels set to *one* or *zero* and, if superimposed on the original frames, they should highlight the position and shape of the airbag, if it is present. A deeper explanation of both the Network architecture and its purpose is provided in Section 2.4.

Detection

The result of the segmentation process can be described as a set of binarized images that is identical in number and single frames naming to the input set. This set of images is then subjected to the actual detection phase: if the frame is included in the user-defined interval for the detection of the content of a specific ROI, the polygonal region corresponding to the ROI itself is extracted from the segmented image and the ratio between the active pixels and the total amount of pixels in that region is compared to a set of empirically defined thresholds. If the user choice is to detect the frame where the boundary of the

[6]https://www.ffmpeg.org

airbag enters the ROI then it is enough that the previously calculated ratio exceeds 0 (plus a small margin for preventing noise presence); otherwise, if the choice is to select the frame where the ROI is completely covered by the airbag surface, the ratio must be closer to 1 to fire the detection. The described algorithm is repeated for each ROI and for each segmented frame, if the frame is part of the interval that has been selected for the aforementioned ROI. Also in the regards of this phase further details are presented in another section (2.5): it is worth to dive into the details of the real algorithm since it is part of the improvement attempts presented in the next chapters.

Report generation

At the end of each detection attempt, a tuple containing the name of the ROI and the number of the frame that fired the detection is saved and, at the and of the overall process, this collection of tuples is dumped into a JSON file. Moreover, for each tuple containing ROI name and detection frame, the Detection Server generates an image containing the two frames preceding the detection and one following it. These elements are then showed in the UI, after being sent to the client by means of the Main Server.

2.4 Machine Learning approach

In Section 2.3 the sequence of operations that leads to the final objective of the tool was presented, which is the detection of areas covered by airbags or containing edges of it. The purpose of the current section is to dive into the details of the ML-based step, which is the *Airbag segmentation* one, describing the data that are used to feed (and train) the Convolutional Neural Network, the architecture of the CNN itself and its training algorithm.

2.4.1 Input and training data

In the context of the explanation of the characteristics of the data, a distinction must be made between a production situation and the training one. The training process is supervised, meaning that, in order for the network to learn, the process must include an input element and an ideal output one, to be compared with that predicted by the network.

Both in the production phase and in the training one, the Network accepts squared RGB images as input. In a production case, these images are extracted from a video loaded by the user, thus depicting the evolution of an airbag deployment, starting from the inside of a closed bag or a steering wheel. Normally, the hooks for the test airbags are placed on special frames with a single color background that can contain grids with lines at various distances; in some cases, instead of the aforementioned fixed frames, the devices can be supported by realistic reproductions of cockpits. Moreover, the lighting characteristics, the shapes of the frames, the position of the camera and the angle of view are not standardized in this type of videos, thus ending up having many different video conditions.

In a training case, the network is fed with frames coming from a custom dataset. The original dataset contains 565 frames extracted from a set of sample videos but a split has been applied on it, taking 452 samples for the actual training and 113 for the validation phase. Looking at the frames it is possible to underline that every stage of the airbag deployment is depicted multiple times, providing good generalization in that regards. On the other hand, being the dataset custom made, it shows a low variety of color combinations, backgrounds and viewing angles and this is not a good aspect in the context of training a network that should be employed for production use with samples that can be completely new from the previously described factors side.

For each frame in the dataset, there is its segmented counterpart that is used in the training algorithm. These binarized versions of the frames are characterized by a single channel where each pixel can assume either the value 0 or the value 1. For this reason, in the context of the training algorithm they are called *masks*.

2.4.2 DCNN architecture

The aim of the *Airbag segmentation* is to obtain a binarized mask as output of the employed network. Hence, the architecture has been chosen to be compliant with the capability of obtaining a result that does not suffer from resolution loss, in order to prepare data that is as detailed as possible for the subsequent detection phase.

The implemented approach is based on the results obtained by Romera et al. in their ERFNet [17]. This particular approach showed an optimal trade-off between the accuracy in the predictions and the corresponding performances in terms of inference time in the field of *Semantic Segmentation*.

Semantic Segmentation application

In order to better understand how the existing approach fits the problem it is useful to review the objective of the task that takes the name of *Semantic Segmentation*.

The aforementioned task is one of the steps towards the goal of the complete *Scene awareness*; in this sense, as stated in [8], it is the evolution of early tasks such as classification, localization and detection, aiming at finer-grained inferences on images and videos. *Semantic Segmentation*'s actual objective is to classify each pixel of an image (or sequence of images) as belonging to the object or are of which it is part, adding important information with respect to the simple image classification task, such as spacial location, scale or position of the subject itself.

In the original version of the AI^{rbag} tool, the detection phase is based on segmented images generated by the previously mentioned DCNN, which addresses a Semantic Segmentation task. This task enables the detection algorithm step to have a version of the original image that is transformed in order to contain only useful information regarding the airbag, thus providing its extension, spacial position, and shape.

Deep ConvNets for semantic segmentation task

At the time of the first implementation of the tool via a ML-based pipeline, in order to pursue the goal of keeping a near real-time performance of the pipeline, the ERFNet solution (by Romera et al. [17]) has been chosen. The acronym ERFNet stands for *Efficient Residual Factorized ConvNet*; the project fits into the approaches for the field of the urban perception achieved by Intelligent Vehicles.

In the general context of the scene perception obtained by means of simple photographic sensors, the need for the combination of dense pixel-level details and a more abstract and conceptual understanding of the content highlighted the capabilities of solutions based on convolutional neural networks. Deep implementations of these algorithms have already proven to outperform many traditional solutions in both simple and complex classification tasks and the same observation can be made regarding the semantic segmentation task. These performances are obtained leveraging on the capability of these networks to learn from the conceptual combination of spatially local information, based on spatial coordinates used as original input [13]. The depth of this network plays a fundamental role in the abstraction of elements coming from different patches of the input image, in a sort of pyramidal flow.

When used for classification purpose, however, these networks usually reduce the output dimensions along their depth in order to increase the density of the encoded information and reduce computational requirements. This assertion, together with the goal of a semantic segmentation task being the generation of an output image that equals in its size and proportion the input one, forces the architecture of a deep convolutional neural network aimed at this purpose to include a way for upsampling the encoded version of the input, in order to perform both *end-to-end* training and inference. Different approaches exist for the upsampling operation: the main goal is to bring a coarse output of the encoder network to a segmented representation of the original input, without loss of resolution, proportions and details.

The first and simplest solution to the aforementioned problem is upsampling via interpolation: this process applies fixed kernels to the output of the encoded data (or feature extraction phase) so that the result is upsampled through the usage of the information given by the neighbourhood for each pixel. The drawback is that the employed kernels cannot be learnt and the output usually suffers from lack of high frequency elements and fine details.

An interesting solution has been presented by Shubhra Aich et al. in [1]. The idea is to regain spacial information from the final feature maps resulting from the encoding phase. They name the described operation a *depth-to-space spacial reordering* and managed to overcome the problem of computational requirements given by heavier and more complex solutions while keeping comparable performances in binary segmentation problems.

A first step toward the solution that has been implemented in ERFNet is presented in [13]. Long et al. introduce a method for reproducing a solution that is similar to the usage of a simple interpolation except for the content of the kernel that can be learnt during the *end-to-end* training process. The proposed architecture implements a *Deconvolution layer* that, as a result, implies the upsampling operation by a factor that, in a usual convolution operation, would have been considered as downsamplig factor (which is the stride factor).

Starting from the presentation of the aforementioned approach [13] many advances have been made both from the point of view of the optimization of the deep convolutional networks architecture and from that of deconvolutions employment as a method for upsampling the encoded data. One of the main improvements in these fields is the introduction of *residual blocks* in the so called ResNets [11]. The novelty in this approach lies in what each single block is expected to learn, since in a usual stack of layers the complex function to be approximated is an unreferenced mapping between the input and the output of the stack itself. Instead, a residual block is expected to fit a residual function between the output and the input of the stack. The output vector of the block becomes:

$$y = F(x, \{W_i\}) + W_s x \tag{2.1}$$

where $F(x, \{W_i\})$ is the actual residual function that has to be approximated, W_s is usually an identity mapping for the input x. This modification easily addresses the degradation of the training performances on deep architectures while speeding up the convergence.

Leveraging on both the described residual blocks and the deconvolution ones as upsampling methodology in the semantic segmentation field, deeper and more elaborated network architectures have been employed also for real-time urban scene awareness on Intelligent Vehicles and it is in this regard that the need for a trade off between complexity and performances comes in. Figure 2.1 shows the standard approach for the realization of a residual block (a), as well as the bottleneck architecture (b) from Long et al., which actually reduces the need for computational resources as the depth of the network increases. The bottleneck solution, however, is demonstrated to be less efficient in taking advantage in the increasing depth with respect to the non-bottleneck approach [19], but with the advantage of being less resource intensive due to the lower number of parameters involved.

ERFNet for fast and accurate segmentation

Based on the described problems regarding deep convolutional neural networks and their application for semantic segmentation tasks, Romera et al., in their ERFNet, present a new residual block that keeps the advantage of the basic ones in terms of the *perceived* kernel size while dramatically reducing the number of parameters and, hence, the training time: Figure 2.1c shows that the novel approach is given by the factorization of a basic convolutional layer with squared kernels into a couple of simpler ones using a single dimensional kernel.

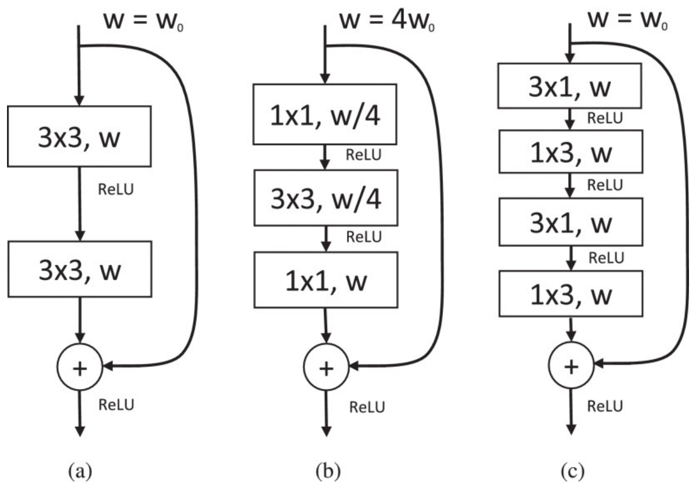


Figure 2.1: Taken from [17], Representation of the basic residual block (a) and the bottleneck one (b) described in the work of Long et al. [13], together with the non-bottleneck-1D version introduced by Romera et al. (c)

ERFNet is developed for semantic segmentation purpose and this is the reason why it has been used in the first production version of the AI^{rbag} tool. It leverages on the proposed non-bottleneck-1D residual block both in its *Encoder* branch and in the *Decoder* one. The Decoder section, in particular, presents a joint use of deconvolutions and non-bottleneck-1D blocks aiming at upsampling the output of the Encoder while returning finer details.

2.4.3 DCNN training algorithm

The original version of the network released by Romera et al. is provided with pre-computed weights both for the Encoder only (that is pre-trained as a classifier on ImageNet dataset [5]) and for the end-to-end architecture, providing near state-of-the-art performances for the time, in terms of *Intersection over Union*, on the Cityscapes dataset [4]. Although these weights can be useful as network initialization, they are not good for the segmentation task on the custom dataset. Hence, the provided training algorithm and the dataset loading class have been modified to fit the current case. In the description of the following details regarding the code, the use of the framework for machine learning Pytorch [15] will be considered implied.

Airbag dataset loading class

Even in the original implementation of the code[7] provided with the paper [17], for each employed dataset there is a dedicated class that is used for actually reading images and their labeled versions from the storage. A similar class is made available for the custom dataset, involving the creation of the list of names corresponding to each frame contained either in the training section or validation one (the section can be chosen by means of a simple argument in the dataset object allocation). The dataset objects includes a public function that is called by the data-loader for the reading operation; each image is then eventually transformed by means of a dedicated function. The transformation class contains two paths that are diversified by the fact that one of them includes a simple *data augmentation* process, in case of application on training images. The data augmentation involves some random center crops, translations, horizontal flips, border additions and other modification based on contrast, brightness, blur and color. All images are originally loaded with pixels having values from 0 to 255 and another task of the transformation function is to bring these values in the [0,1] interval, *without* per-channel normalization.

Weighted Loss function

The Loss function employed for the training is a Negative Log Likelihood that takes as input the batched results of the network prediction and the targets that, in this case, are equivalent to the masks described in Subsection 2.4.1. In particular, the results of the prediction are first submitted to the Pytorch implementation of the LogSoftmax function and then used as the first argument of the actual NLLLoss (Pytorch implementation for the Negative Log Likelihood loss).

[7]https://github.com/Eromera/erfnet_pytorch

The described approach is practically equivalent to the one that directly makes use of the CrossEntropyLoss, as reported in the documentation related to the NLLLoss[8].

Moreover, the loss is weighted by means of some custom weights: their values are only estimated on the base of the first 10 loaded samples out of the total 452 training ones and the calculation is executed at runtime. The computation of these weights involves a small loop that precedes the beginning of the training phase, aimed at collecting the first 10 segmented masks in a tensor. Starting from the resulting tensor, the ratio between the number of pixels respectively set to 1 and 0 is used in the calculation via the weighting strategy introduced by Paszke et al. [14].

Training procedure

Based on the previous details, the actual training algorithm is quite straightforward. An important element is the *optimizer*: the actual optimization function is based on the Pytorch implementation of the *Adam* algorithm[9]. When instantiating the optimizer, it is given an initial learning rate of 5^{-4} and a weight decay of 1^{-4}, while the other parameters are left at their default values. As in the original work by Romera et al., the provided learning rate is not fixed and its value is subjected to a scheduler that follows a linear function with negative slope up to a number of epochs close to the final one, beyond which the slope itself decreases with increasing epochs.

The default dataloader is set to work with a batch size of 6 samples and it involves a shuffling operation before each training epoch. For each epoch, the algorithm keeps track of different indexes: as usual, in a training algorithm, the per-epoch average training and validation losses are collected, together with the overall per-epoch training and validation time, which are used for estimating a per-image training and inference time. The main metric that is used for evaluating the trend of the training and choosing the best epoch between the various saved checkpoints is the *Intersection over Union* (IoU). This metric is frequently used in the field of the semantic segmentation and its based on what, in statistics, is called the *Jaccard Index*. As a statistical index, it is used to measure the similarity between two finite sets, so, regarding the semantic segmentation field, it is widely used for obtaining a percentage of the similarity between the output prediction and the target mask. As a formula, it can be computed as follows:

$$IoU = \frac{target \cap prediction}{target \cup prediction} \Rightarrow \frac{TP}{TP + FP + FN} \tag{2.2}$$

where the intersection between the target mask and the prediction can be seen as the amount of pixels that are correctly predicted (True Positives) and the union between the same elements as before can be seen as the total amount of correctly predicted pixels and the wrong ones (*False Positives* + *False Negatives*).

For each batch in a epoch, the result of the prediction includes the same number of samples as the input one, but having, for each single predicted segmentation, 2 channels, corresponding to the respective probability of each pixel being 0 or 1. This output is directly given as input to the loss function, together with the target mask that only has

[8]https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html

[9]https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

a single channel containing both zeros and ones. For the calculation of the IoU metric, instead of using the predictions as they are generated, each segmentation is "flattened" by means of the application of a *argmax* function that extracts the index of the channel with the higher probability.

After each epoch the algorithm automatically saves the current version of the weights in order to make possible the resuming of the training process.

In general, being the network composed of two branches, namely Encoder and Decoder, the training process can be operated either on the entire structure at once or in a two steps process, meaning that the Encoder can be trained as initialization phase and then the *end-to-end* process is carried out, being the propagation of the error damped with increasing layers. However, in the first case, the Encoder can be initialized with the weights provided by the creators of the official ERFNet project, thanks to a training process on ImageNet with classification aims.

2.5 Detection algorithm

As described in Section 2.3, the segmented images, resulting from the inferences of the neural network, are subjected to a further algorithm that extracts the relevant information from the user-defined ROIs, based on an analysis made by considering some fixed thresholds. In the following, the aforementioned algorithm is described in its details, following 6 points in reference to the comments in the Algorithm 1.

1. Before the algorithm starts, it is necessary that some elements are provided: the user-defined ROI list is needed since there are various loops involving each ROI object, moreover, the first and the last frame indexes are used to limit the iterations. The first steps consist in instantiating two elements: the *full_rois* matrix, which will be filled during the first loop, and the *detections* dictionary, which is subsequently filled with keys corresponding to the names of each ROI.

2. The first outer cycle is based on covering each frame index between the ones that have been provided regarding a specific video. For each frame index, the corresponding segmented image is loaded from the mounted volume and transformed into a binary mask (zeros are False cells and ones are True cells). Given the binary mask, an algorithm for removing small elements floating in the matrix is employed, the corresponding function is given a connectivity value of 2 pixels (corresponding to the neighbourhood evaluation of each pixel) and it is provided by the *morphology* module in the *Skimage* library[10]. With the segmented frame that has been "cleaned" from any outlier pixel, it can be merged with a mask that is created by means of a custom function: the method involves the creation of a global mask filled with True cells, putting the pixels that are not part of any ROI to False and then multiplying the obtained mask with the clean segmented frame.

3. As stated in the Point 1, during the first round in the outer loop the matrix called *full_rois* is filled in a similar way to how the ROI mask has been applied on the

[10]https://scikit-image.org/docs/stable/api/skimage.morphology.html#skimage.morphology.remove_small_objects

segmented frame. This time, however, the ROI application is done on a temporal matrix that is filled with True values, meaning that this process aims at obtaining a counterpart of the last version of the segmented frame of the previous step that has every pixel inside the ROIs set to True.

4. Within the outer loop there are two inner ones, which are non-nested between them, both cycling over the user-defined ROIs. The first inner loop fills a dictionary containing the reference percentage thresholds that are used in the subsequent comparison. In particular, with reference to the number of pixels in a ROI, the threshold for considering it as completely covered by the airbag is set to 99%, while the threshold for the detection of a simple airbag boundary is set to 5%. Subsequently, if in the ROI object there is no reference to the type of detection that must be made, it is given as a default type the *boundary* one. Based on its type (*all_area* or *boundary*), the ROI object is equipped with the corresponding percentage threshold value, taken from the dictionary previously created in the current step.

5. Remaining in the body of the same loop as in the previous step, the only thing that is left is the calculation of the threshold over which the detection is activated in terms of number of pixels. This is done by extracting a box from the *full_rois* matrix that bounds the current ROI coordinates and then by counting all the pixels that are set to True. This number corresponds to the measure of the area of the current ROI and it is used for calculating the number of pixels needed for the detection multiplying it by the percentage threshold that has been set in the previous step. The described operation is the last one in its loop.

6. After the calculation of the detection threshold in terms of number of pixels for each ROI, another loop, involving again the various user-defined ROIs, is used for actuating the detection. In particular, a check on whether the current frame index belongs to the interval of interest for the current ROI is done, in order to continue with the actual detection. With the aim of counting the active pixels in the current ROI, the box that bounds it is extracted from the segmented frame and, as it has been done in the previous step for the entire ROI area, a sum on all the pixels set to one is performed. If the count of the active pixels in the box containing the ROI extracted from the segmented image is greater than the activation threshold calculated for the same ROI, the index of the current frame is added to the *detections* dictionary that is instantiated at the beginning of the algorithm and the last inner loop body is terminated.

The described process is repeated until either all the ROIs have been analyzed for each frame or a detection instant for every selected ROI has been detected. The first case means that there can be ROIs for which the algorithm has not detected any relevant event.

Algorithm 1 Detection algorithm

```
Require: $roi\_list$, $start\_frame$, $end\_frame$                          ▷ (1.)
  $full\_rois \leftarrow NULL$
  $detections \leftarrow \{\}$
  for each $roi \in roi\_list$ do
      $detections[roi[name]] \leftarrow NULL$
  for each $t \in$ range(start_frame, end_frame) do
      $seg\_frame \leftarrow$ read_from_memory($t$)                           ▷ (2.)
      $seg\_frame \leftarrow seg\_frame > 0$
      $seg\_frame \leftarrow$ remove_outliers($seg\_frame$, $connectivity = 2$)
      $seg\_frame \leftarrow$ apply_polygonal_mask($seg\_frame$, $roi\_list$)
      if $full\_rois$ is $NULL$ then                                         ▷ (3.)
          $size \leftarrow seg\_frame$.shape($1$)
          $full\_rois \leftarrow$ draw_rois($roi\_list$, $size$)
      for each $roi \in roi\_list$ do
          $detection\_thresholds[all\_area] \leftarrow 0.99$                 ▷ (4.)
          $detection\_thresholds[boundary] \leftarrow 0.5$
          if $roi[type]$ is $NULL$ then
              $roi[type] \leftarrow$ "$boundary$"
          if $roi[perc\_threshold]$ is $NULL$ or $roi[type] =$ "$boundary$" then
              $roi[perc\_threshold] \leftarrow detection\_thresholds[boundary]$
          else if $roi[type] =$ "$all\_area$" then
              $roi[perc\_threshold] \leftarrow detection\_thresholds[all\_area]$
          $roi\_area \leftarrow$ extract_box($full\_rois$, $roi[coordinates]$).sum()   ▷ (5.)
          $roi[act\_threshold] \leftarrow roi\_area \times roi[perc\_threshold]$
      for each $roi \in roi\_list$ do                                        ▷ (6.)
          if $roi[start\_frame] < t < roi[end\_frame]$ then Continue
          $active\_pixels \leftarrow$ extract_box($seg\_frame$, $roi[coordinates]$).sum()
          if $active\_pixels > roi[act\_threshold]$ then
              $detections[roi[name]] \leftarrow t$
```

Chapter 3

Detection of biases and problems

Based on the stock version of the tool, the first experiment that is described in this thesis work regards the evaluation of the performances of the ML-based approach. The main goal of the current chapter is to focus on some types of input, feeding the employed segmentation network alone both with a set of expected inputs and preprocessed images. The aim is to compare and analyze the segmentation results, obtained by means of different types of input. In particular, the following sections focus on:

- **Expected input**: the input frames are extracted from a demo video and used to directly feed the segmentation network.
- **Pre-processed images**: the expected samples are pre-processed in the following ways before being used to feed the segmentation network:
    - **Decolorization**: the images are modified in order to remove contrasts and colours.
    - **Motion detection**: on a sequence of images belonging to the same video, each frame is transformed in order to highlight only the pixels that have different values with respect to the previous frame.
    - **Background removal**: on a sequence of images belonging to the same video, the common background is removed.

3.1 Expected input

As described in Section 2.4.1, the expected input for the ERFNet that is employed in the stock version of the tool accepts RGB squared images. In this section the algorithm is tested with frames extracted from a sample video[1] depicting the deployment of an airbag. The video, while not portraying a perfectly professional test context, is still acceptable for the purpose of this experiment, being recorded using a professional high speed *pco.dimax*[2] camera at 4500 frames/s, made by the company Excelitas PCO.

The frame extraction is performed by means of the FFmpeg framework which is the same method that is used in production mode, as reported in Section 2.3. The results are

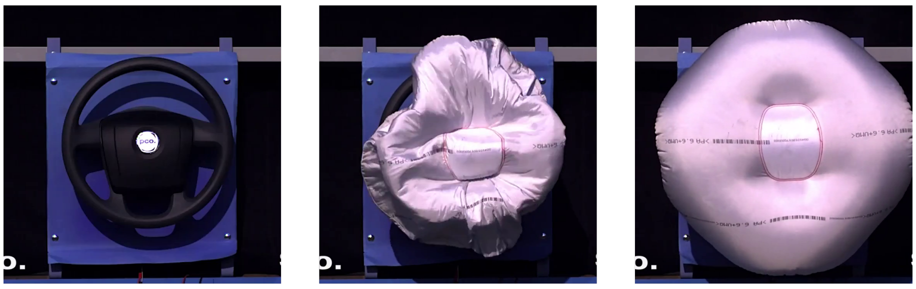
[1] https://www.youtube.com/watch?v=TuHK3mNNMLk&ab_channel=ExcelitasPCO

[2] https://www.pco.de/highspeed-cameras/

then analyzed, looking for any element that could represent a consequence of a bias that affects the prediction.

Both when integrated in the tool and in a stand-alone condition, the segmentation algorithm does not involve any transformation or normalization, since they are not applied even during the training process. The only operation that is performed to each single frame is resizing it to the resolution of 768×768 pixels, this being the one used during the training. After the actual call to the forward method of the segmentation network, the resulting binarized images are multiplied by 255 in order to return an image that can be distinguished by a human when saved in the segmentation folder.

With the aim of having an idea of the characteristics of the predictions on all the phases of the video, Figure 3.1a shows three stock frames.


(a) Expected input.

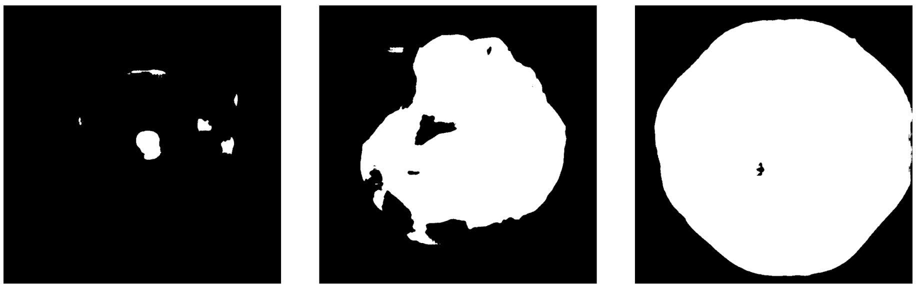

(b) Segmentation based on expected input.

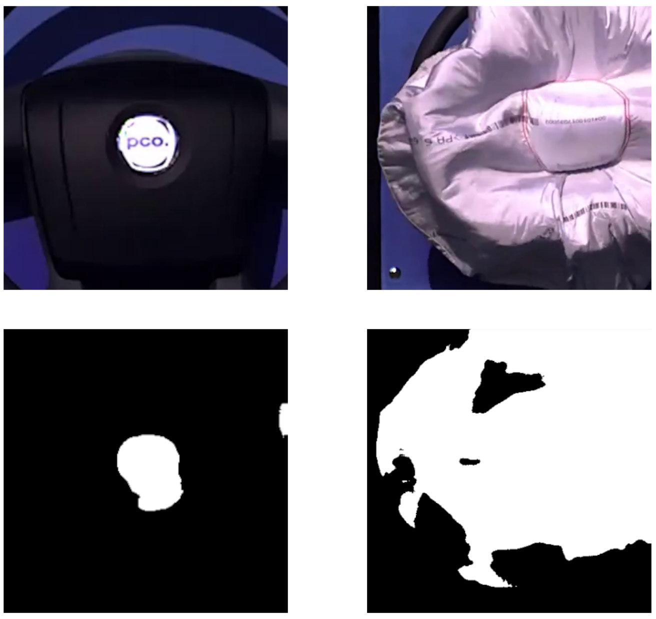
Figure 3.1: Sample frames extracted from a video of an airbag deployment, recorded with a high speed camera (a), together with the segmented images obtained by means of the default model (b).

In Figure 3.1a, the first frame depicts an initial situation, before the deployment of the airbag and, hence, without any section to recognize as containing interesting pixels. The second frame contains the airbag during its deployment phase: in this situation the airbag is characterized by many shaded areas due to the unfolding of its surface during the inflation. In the third image the airbag looks completely deployed: its surface is quite smooth and the predominant color is the white of the fabric, with some light reflections due to the lighting.

In Figure 3.1b, the results of the segmentation are showed by following the same order as the input frames. According to the description of the first input frame, the corresponding segmentation mask should be completely black, without showing any white pixel. However, the result is quite different with respect to the ideal one: the first segmented image presents several positively labeled areas in locations that correspond to reflections and high-contrast elements in the original frame. In particular, in the exact center of the first binarized frame it is possible to recognize a circular shape that corresponds to the logo in the same position of the steering wheel, if looking at the original version. In addition to the central logo, also some other reflections on the steering wheel rim mislead the prediction.

Looking at the second prediction, despite the fact that the starting image is objectively complicated due to the high number of folds both at the edges and on the foreground surface of the airbag, the result is quite confusing. In particular, the same reflections that fooled the segmentation in the first frame influence also the second prediction, with the addition of some other artifacts due to the fact that the airbag joins some areas of the fixed frame where the color of the frame itself is similar to the airbag one. Moreover, the inner part of the predicted section should be homogeneous and should not contain any holes but this is not the case, since, in addition to the jagged edges, there are also some inner areas that are predicted as belonging to the background.

For what concerns the third prediction, the result is quite convincing but, between all the possible situations, the one corresponding to a fully inflated airbag with good light condition is the easiest possible for the segmentation model to work. The result does not contain any relevant hole or jagged edge and there is no other positively labeled section except for the airbag one, if comparing it with the original frame.



(a) Logo area. (b) Shaded areas.

Figure 3.2: Focus on the false positive predictions around the logo area (a) and on the false negative ones corresponding to some shaded airbag sections (b).

From the described attempt it is possible to extract the reason for one of the experiments that are described in the following section. The common factor in all the predictions presenting errors and artifact is the fact that these errors derive from some particular color combinations and high contrast areas. Figure 3.2a focuses on an example supporting this hypothesis: the high contrast value of the reflection generated by the logo in the center of the steering wheel and its central position induce the model to consider it as a part of the airbag. The complementary problem can be recognized for the shaded edges of the bag in Figure 3.2b. Another factor that enforces the color combination and contrast problem is given by the tendency of the network to blend together correctly classified pixels and neighboring areas that have similar colors to the surface of the airbag.

This being highlighted, in the next section an attempt to bypass the yet described problems is presented without retraining the network, in order to enforce the reported conclusion.

3.2 Preprocessed input

In the following, three pre-processing approaches are described. They are meant to test both the generalization capacity of the network and some approaches for isolating the subject of the detection in the input frame. Moreover, in order to keep track of any changes, the analyzed frames are coherent with the ones showed in the previous section.

3.2.1 Decolorization

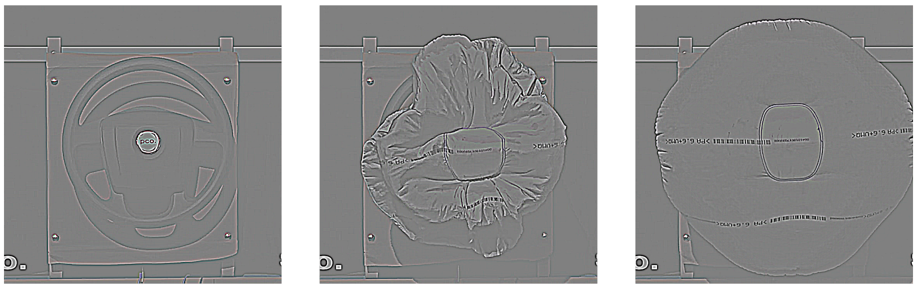
The first experiment involves a per-frame pre-processing, meaning that each frame can be transformed without using any other frame from the same video as base or reference. The idea of the *Decolorization* process comes from the transformation introduced by Benjamin Graham in [10]. Graham's aim was to *normalize* lighting and color aberration variations within the employed dataset by removing the local average color in the neighbourhood for each pixel. Despite the need for this kind of normalization, the dataset involved[3] is quite coherent in the depicted objects, since it contains a set of retina images with less varying characteristics with respect to the ones that can be detected in the custom dataset employed in this thesis work, according to an empirical analysis. For this reason, the aim of the operation is not shared with its original application, although it is possible that, in training conditions, the process could benefit also from the point of view of a further dataset normalization. As reported in [10], the operation involves the convolution of the original image with a Gaussian filter (in order to obtain a blurred version of each frame) and the subtraction of the result from the original image. The Gaussian Blur step enables the calculation of the actual local average color for each pixel, where the *locality* is given by the kernel size and the standard deviation used in the convolution with the Gaussian function. The subtraction of the local average color from each pixel is based on a weighted sum, plus a value that increases by 128 the average pixel values.

In the following, two attempts are described, involving different kernel sizes and standard deviation values.

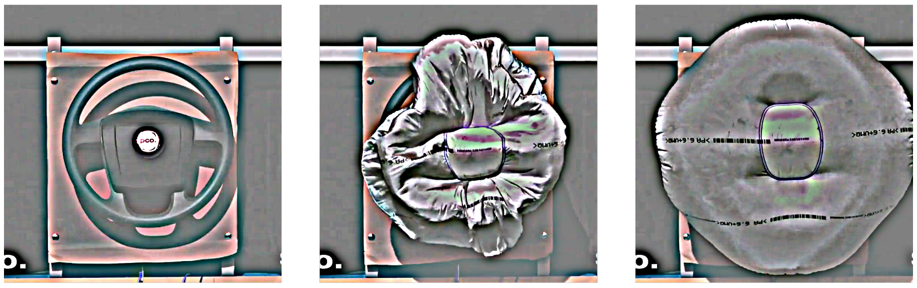
The first case, showed in Figure 3.3a, uses a low standard deviation value, meaning that the local average for each pixel is very similar to the value of the pixel itself and the output depicts a version of the original frame that is deprived of most of its original contrasts and color combinations.

The output predictions in Figure 3.4a result in a definitely worse quality with respect to the ones obtained by means of the expected type of input. In the first prediction it is

[3]https://www.kaggle.com/c/diabetic-retinopathy-detection/data



(a) Heavy decolorization.



(b) Light decolorization.

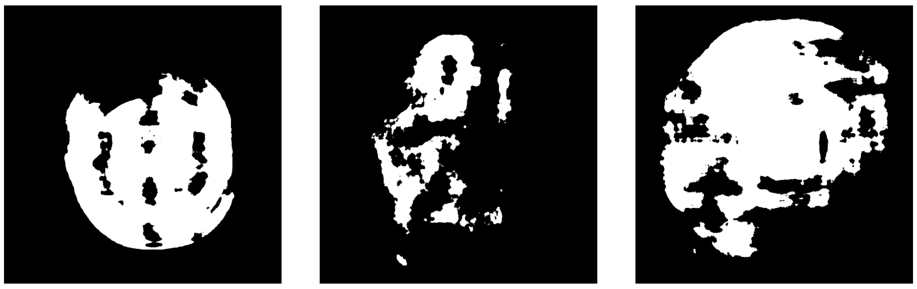
Figure 3.3: Collection of three sample frames after a heavy decolorization process (a) and a lighter one (b).

evident the model completely misunderstands the content in the image, wrongly returning positively labeled pixels where, in ideal conditions, there would be a completely expanded air chamber. In the second prediction the number of false positive labeled pixels decreases but the quality of the prediction itself does not seem to get acceptable. In the third image the positively labeled area increases again but, this time, as a reaction to the actual airbag inflation.

Figure 3.3b shows a lighter decolorization, meaning that the blurred version of the image to be subtracted from the real one is obtained through a higher standard deviation with respect to the first decolorization attempt. The effect of using an higher standard deviation, and therefore also a larger kernel size, is that, for each pixel in the original image, the subtracted value derives from an average on a wider area compared to the previous case. This means that the resulting image presents less accentuated colors on some surfaces but still visible contrasts in correspondence with the contours of the objects.

Regarding the results based on the input just described, showed in Figure 3.4b, a slightly improved first prediction can be noticed. Specifically, being the colours closer to the ones in the corresponding original image, the segmented version presents a decreased false positive area with respect to the previous case. In the second and third prediction, however, there is no visible improvement if compared to the havier decolorization case.

The analysis that is described in this section seems to reinforce what has been highlighted in Section 3.1, regarding the poor generalization performances of the employed model in relation to contrasts and color combinations.


(a) Segmentation based on heavy decolorization.

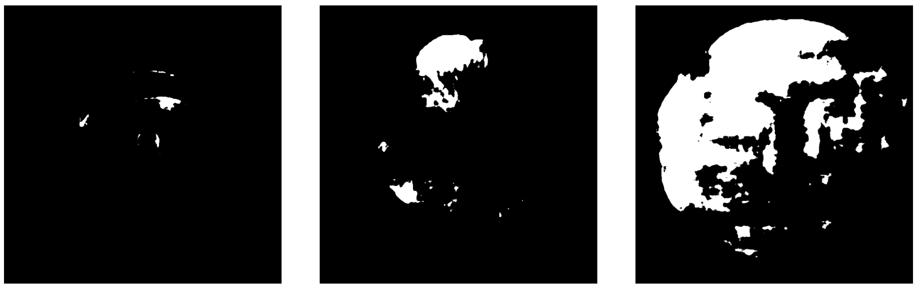

(b) Segmentation based on light decolorization.

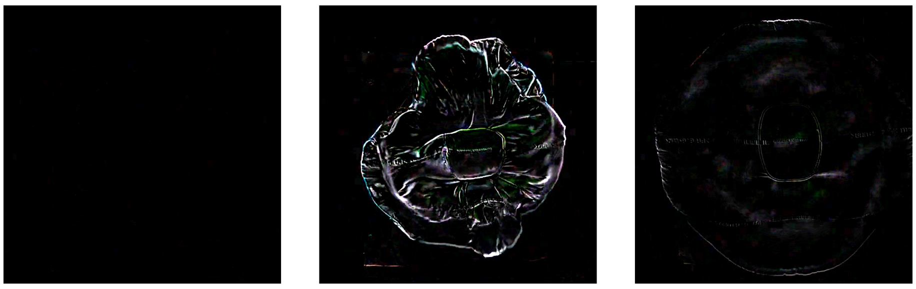
Figure 3.4: Segmented predictions corresponding to a set of decolorized frames by means of both the heavy approach (a) and the light one (b).

3.2.2 Motion detection

The experiment described in this section is based on a different pre-processing methodology for the input images. Although the resulting input images can be similar to the previous attempt, the main aim, in this case, is to isolate the actual moving pixels in each frame of the video, with respect to the previous one. The pre-processing step is quite simple: for each frame, the previous one is subtracted through the same type of weighted sum as the previous experiment. The only difference with respect to the weighted sum employed in Subsection 3.2.1 is that the the result of the sum itself is not increased by 128, since, in the images that contain many moving pixels, the shapes are more visible than in the previous test. In addition to that, the aim of the current test also lays in understanding how the model behaves with completely black images, as in the case of the first frame in Figure 3.5a.

The result of the pre-processing step actually removes the most of the elements that do not coincide with the airbag itself, highlighting the edges and the folds on the air chamber surface. A drawback of the described methodology is that, in frames that do not contain many moving pixels with respect to the previous one, the result is an image with hardly recognizable and low-contrast content, like in the last frame of Figure 3.5a. Moreover, since the support frame is not completely rigid and immovable, some modified images present noise in terms of pixels detected as having different value compared to the previous sample.

Figure 3.5b shows the effects that the presented pre-processing methodology has on the model output and, in these regards, the result seems worse than the previous case, with


(a) Frames with motion detection.

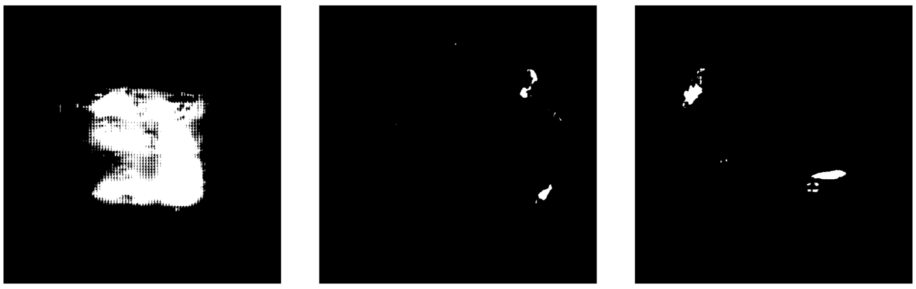

(b) Segmentation based on frames generated with motion detection.

Figure 3.5: Sample frames generated by means of motion detection pre-processing, together with the resulting segmented images from the default model.

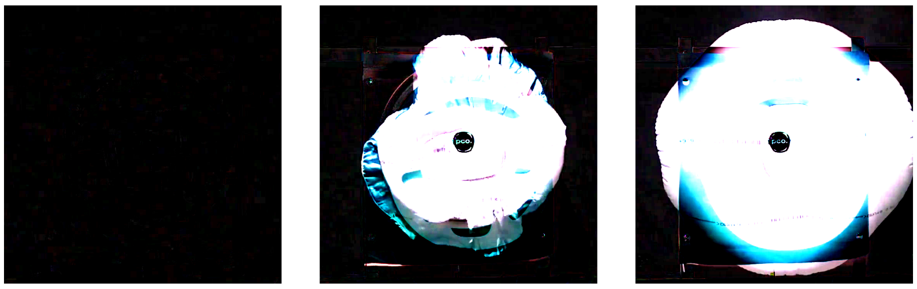
whom it shares some elements. Specifically, the first output is completely misunderstood, since the false positive pixels are spread as if in the original image there was an airbag during the inflation phase, but this is not the case. In the rest of the output predictions, the positively labeled pixels are not compactly spread in the airbag area, neither during the airbag deployment or in the condition of complete extension. In general, it seems that the completely black (or empty) image case is not covered by the model capabilities, while images that contain sets of moving pixels with the appearance of an airbag are not enough for a good prediction, if they are not supported by the inner color of the generated shape.

Also in the case of the experiment described in this section, the model seems not to generalize enough for a correct detection of the airbag bounds, even if they are isolated from the most of the background elements.

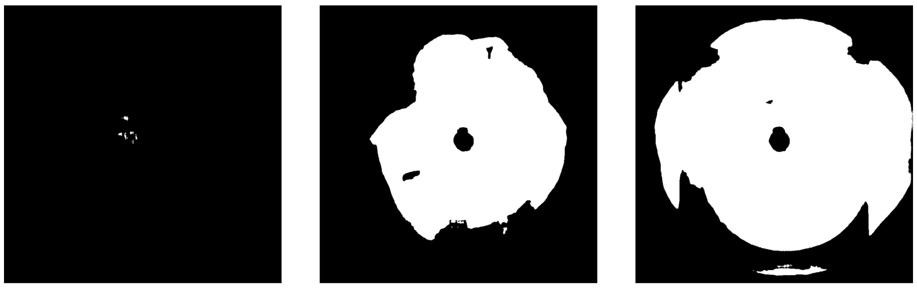
3.2.3 Background removal

The analysis carried out in this section merges the attempt of isolating the airbag from the background with the need for an easier representation of the airbag itself in terms of colors and contrasts. The latter goal derives from the inability of the model to recognize the air chamber if deprived of its characteristic colors and contrasts. Hence, the overall objective of this experiment is to test the behaviour of the employed network with input images that are deprived of the sections depicting the background.

The background removal algorithm is based on the weighted sum employed in Subsection 3.2.2, meaning that black pixels are not mapped to grey; specifically, in this case the subtracted frame coincide with the first one in the sequence that is extracted from the video. By doing so, the frame that is subtracted contains every fixed element in the image and, after the operation, deletes the constant fixed elements from each subsequent frame. However, the first frame includes also the container of the air chamber (that could be either a steering wheel or another bag) when it is closed, before the deployment starts. This implies that, when the initial frame is removed from a subsequent one containing the deployed (completely or not) air chamber, the white surface of the airbag presents shadows which can be traced back to the negative version of the removed frame.



(a) Frames with removed background.



(b) Segmentation based on frames with removed background.

Figure 3.6: Sample frames generated by means of background removal pre-processing (a), together with the resulting segmented images from the default model (b).

As Figure 3.6a shows, the first frame is completely black, since the removed frame is almost identical to the considered one. In the center of the second frame, however, the negative of the steering wheel logo can be recognized; in the same way, some other visual artifacts in the place of the steering wheel support are due to its movement after the explosion. The third frame in Figure 3.6a presents shadows corresponding to the surface of the support in the peripheral areas of the airbag surface, in addition to the logo.

Given the described images, it is interesting to understand how the model performs having the important object as isolated and similar to the original as possible by means of a fast pre-processing. In Figure 3.6b, the first prediction (deriving from a black image) is quite perfect, showing only a minimal amount of noise in the form of some false positive pixels. This case seems to be in contrast with the corresponding prediction in Subsection 3.2.2, since both predictions have a black image as input. However, the difference lays in the fact that the current black input image is the result of the subtraction of the first frame of the sequence, while the previous case involved the removal of the preceding frame, resulting in a lower number of "moving" pixels that generate an "unreal" condition. The second segmented image is still quite acceptable, but the prediction suffers from the influence of some negative elements of the original frame, such as the logo in the center and some shadows at the ends of the airbag surface. The third prediction confirms, in a more explicit way, the observation made for the previous sample: the output presents some negatively labeled areas that correspond to the wider shadows characterizing the third input frame.

The described experiment shows how well the original model performs if provided with known colors and contrasts, with respect to conditions which only provide shapes and highlighted edges. Despite some predictions are heavily affected by shadows and visual artifacts deriving from the provided input, the results are still interesting, since the model performances improve with the removal of disturbing elements belonging to the background.

Chapter 4

First approach: data pre-processing and decolorization

Chapter 3 highlights how dependent the default model is on color combinations and recurring contrasts in the recognition of the segmentation subject. Similar patterns and combinations to the ones that characterize the presence of an airbag in a specific area can mislead the model into recognizing certain areas as falsely positive or falsely negative, in terms of whether the pixels belong to the actual air chamber or not. Moreover the default training process is not binary, in the sense that the model is also set to learn a background class in addition to the airbag one. By means of the empirical tests carried out in the previous chapter, it is clear that learning a background class could lead to a wrong prediction in case the actual submitted background is not known from the training process.

The aim of the approach presented in this chapter is to improve the current model performances by keeping its overall architecture and concept. In particular, the changes are made in order to introduce a better generalization and normalization in terms of color combinations and contrasts expressed in the input images. This being said, the only modification on the actual model architecture is the reduction of the number of output channels in the last convolutional layer: indeed, the default model presents 2 output channels, meaning 2 classes respectively used for the background and the airbag, while the approach described in this section learns how to distinguish only the class corresponding to the airbag with respect to anything else.

Table 4.1a shows the integration of the pre-existing ERFNet architecture with the modification previously described. As the table reports, the resolution corresponds to a squared image and this characteristic has not been modified with respect to the default solution. Moreover, Table 4.1b describes the blocks introduced by Romera et al. in [17], both for the upsampling operation and for the substitution of the usual non-bottleneck and bottleneck residual blocks.

Modified ERFNet		
Arm	**Output size**	**Architecture**
Encoder	384×384	Downsampler block, 16
	192×192	Downsampler block, 64
		5× Non-BT-1D, 64
	96×96	Downsampler block, 128
		8× Non-BT-1D, 128
Decoder	192×192	Upsampler block, 64
		2× Non-BT-1D, 64
	384×384	Upsampler block, 16
		2× Non-BT-1D, 16
	768×768	Upsampler block, 1

(a) Representation of the Encoder-Decoder ERFNet architecture, adapted for the prediction of a single class and for an input resolution of 768×768 pixels.

Blocks explained		
Block name	**Type**	**Architecture**
Downsampler Block, x	Convolution	$3\times3, x$
Non-BT-1D, x	Convolution	$\begin{bmatrix} 3\times1, x \\ 1\times3, x \\ 3\times1, x \\ 1\times3, x \end{bmatrix}$
Upsampler Block, x	Deconvolution	$3\times3, x$

(b) Explanation of the notation employed in the ERFNet architecture for what concerns the Downsampler block, the Non-Bottleneck-1D block and the Upsampler block.

Table 4.1: Representation of the modified ERFNet architecture (a), with the addition of an exploded view of the blocks used (b).

4.1 Training algorithm

The training process was performed on hardware belonging to an AWS EC2 (Elastic Compute Cloud) instance[1]. The CPU model was a *Intel Xeon E5-2686 v4*, supported by a 32GB RAM. The GPU was a *NVIDIA Quadro M4000* instance, with a 8GB dedicated RAM. Both the underlying system and the container one were running Ubuntu 18.04 LTS as a Linux distribution.

[1]https://aws.amazon.com/ec2/features/

4.1.1 Pre-processing technique

The main change introduced in the proposed approach lays in how the data are pre-processed before entering the actual training algorithm. Indeed, as exposed in Section 3.2, it seems that the current version of the segmentation model suffers from a category of biases that is linked to the lack of variety of color combinations and light conditions in the training dataset. For this reason, the current proposal leverages on the same principle as the base of the experiment in Subsection 3.2.1, which involves a per-image color processing aimed at the normalization of the aforementioned light conditions and contrasts. However, while the Decolorization experiment in Subsection 3.2.1 comprehends both a heavy version and a lighter one, the current pre-processing operation only consists in the "lighter" approach. The reason is mainly empirical: approaches with low standard deviation values for the removed blurred versions of the frames return output images that present a heavy lack of color informations, meaning that the training of the deep model should force the weights to learn features which derive only from shapes and raw patterns, without any explicit color and contrast information. For this reason, even if the proposed heavy version could potentially bring to a very robust model, the training convergence, by keeping the model architecture constant, is too slow. A trade-off is represented by the lighter decolorization version, which, by the way, is closer to the goal for which the methodology was introduced in [10] since it works like a per-image color and light normalization.

The actual decolorization process is actuated for each loaded image in the dataset class, by means of the transformation function, provided by a dedicated class. The decolorization is performed at the end of the stack of augmentations included in the aforementioned function, introduced in Subsection 2.4.3.

4.1.2 Training procedure

The actual training algorithm is similar to the default one, described in Subsection 2.4.3 in terms of how the model is fed. There are some differences deriving from the fact that the modified proposal outputs a single channel image, since the only learned class is the airbag one.

The first different element lays in the employed Loss function. In the default approach, the unnormalized predictions (or *logits*) are given as input to a LogSoftmax activation function and the obtained results are used for the actual calculation of the loss value, by means of a weighted NLLLoss. In this case, however, the output of the model corresponds to a single 2-dimensional vector containing the resulting logits, representing the raw probability prediction that each mapped pixel belongs to an airbag or not. The described output is then directly given to the Pytorch implementation of a Binary CrossEntropy Loss, called *BCEWithLogitsLoss*[2], which introduces some peculiarities. First of all, there is no need for a further manual step through an activation function, since the introduced Loss implementation provides an integrated and optimized way for applying a Sigmoid activation function to the generated logits. The described approach is usually employed in the

[2]https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

case of binary classification, since it corresponds to the application of a Softmax activation in a single label situation [2]. Secondly, the provided interface accepts a rescaling parameter, meaning that, if the positive and negative occurrences of the analysed class are unbalanced, it is possible to obtain a weighted version of the loss by simply computing the ratio between the background pixels and the ones belonging to the airbag. In this case, the reweighting strategy is as simple as counting the positive and negative pixels and then computing their ratio, which is in contrast with the more complex algorithm employed in the default approach, due to its multi-class nature.

As in Subsection 2.4.3, in order to evaluate the best performing update during the training phase, the Validation epochs are based on a single pass over the Validation set and the computed loss and IoU values are collected in order to determine the goodness of the updates. For what concerns the IoU metric, it is still computed by considering both the background and the airbag class in order to be comparable with the values obtained for the default approach.

4.2 Comparison

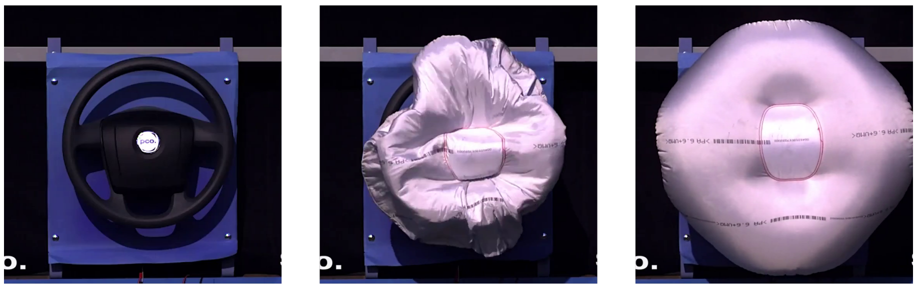
In order to compare the performances of the default segmentation model and the one whose training phase and terminal layer are modified, two proofs are reported. Firstly, a merely empirical experiment shows a comparison between the results obtained by the two versions of the model through the segmentation process applied on expected input. On the other hand, a script performs multiple validation epochs, while calculating the IoU metric in the same way as it is done during the validation phase within the training process. In particular, the data analysed in the second experiment is more aggressively augmented with respect to the validation phases in the training processes of both the models. A specific focus is applied on the color augmentation section: in fact, colors are not only saturated or desaturated as in the normal validation phase, but some hues are also inverted. This is done with the aim of comparing the generalization capabilities of the models.

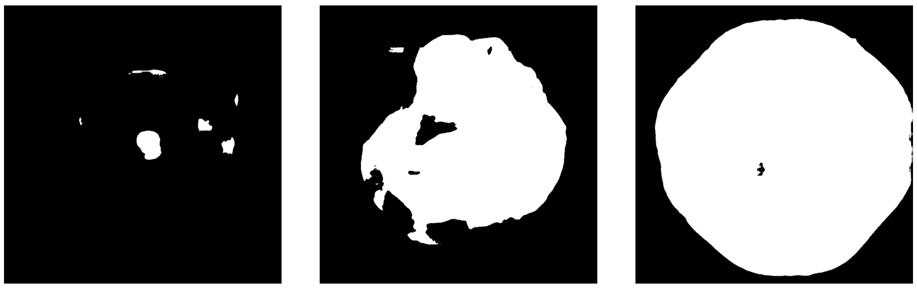
4.2.1 Visual comparison

The first test regards a visual comparison between three segmented images, respectively returned by the default model and the updated one. Three frames, belonging to the same video as the one used in Chapter 3, are used, in order to represent all the main states of the airbag deployment.

Figure 4.1a represents the actual input frames to the compared models and, as stated before, the first frame do not contain any airbag trace, the second image contains the air chamber during the deployment phase and the last image shows a fully deployed airbag.

Figure 4.1b represents the result of the segmentation process performed by the default model. As highlighted in Section 3.1, the model is quite sensitive to the light contrasts that are generated by the reflections on the logo of the steering wheel in the first frame. In the subsequent predictions performed by the default model, the main problems regard the airbag during the deployment phase, in fact the second segmentation is jagged and lacks of true positive predicted pixels in some areas that correspond to shaded surfaces.


(a) Expected input.


(b) Segmentation based on expected input with default model.

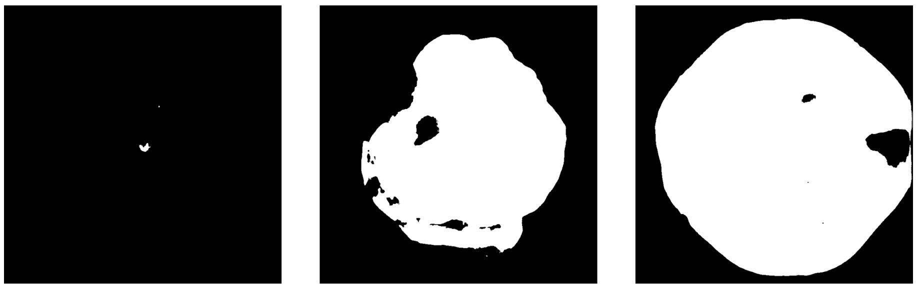

(c) Segmentation based on expected input with updated model.

Figure 4.1: Comparison between the predicted segmented images obtained by means of the default model (b) and the ones obtained by means of the updated one (c).

The third segmentation generated by the first version is almost perfect, since it covers the majority of the airbag area.

Figure 4.1c shows the corresponding predictions resulted from the model with the introduced modifications. A first improvement can be recognized in the initial prediction: in fact, it is evident a drastic reduction of the number of false positive pixels in the neighbourhood of the steering wheel logo. In the second prediction the number of true positive pixels seems higher with respect to the previous counterpart, while the third prediction is not as accurate as the corresponding one in Figure 4.1b, since it contains a hole in the rightmost end of the prediction surface.

In general, except for the third prediction, the updated model seems improved with respect to the default one, reinforcing the thesis that, by normalizing the dataset in terms of colors and contrasts, the final model can benefit from it in accuracy.

4.2.2 IoU comparison

The second test is based on the evaluation of the same metric that is employed in the validation phase of each training epoch, which is the IoU value. The computation of the metric is carried out in the same way as during the training, by means of the same class and related functions. Specifically, for each batch, the total number of true positive, false positive and false negative pixels is computed and summed with their corresponding class counters. At the end of the epochs, for each model, the aforementioned counters are used as in Equation 2.2.

IoU comparison	
Model	**IoU (%)**
Default model	95.37
Modified model	98.64

Table 4.2: Comparison between the values of the Intersection over Union values computed by means of multiple iterations of a segmentation process over an aggressively augmented validation set.

Table 4.2 highlights the performance improvement obtained in terms of IoU calculated on both the airbag class and the background class, even if the updated model is trained to recognize only the airbag class.

The obtained results confirm the overall improvement in the segmentation performances, meaning that the updated model can be integrated in the Detection pipeline of the analyzed tool, aiming at translating these advances into a better final detection quality.

Chapter 5

Second approach: classifier with forced attention

As described in Section 2.3, for each frame, the current detection pipeline is based on different steps: a pre-processing based on a Machine-Learning algorithm, the application of each defined ROI on the segmented image generated by the Machine-Learning model, and the actual detection, performed by means of some thresholds measuring positively labeled pixels with respect to the entire ROI area. Chapter 4, is focused on evaluating a method for improving the generalization of the Machine-Learning step whose main task is to semantically segment each given input image. The success of the detection operation, however, is also strongly dependent on the steps following the one that has been analyzed: for this reason, improving the ML model does not necessarily mean improving the overall detection performance as well.

In this section, another pipeline approach is presented. The conception of the following sequence of operations is directly driven by the main aim of the pre-existing pipeline, which is the actual detection of a condition between the airbag and a pre-defined area. In this context, by changing the view of the problem statement, *detecting* the time in which the airbag enters the ROI or completely covers it is the same as classifying each ROI (by tracking the time) as containing a piece of air chamber or being completely covered by it. This being said, the ML approach described in this section completely changes and covers most of the sequence of operations previously performed in different steps. Indeed, in terms of pipeline phases, the modification regards the *airbag segmentation* and *detection*; according to a deeper view, however, the *detection* phase includes the per-frame isolation of each ROI and its evaluation through the predefined thresholds (as described in Section 2.5), which can be considered as two well defined sub-phases due to their belonging to two different inner loops within the outer loop.

In order to make a comparison with the three highlighted stages, the proposed approach incorporates them all in the form of a unique ML-based step. The employed model takes two parallel elements as input: the usual extracted frame and a binary mask which represents the coordinates of a ROI. The output is a class label that represents either the absence, the presence or the invasion of a ROI by an airbag section. Depending on the type of ROI that the user defined, either the second or the third label fires the actual detection of the time when the condition has been met for that specific ROI.

In the following, different aspects regarding the proposed approach are described:

- the first analysis regards the choice of the architecture and the reason that has driven to its proposition;
- a particular focus is necessary on describing how the data are employed in inference condition and how the original dataset is transformed to be adapted to the new model architecture and task;
- as for the previous models, the training algorithm is a fundamental point for linking the performances obtained with the way the model itself works.

5.1 Architecture choice

Instead of just presenting an improvement attempt for the existing model, this thesis work also focuses on the realization of an alternative algorithm for satisfying the problem statement expressed in Section 1.3. Mainly driven by the need for a better time performance, the first filter between the various approaches gives as result a *classifier* architecture. Indeed, keeping as goal the aforementioned modified view for the problem, a simple classifier model can help to improve at least the performances in terms of inference time. This is due to the lack of a second arm used for upsampling the result of the encoder. Using a classifier, in fact, can be viewed like using only the encoder section of the architecture employed for the segmentation task and, therefore, this reduces the inference time.

The main problem regarding the classifier architecture is how to use both the information deriving from the input image and the one linked to the user-defined ROI while the usual classification task involves the analysis of a single input image.

The *usual* objective for a classification-oriented model is to determine the type (or class) of an object contained in the input image. Starting from the proposition of the first CNN approach for image classification [12], the logic and precision applied to this task evolved with the introduction of the *region proposal algorithms*. These algorithms were usually built on shallower concatenated models or similarity based recursive process such as Selective Search. *RCNN* [9] is the turning point for the aforementioned models based on region proposal algorithms: in fact, many improvements arose from it, such as *Faster R-CNN* [16] which dramatically boosts inference and training performances by employing a single Fully Convolutional backbone aiming both at extracting region proposals and at evaluating them by means of a further classification section.

The common point in the referenced models is the fact that the *region proposal process* itself is learned too, since, most of the times, the objective is to classify the main object in the input image. The context of this thesis work requires the classification to be forcibly bounded with a set of user defined ROIs and this point makes the previously mentioned models incompatible with the goal itself. The aforementioned bound is obtained by using both the informations as input for the model: indeed, the next reasoning regards how to get the optimal combination of these input elements.

A practical survey over the different methodologies for manually forcing the attention in models aimed at classification has been carried on by Sagi Eppel in [7]. Eppel evaluates two main categories of attention induction methods, namely *hard attention* and *soft attention*. The *hard attention* approach involves merging the input image and the ROI before entering any convolutional model, by zeroing all the pixels in the image that are not included in the ROI: this means multiplying the input image by a binary mask containing ones within the ROI and zeros outside of it. The *soft attention* method involves merging the two input elements along with the feature extraction process within the employed deep convolutional network. [7] highlights how inducing attention within the convolutional backbone outperforms any *hard attention* method in classification tasks.

In a previous work, Eppel introduces the so-called *Valve filters* [6]: this approach involves the deep convolutional backbone to be prepared to accept both the image and the ROI based binary mask in parallel. The image is given as input to the first convolutional block of the backbone, while the binary mask is resized to be compatible with the output of the aforementioned convolutional block and then passed through another convolutional layer whose goal is to extract the same number of feature maps as the ones in the main branch. The reason why these feature maps are called *Valve filters* is that their aim is to regulate the expression of the different features extracted by the first block in the main branch, focusing the attention of the subsequent feature extraction operations to the area belonging to the ROI. The strength of this approach lays in the fact that *regulating* the expression of some extracted features is far from the logic of zeroing every background glimpse of the input image. This means that keeping elements from the background of the main input helps the rest of the convolutional architecture to exploit information coming from the context and the environment around the ROI, dramatically improving the performances of the model with respect to *hard attention* methods which completely delete any background element.

Based on the presented analysis, the classifier is equipped with a deep convolutional backbone, whose input is given by the merged outputs of two input convolutional layers, extracting the same number of feature maps from the actual input frame and the ROI. The merging operation involves the addition of the *Valve filters* to the first feature maps extracted from the image. Multiplication was also tried but, confirming what was asserted in [6], the the training convergence is a bit slower. The first convolutional layer encountered by the input image and the one that is added for the parallel ROI input are part of the first block of a modified *Resnet50*, which works as backbone of the model. Table 5.1 summarizes the overall classifier architecture in inference mode.

5.2 Input and training data

The default approach is based on a single path, which is characterized by a single input source and a single output end. The input is represented either by an image in a simple inference condition or by a batch of images during the training phase. The approach presented in this chapter introduces a second input branch, which is parallel to the main one until the conjunction point. The main branch still takes as input the same images as the default approach, meaning that each extracted frame is either used as input for inferences or collected for the training process. The second branch is fed with a binary mask that, in a inference condition, is derived from a user input; in this context, the way the user defined coordinates are translated into a binary mask is similar to the sequence of operations presented in the Algorithm 1.

The main changes are applied on how the data is used for training purpose. Indeed, the original dataset is not compatible with how the current model should be fed during the training phase, since it does not include neither a collection of binary maps or class indexes as labels. What is needed by the current approach is more complicated with

Classifier with forced attention			
Layer name	**Output size**	**Architecture**	
block 1	192×192	7×7, 64, stride 2	3×3, 64, stride 1
block 2	96×96	3×3 max pool, stride 2	
		$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	
block 3	48×48	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	
block 4	24×24	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	
block 5	12×12	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	
	1×2	Average pool, 2-d fc	

Table 5.1: Representation of the architecture of the introduced Classifier with forced attention, based on a Resnet50 with the addition of a second input convolutional layer for the ROI mask and the modification of the last linear layer which generates an array with length 2, instead of the previous version having length 1000, used for ImageNet classification.

respect to the standard approach: the input samples should be made by tuples of frames and binary masks, while the needed labels are actually class indexes, instead of reference segmentations.

In the following, the description of how the original dataset has been adapted is reported: specifically, the focus of the analysis is on the fact that the entire adaptation process is carried on at runtime, during the training process and for each single batch of images coming from the original dataset. This means that there is not a fixed transformed version of the dataset and the training samples change for every epoch, since the ROI sampling is randomized in every loop.

The reported *translation module* introduces some peculiarities and improvements to the training process.

- The lack of a fixed training dataset induces a considerable gain in the data augmentation process. Indeed, the original dataset is quite limited in terms of number of samples and the standard augmentation is bounded with the frequent patterns and backgrounds which are distributed over the entire collection. In the proposed approach, the introduction of the ROI input forced the development of the *Translation module* to include a ROI proposition method and that is a powerful source for a better generalization.

- The standard dataloader, employed in the default version of the detection backend, is not aware of the class imbalance that is obtained by simply loading every image in the dataset: for this reason the default loss function is provided with class weights, which are calculated as described in Section 2.4.3. Being the *Translation module* implemented as a dataloader, it is made to provide balanced samples in terms of reference class labels. Specifically, the presented module loops over the loaded batch until a balanced combination of samples is collected, then returning the results as a usual dataloader does.

- The random ROI proposal actuated for the training samples depends on the provided batch, which, in turn, depends on the data shuffling operation: for this reason it is possible for some batches not to include enough positively labeled regions for the class balancing operation. In this regards, the *Translation module* saves a backup batch that is reloaded in case the current set of images does not provide useful labeled regions.

A detailed description of the translation module is reported in the following. The algorithm also shows some sections referring at the *multi-task* mode, which is described in Section 5.3.

1. As initial conditions, it is necessary that some elements are defined: in particular, the given objects are the number of ROIs (called *patches*) per batch, the number of class labels (3 is the default value), a boolean variable indicating whether the loaded data are validation data or training ones, another boolean variable pointing at the multi-task mode, the batch size, the batch itself and the resolution to be used for the returned elements. The first step of the actual algorithm regards the initialization of the backup batch, that is subsequently used when a future batch is not balanced for what concerns class labels.

2. When the translation module is used in the context of a validation phase, the corresponding function is employed: the only difference with respect to the reported algorithm lays in the fact that, after the first validation epoch, the sequence of patches is fixed for the subsequent iterations in order to have a coherent comparison within the different epochs results.

3. If the Translation module regards a training batch, the first sequence of operations involves the initialization of the data structures that are returned at the end of the algorithm. In particular, the index that is used to point to the free cells in the structures is set to zero and the maximum possible number of patches per class that can be obtained by using the provided total upper limit number of patches is calculated. Subsequently, the array that contains the per-class counters is initialized and then filled with values corresponding to the previously computed number of patches per-class. Afterwards, the frames and the masks arrays are initialized, respectively with 3 channels for the RGB frames and 1 channel for the binarized masks. The last initializations correspond to the labels array and, if the multi-task mode is active, to the coverages array. The latter element contains the calculated airbag coverage levels in each ROI.

4. Before starting the actual patches search process, the initial timestamp is saved in order to be used for checking whether the elapsed time justifies the use of the backup

batch. At this point, the actual patches search starts and the stop conditions correspond to the reset of the class labels counter. The first operation in the search loop corresponds to the timestamp check: if the elapsed time is greater than 60 seconds, then the backup batch is employed, otherwise, if there is no backup batch, the entire training process is stopped. This case can happen only for the first batch of the first epoch, since, if it completes without any problem, there will always be a backup batch along the entire training process.

5. The main operation in the patch search process involves looping on the entire batch, which is very likely to occur more than once due to the randomness of the coordinates sampling. For each round over the batch, the frame and the corresponding segmentation mask are extracted; then there is the actual random sampling, which is based on the bounding box of the provided frame, meaning that there is no possibility for the sampled coordinates not to be within the image limits. The subsequent operation is the ROI isolation from the provided binarized mask (by means of the same function used in the Algorithm 1), aiming at getting the base for the label computation. The latter operation regards the extraction of the class index that is compared with the predicted one in the training phase and it is carried out by measuring the ratio between the active pixels and the total number of pixels in the ROI that is applied on the segmentation mask. Together with the extracted label, also the aforementioned ratio is returned, corresponding to the coverage that is used for the multi-task mode.

6. If the counter that tracks the extracted label did not reach zero during the previous loops, the selected frame is used to extract the original size for the ROI coordinates projection, then it is resized by using the provided resolution and it is added to the corresponding data structure. The ROI mask is created by using the sampled coordinates and the original size, then it is resized and added to its collection too. In the same way, also the labels and the coverage value (if needed) are added to the respective arrays. After the collection of each element, the counter corresponding to the found label is decreased, while the index of the empty cell in the data structures is increased. If all the label counters have reached the zero value, then the entire translation step is arrested and the collected elements are returned to the training process.

7. If the employed batch provided a set of balanced labels and the backup batch has not yet been saved, then the current batch is used for this aim. When the patches search is over, the data structures are returned to the main process, both if the multi-task mode is active and if it is not.

5.3 Training algorithm

Regardless of the chosen deep convolutional backbone, the training process can be set with two modes. The traditional training only regards the predicted class labels and the reference class labels; on the other hand, the multi-task mode makes use of a double-headed convolutional backbone, aiming at predicting both a class label and the airbag coverage percentage for each ROI in a frame.

Algorithm 2 Translation step

```
Require: $max\_patches$, $num\_classes$, $is\_validation$, $is\_multi\_task$, $batch\_size$, $batch$,
  $height$                                                                                    ▷ (1.)
  $backup\_batch \leftarrow NULL$
  if $is\_validation$ then                                                                    ▷ (2.)
      $validation\_call(batch)$
  else
      $k \leftarrow 0$                                                                        ▷ (3.)
      $n\_patches \leftarrow adjust\_n\_patch(num\_classes, max\_patches)$
      $label\_counter \leftarrow$ zeros$(num\_classes)$
      for each $i \in$ range(num_classes) do
          $label\_counter[i] \leftarrow n\_patches$
      $frames \leftarrow$ zeros$(n\_patches, 3, height, height)$
      $masks \leftarrow$ zeros$(n\_patches, 1, height, height)$
      $labels \leftarrow$ zeros$(n\_patches)$
      if $is\_multi\_task$ then
          $coverage \leftarrow$ zeros$(n\_patches)$
      $time\_0 \leftarrow$ time()                                                             ▷ (4.)
      while sum$(label\_counter) \neq 0$ do
          if time() $- time\_0 > 60$ and $backup\_batch$ is not $NULL$ then
              $batch \leftarrow backup\_batch$
          else if time() $- time\_0 > 60$ and $backup\_batch$ is $NULL$ then Error
          for each $(frame, segmentation) \in batch$ do                                       ▷ (5.)
              $box \leftarrow$ sample_box$(frame)$
              $seg\_crop \leftarrow$ extract_box$(segmentation, box)$
              $lab, cov \leftarrow$ compute_label$(seg\_crop)$
              if $label\_counter[lab] \neq 0$ then                                            ▷ (6.)
                  $original\_size \leftarrow frame.$shape$(1)$
                  $frame \leftarrow$ resize$(frame, height)$
                  $mask \leftarrow$ draw_rois$(bbox, original\_size)$
                  $mask \leftarrow$ resize$(mask, height)$
                  $frames[k] \leftarrow frame$
                  $masks[k] \leftarrow mask$
                  $labels[k] \leftarrow lab$
                  if $is\_multi\_task$ then
                      $coverage[k] \leftarrow cov$
                  $label\_counter[lab] \leftarrow label\_counter[lab] - 1$
                  $k \leftarrow k + 1$
                  if sum$(label\_counter) = 0$ then Break
      if $backup\_batch$ is $NULL$ then                                                       ▷ (7.)
          $backup\_batch \leftarrow batch$
      if $is\_multi\_task$ then
          return$[frames, masks, labels, coverage]$
      return$[frames, masks, labels]$
```

The training process was performed on the same hardware as in Section 4.1.

Standard training

Specifically, the traditional approach only trains the model for classification aims, with reference to the number of classes defined before starting the process. The chosen number of classes is 3, since the main objective is to recognize whether the ROI simply contains a flap of an airbag that does not completely cover the region itself or whether the entire surface of the ROI is covered by the fabric of the airbag. The third class is related to the case the airbag is not detected at all. The specific class order is: 0 if no airbag is detected, 1 if only a section of the airbag is detected within the ROI, 2 if the ROI is completely covered by the airbag surface. Said labels do not correspond to the ones directly predicted in the training phase, since, as explained in the following, the learned classes correspond to the *"partial coverage"* and the *"complete coverage"* ones in order not to bias the model towards the knowledge of a subsection of the possible range of backgrounds.

In the context of the standard training process, the loss is a Multi Label Binary Cross Entropy Loss, implemented by Pytorch[1]; the employed function is the same as the one used in the case of the first approach in Section 4.1 with a different number of classes. A Binary CrossEntropy loss is computed considering each detected class, by using as input both the logits (which are passed through a Sigmoid activation function) and the one-hot-encoded labels, meaning that the original 1-dimensional label array is transformed into a 2-dimensional array. The original label array is the one that derives from the Translation module in Section 5.2. In this regards, there is no need for a class weighting strategy, since the aforementioned module provides a balanced set of training data. The choice of the Binary CrossEntropy loss with Sigmoid activation function instead of a more standard approach like the combination of the usual CrossEntropy loss with Softmax activation function, was driven by the same principle as the one introduced for the first approach: the possibility to have an independent evaluation of the error for each class enables to focus only on learning the *"partial coverage"* class and the *"complete coverage"* one, linking the *"absence"* class to a low probability value corresponding to the highest one in the predicted tuple. Said methodology involves a further fine-tuning process related to the selection of the best probability threshold below which the final label corresponds to the *"absence"* class; the aforementioned analysis is performed by means of a cross validation in a subsequent step with respect to the training one.

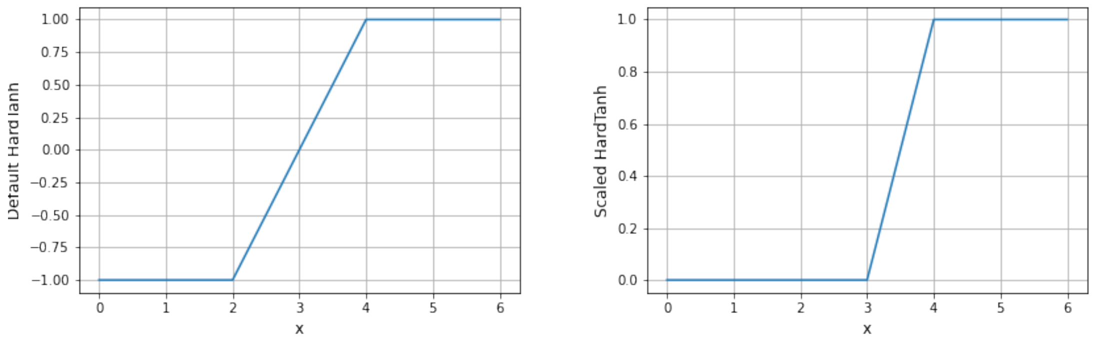
As for the training process related to the default approach, the optimizer is based on the Pytorch implementation of the Adam function, with a learning rate scheduler that is quite different with respect to the default one. Indeed, the trend is cyclic, with a fixed maximum and minimum values of the learning rate within which the actual value varies multiple times.

At the end of each epoch, in order to save the best performing update of the model, a control over the prediction accuracy and loss is performed: the best epoch is the one that shows the best combination in terms of minimum value of the loss and maximum value for the accuracy in the validation phase.

[1]https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

Multi-task training

The second training mode involves the addition of a second head to the deep convolutional backbone while the first output is still given by a linear layer with three generated features, corresponding to the previously mentioned class labels. The second output is the new one and the goal is to predict the coverage of the airbag surface in the ROI. The idea behind the introduced mode is to combine a sort of *linear regression* task with the main task, aiming at improving the awareness and the understanding of the model in terms of what should be evaluated in order to perform the right prediction by means of the classifier head. The actual prediction corresponds to a number that, potentially, can range from $-\infty$ to $+\infty$. For training purpose the raw output is used in the loss, together with the calculated coverage; in the validation phase, instead, the raw output is passed through a Hard Tanh activation function, which linearly isolates the values in $[0,1]$. The aforementioned activation function is implemented by means of the Pytorch framework[2], in a translated and scaled version, so that the output value is bounded in the previously reported interval, as showed in Figure 5.1b, with respect to the default version which is limited in $[-1,1]$.



(a) Default HardTanh

(b) Scaled HardTanh

Figure 5.1: Representation of the Hard Tanh activation function: the default version (a) on the left saturates in $[-1,1]$, the scaled version (b) on the right spans in the range $[0,1]$

Since the multi-task mode involves the model to be trained for both the tasks, the loss function is adapted to the described case by simply summing the previously mentioned Multi Label Binary Cross Entropy version (employed for the classification task) and the Mean Squared Error, which is used as baseline loss function for Linear Regression tasks. The value of the loss that is computed through the described sum is only used for the training phase, indeed, the choice for the best performing update of the model in terms of validation accuracy and loss is still based on the Cross Entropy loss taken alone. This means that the multi-task mode only works as support for the actual learning process, without influencing the final prediction.

[2]https://pytorch.org/docs/stable/generated/torch.nn.Hardtanh.html

5.4 Detection Pipeline integration

The approach presented in this chapter provides an end-to-end path from each frame-ROI tuple to the detection of the airbag in the specified region. For this reason, the integration of the described model in the *Airbag Detection backend* involves a substantial change in the Detection Pipeline. The new pipeline does not include the segmentation step, which was a time and resource consuming process to be carried out. Instead, after the frames extraction, the presented model is directly fed with the extracted frames and the user-defined ROIs.

The pipeline described in the following introduces a new detection methodology based on two control modes. The *instant control* mode bases the detection on the label predicted by the classifier by feeding it with the selected frame and ROI mask: if the predicted label corresponds to the one that should be met depending on the ROI type, then said frame is intended as firing the detection. However, this methodology is subjected to the possibility that the classifier wrongly predicts the first frame in the defined range for that ROI as the one that causes the detection, inhibiting the analysis of the following frames. For this reason the *memory control* mode can be activated in case the previous methodology is considered unreliable for the described reason: the *memory control* methodology compares each predicted probability corresponding to the selected class with the first one in the sequence generated by the frames range of the ROI. By doing so, the detection is not fired with a single "absolute" prediction which is only based on the higher probability value between the two predicted ones, instead, it takes into consideration the trend of the probabilities themselves, providing a slightly less accurate but more robust approach. This double control paradigm is based on the assumption that the user selects a range of frames of interest for a certain type of detection that is wide enough to also contain frames prior to the actual occurrence of the ideal detection, in order to support the idea that, if the *instant control* mode generates an unexpectedly early prediction, in can be considered unreliable. In the following, Algorithm 3 is described in steps.

1. As well as in the previous version of the detection phase, also the updated version needs some initial conditions. In addition to the list of user-defined ROIs and the indexes of the initial and final frames, there is one parameter more: the height variable is used as reference size for the resizing operation carried out in the subsequent steps. The first operation is the initialization of the *detection* dictionary, in the same way as the previous version in Algorithm 1. Then, the first frame of the selected sequence is loaded from memory, in order to extract the original resolution, before it is resized.

2. The first loop regards the list of ROIs and the detection dictionary: the latter is filled with *NULL* values in order to be checked when a detection occurs. For each ROI in the list, a mask is created and added to the corresponding dictionary, in order to avoid the repetition of this phase for each image when the algorithm loops over the frames sequence. Subsequently, the classifier model is instantiated and the pre-computed weights are loaded.

3. Before entering the main outer loop (over the extracted frames), two service structures are created, in order to enable the switch from the *instant control* mode to the *memory control* one. In particular *rois_start* is set to have the same length as the

ROI list and two cells per position that will contain the first predicted probabilities for each ROI and class, which are used during the analysis as a comparison for the subsequent predicted probabilities. The second instantiated structure, namely *i_ctrl*, is meant to contain the actual boolean switch between the two aforementioned control modes. When the outer loop starts, the first check regards whether the frame index corresponds to the first one in the sequence: if the condition is satisfied there is no need for reloading the first image, since it was used to extract its initial resolution in the first step. Otherwise, any subsequent frame is loaded from the storage, normalized and resized based on the given resolution.

4. After the selection of the current image, the only inner loop over the ROIs starts; the first condition checks whether the ROI on duty comprehends the current image index in its user-defined observation interval. If the condition is met, the classifier is fed with the current frame and ROI mask couple in order to obtain both the 2 class probabilities and their transformation into the corresponding 3 labels. Afterwards, another check is done on whether the frame index corresponds to the first one in the specific ROI range: if so, the predicted probabilities are assigned to the corresponding slot in the *rois_start*, representing the term of comparison in case the *memory control* mode is selected.

5. The subsequent condition checks the type of ROI and assigns the corresponding reference label; in particular, if the type is not declared then the default one is *boundary*, while the *all_area* type must be explicitly specified according to the first version of the detection algorithm. If the ROI type is *boundary*, a check is made on whether the *instant control* mode is active or not. If the condition is met a variable containing the reference label to be compared with the predicted one is set to 1 and the instant to be assigned to the corresponding slot in the detection dictionary is set to the subsequent value with respect to the current one, due to the high sensitivity of the classifier to this particular class. If the *instant control* mode is switched off, a further control statement checks whether the current predicted probability corresponding to the boundary class is significantly higher than the one saved in the *rois_start* data structure: if so, the current instant is added in the corresponding detection dictionary. For what concerns the procedure for the *all_area* ROI type, the path is the same as the one described in the previous case, with the only difference laying in the fact that, if the *instant control* mode is active, the instant that is assigned to the detection dictionary in case the predicted label and the reference one correspond, is not increased by 1.

6. For each ROI, if the *instant control* mode is still active, if the predicted label corresponds to the reference one and if there is no time value that has already been set for that ROI in the detection dictionary, a further check controls whether the current instant is the initial one in the range that corresponds to the analyzed ROI. If the condition is met, likely the classifier failed to generate a reliable probability difference between classes, meaning that the *instant control* mode would have predicted the first frame in the range as the cause of the detection and said mode is switched off in order to base the subsequent analysis on the *memory control* mode. If the frame that caused the detection based on the *instant control* mode is not the first one, then the instant saved as in Point 5 is inserted in the detection dictionary.

Regardless of the approach in terms of model, it is evident that the presented detection phase is cheaper (in regards of resource allocation) and faster with respect to the previous version. Indeed, even without considering the time performance boost introduced by the classifier, the new algorithm presents only one inner cycle per frame and speculatively generates masks before they are actually used, as it is described in the second step of Algorithm 3. Moreover, the presented detection phase replaces both its counterpart in the default approach and the segmentation phase, dramatically simplifying the overall pipeline and reducing the video analysis time. Another point in favour of the presented detection process is that it provides a backup control mode which can be activated in case the default one makes unexpected predictions.

Algorithm 3 Detection algorithm with classifier

```
Require: $roi\_list$, $start\_frame$, $end\_frame$, $height$    ▷ (1.)
  $detections \leftarrow \{\}$
  $first\_frame \leftarrow$ read_from_memory($start\_frame$)
  $original\_size \leftarrow first\_frame$.shape(1)
  for each $roi \in roi\_list$ do    ▷ (2.)
      $detections[roi[name]] \leftarrow NULL$
      $roi\_mask \leftarrow$ draw_rois($roi$, $original\_size$)
      $roi[mask] \leftarrow$ resize($roi\_mask, height$)
  $classifier \leftarrow$ load_model_and_weigths()
  $rois\_start \leftarrow$ zeros(len($roi\_list$), 2)    ▷ (3.)
  $i\_ctrl \leftarrow$ ones_bool(len($roi\_list$))
  for each $t \in$ range(start_frame, end_frame) do
      if $t = start\_frame$ then
          $frame \leftarrow first\_frame$
      else
          $frame \leftarrow$ read_from_memory($t$)
      $frame \leftarrow$ normalize($frame$, $[0.3005, 0.2984, 0.3007]$, $[0.2593, 0.2479, 0.2682]$)
      $frame \leftarrow$ resize($frame$, $height$)
      for each $roi \in roi\_list$ do    ▷ (4.)
          if $roi[start\_frame] < t < roi[end\_frame]$ then Continue
          $label$, $probs \leftarrow$ classifier($frame$, $roi[mask]$)
          if $roi[start\_frame] = t$ then
              $rois\_start[i, :] \leftarrow probs$
          if $roi[type]$ is $NULL$ or $roi[type] =$ "$boundary$" then    ▷ (5.)
              if $i\_ctrl[i]$ then
                  $ref\_label \leftarrow 1$
                  $instant \leftarrow t + 1$
              else if $probs[0] =$ min($2 \times rois\_start[i, 0]$, 0.96) then
                  $detections[roi[name]] \leftarrow t$
          else if $roi[type] =$ "$all\_area$" then
              if $i\_ctrl[i]$ then
                  $ref\_label \leftarrow 2$
                  $instant \leftarrow t$
              else if $probs[1] =$ min($2 \times rois\_start[i, 1]$, 0.96) then
                  $detections[roi[name]] \leftarrow t$
          if $i\_ctrl[i]$ then    ▷ (6.)
              if $ref\_label = label$ and $detections[roi[name]]$ is not $NULL$ then
                  if $roi[start\_frame] = t$ then
                      $i\_ctrl[i] \leftarrow False$
                  else
                      $detections[roi[name]] \leftarrow instant$
```

Chapter 6

Test Suite and comparison

The presented modified approaches are quite different from each other in terms of how the problem is tackled. The first proposal aims at improving only some of the characteristics of the default methodology, such as generalization and the focus of the model capacity. By simply modifying the training process and the output shape, the rest of the detection pipeline the segmentation model is part of is almost kept unchanged, meaning that, for each frame, the aim is still checking whether a ROI condition is satisfied.

On the other hand, the proposed classifier needs several changes in how the frame is analyzed: in fact the number of steps in its detection pipeline decreases, together with the number of nested loops over the frames and the ROI set.

Given the diversities between the two presented proposals, the goal of this chapter is to compare them from different points of view. Since the problem statement mainly focused on the robustness of the model and the time performance, three tests are presented. The common idea in these experiments is that all of the described algorithms have the purpose of defining whether, in an ROI applied to an image, the described area does not contain an airbag, partially contains it or is completely covered by it. This common aim brings to the definition of a "challenge" statement that can be used to compare the different approaches performances. Regardless of how the goal is pursued, all of the presented algorithms are capable of predicting one of the three aforementioned states for a ROI, meaning that they can be compared in a classification challenge. Obviously, while the classifier approach is naturally predisposed for the challenge objective, the default model and its modified version need to be equipped with a procedure that ensures their compatibility. This procedure, however, is already given in a great portion by the subsequent steps of the pipeline of which the semantic segmentation models are part. Indeed, by only modifying the section of code that regards the output of the overall pipeline, it is possible to obtain a class label prediction, which is based on the semantic segmentation of the airbag in each image.

In order to perform the aforementioned three tests, a dedicated test suite has been developed in the form of a Python Notebook which uses the files for the various models and a support file containing various useful functions. In addition to the tests provided with said test suite, a further section shows three empirical experiments by directly employing the UI of the ***AI^{rbag}*** tool: the results are presented in the form of the images embedded in the generated report, depicting the polygonal ROIs and a sequence of frames around the detection point.

In the following description of the performed tests a nomenclature has been used to indicate the various approaches in a more direct way: **Classifier** stands for the approach based on the Classifier with forced attention, **Dec. Detector** indicates the modified algorithm based on the semantic segmentation phase which is characterized by the introduction of the Decolorization in its training process, **Detector** stands for the default algorithm.

The experiments carried out by means of the test suite are not representative in terms of realistic absolute time performances, since they are carried out on a different hardware with respect to the training processes presented in Chapter 4 and 5. In particular the employed hardware belongs to a simple Notebook PC equipped with Windows 10 OS and characterized by a Intel Core i7-10510U processor, supported by a 16GB RAM and a NVIDIA GeForce MX250 with 2GB of dedicated memory.

6.1 Classification with augmented dataset test

The first experiment is directly based on the same concept as the Translation step employed for the training phase of the Classifier approach in Section 5.2. In particular, for each sampled batch, the aforementioned process is used to create a balanced set of class labels, corresponding to a set of frame-ROI couples. The batches are sampled from the same Validation Set as the one that has been used for the training processes of the two introduced approaches, in order not to exploit any prior knowledge of images coming from the training phase.

The aim of this test is to compare the two introduced approaches and the default one by using the respective resolutions employed for the training samples. Indeed, the classifier model is trained with a lower resolution with respect to the segmentation approaches and this contributes to the time performance boost that the methodology provides, as described in Chapter 5.

For each batch, the three models are fed with the exact same sequence of input frames with some differences that regard the resolution and the way the ROI is provided to the algorithms. Moreover, the entire validation set is augmented in a way that ensures the same augmentation process for each parallel input, where "parallel" indicates the fact that the same image, with different resolutions, is used as input for the three algorithms in the same loop body. For what concerns how the ROIs are provided to the different algorithms, the idea is to emulate what happens in the respective production pipelines, in order to be as faithful as possible to the real performance of the different approaches once integrated in their production workflows. This means that, while the classifier directly takes as second input a binary mask representing the ROI, the algorithms based on the segmentation process need the same information in the form of a set of coordinates, which are then applied on each segmented image. In this regards the random ROI sampling presented in the Translation step described in Algorithm 2, when used for the segmentation based algorithms, also returns the coordinates which generated the created masks.

The test is repeated for a total of 30 epochs for each model, in order to collect stable and reliable information. The results are reported in the form of the classification accuracy, its standard deviation values across the epochs and total processing time in Table 6.1. The choice of reporting the total processing time instead of the average inference time is due to the fact that overhead in the adaptation of the masks or ROIs coordinates would affect too much such a small time value.

As Table 6.1 shows, both the presented methodologies introduce an improvement in

Random patches with fixed resolution test				
Algorithm	**Accuracy (%)**	**Standard dev. (%)**	**Resolution (px)**	**Time (s)**
Classifier	98.63	0.7	384×384	402
Dec. Detector	98.74	0.9	768×768	1858
Detector	83.80	1.5	768×768	1731

Table 6.1: Comparison between the Classifier algorithm, the introduced Detector with decoloration and the default Detector approach, in terms of Classification accuracy, Standard deviation, Working resolution and total processing time. The data are the result of 30 passes over a Validation Set with ROIs generated by an automated ROI proposal algorithm.

the classification performance with respect to the default approach. In particular, it is evident how the Classifier manages to reach comparable accuracy value with the one showed by the Dec. Detector, while working with lower resolution images and taking more than 4 times less time with respect to both the other approaches, supporting what is described in Chapter 5. Moreover, the lower standard deviation corresponding to the Classifier represents a more coherent behaviour along the test, with respect both to the Dec. Detector and the default one, which is the worst performing solution according to this metric.

6.2 Classification with varying resolution test

The second test is strictly linked to the first one in terms of how the ROIs are computed and provided to the three approaches. The aim, however, is symmetric to the previous one: the presented experiment shows the trend of the Classification accuracies with a working resolution that spans in a predefined interval. Indeed, in this case the Validation set is explored only once for each value of the resolution, since the goal is not to obtain a stable and average value of the accuracies but to analyze the behaviour of the models with varying image resolution and, therefore, with different resource allocations levels.

In order to keep the comparison as fair as possible, even with varying resolutions, with subsequent epochs, the Validation set augmentation is limited to random flips and rotations and it is kept frozen after the first epoch, with the aim of obtaining consistent comparisons along the resolution axis.

Figure 6.1 shows how the Classifier performs well with a wider range of working resolutions with respect to the Dec. Detector and the default Detector. This is due both to the lower training resolution and to the greater depth of the classifier's convolutional backbone. Indeed, both the Dec. Detector and the default Detector are trained with a higher working resolution, since they need the maximum possible pixel level precision in the segmentations in order to provide an affordable pixel mapping for the computation of the coverage in each ROI, as explained in Section 2.5.

Table 6.2 reports informations regarding the best accuracy achieved for each algorithm (even if the value itself is not fundamental) and the corresponding working resolution. The main thing to notice is that, while the Classifier's best accuracy is achieved with a working resolution that is not exactly the one used for the training phase, both the Detector models

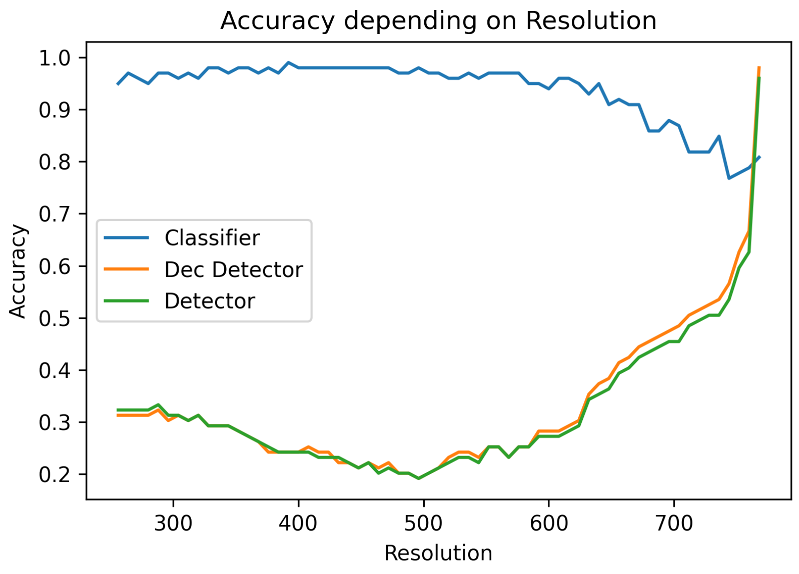


Figure 6.1: Representation of the trend of the Classification accuracy when it depends on the working resolution. The image describes the three models' outputs trend.

Fixed patches with varying resolution test		
Algorithm	**Best Accuracy (%)**	**Best Resolution (px)**
Classifier	98.99	392×392
Dec. Detector	97.98	768×768
Detector	95.96	768×768

Table 6.2: Comparison between the Classifier algorithm, the introduced Detector with decoloration and the default Detector approach, in terms of best Classification accuracy value achieved while varying the working resolution.

are strongly linked with the original training resolution, as it is evident by comparing this information with the classification accuracy peak in Figure 6.1.

6.3 Video simulation test with realistic ROIs

The presented experiment aims at simulating a real use case of the tool, meaning that a sequence of frames belonging to the same video is used as validation set. Even if it is not fundamental that all the frames belong to the exact same video, it is a practical way for experimenting the algorithms performances with the same label distribution as in a realistic situation. Indeed, in real airbag deployment videos, it is frequent that a significant part of the initial frames do not contain any section of an air chamber, since the explosion does not necessarily happen instantaneously. This means that real application performances could be far from the ones in perfectly balanced environments.

In this case, there is a meaningful difference in terms of how the ROI masks are provided to the algorithms: in fact, the ROIs are defined by a human, by introducing for the first time the idea of user awareness regarding the position of the ROIs themselves and their shape. Moreover, for the first time in the entire training and testing phases, polygonal (and not only rectangular) ROIs are employed: indeed, the *ROI proposal* process (employed in previous tests in Sections 6.1 and 6.2) introduced in the Translation step (Algorithm 2), generates only rectangular ROIs with the aim of speeding up the generation process itself. With these premises, the current test is an interesting benchmark for different reasons: understanding the behavior of the Dec. Detector and the Classifier when fed with realistic input frames and testing the knowledge transfer capabilities of the Classifier when the proposed ROIs have polygonal and not only rectangular shapes.

In order to perform an automated accuracy evaluation process by exploiting the code implemented for the Translation step, the frames are extracted from the original dataset, together with their manually segmented counterpart. Since it is highly possible that some of the frames have been used in the training phase for some of the models, before using these images as input for the algorithms under test they are subjected to a heavy augmentation process, involving transformations which are similar to the ones that have been used in Section 4.2 for comparison purpose. The test performs 10 epochs on the set of images, by randomizing the augmentation process not only for each pass but also for each single input image.

Video simulation test				
Algorithm	**Accuracy (%)**	**Standard dev. (%)**	**Resolution (px)**	**Time (s)**
Classifier	97.15	0.3	384×384	322
Dec. Detector	98.22	0.4	768×768	1395
Detector	81.01	1.1	768×768	1318

Table 6.3: Comparison between the Classifier algorithm, the introduced Detector with decoloration and the default Detector approach, in terms of Classification accuracy, Standard deviation, Working resolution and total processing time. The results are obtained through 10 epochs of simulation of a real use case, with ROIs defined by a human and frames belonging to the same video.

Table 6.3 summarizes the informations collected through multiple iteration of the described test over the extracted frames. It is evident that the heavy preprocessing performed during the training of both the Classifier and the Dec. Detector has defined a higher robustness, also in the case of realistic use cases. In particular, the Dec. Detector demonstrates an impressive improvement with respect to its default counterpart, with the only drawback of a slightly longer computation time. Both the Dec. Detector and the default one, however, confirm that they are extremely slower with respect to the Classifier time performance, which shows only a few percentage points less, in terms of Classification accuracy, than the best value. Moreover, Table 6.3 confirms the observation made in Section 6.1 regarding the standard deviation, which reaches its lower value with the Classifier results.

Figure 6.2 gives a visual representation of the analysis previously carried out. From a visual point of view is easier to understand how faster is the classification approach, with respect to the algorithms based on semantic segmentation. Moreover, the figure highlights the drop in the accuracy value of the default Detector if compared to the other algorithms.

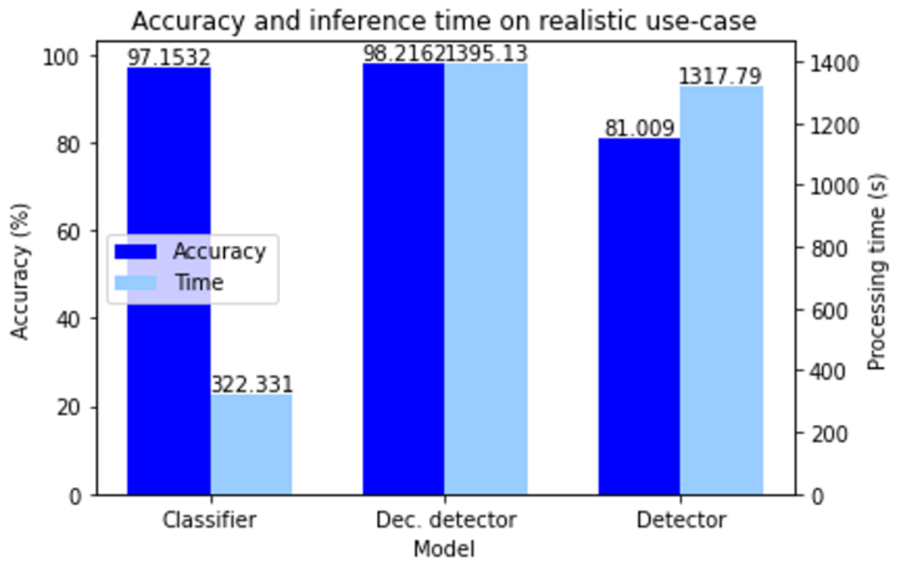


Figure 6.2: Visual comparison of the performances of the Classifier, the Dec. Detector and the default Detector in terms of Classification accuracy, together with the respective computation times after testing their behavior in a realistic use case, given by the dataset being composed by frames belonging to a single video and the polygonal ROIs being defined by a human.

6.4 Empirical tool test

In order to dive into the details of how the obtained results and performances can be translated into an actual use case, in this section the results of 3 tests by means of the Tool User Interface are presented, each one describing the output of the detection pipeline when integrated with the default Machine Learning model, its modified and retrained version (Chapter 4) and the Classifier with forced attention (Chapter 5). In particular, the experiment shows the real behaviour of the described detection pipelines in terms of detection accuracy when fed with polygonal ROIs. Again, it is worth to mention that the employed version of the Classifier with focused attention has not been trained with polygonal ROIs but with rectangular ones, preferring a greater variability of augmentation applied to frames, according to the usage limits imposed by AWS. For this reason, the performances presented in the following test and in the previous one rely on the Transfer Learning properties of the model.

First test The first experiment is performed with the combination of ROIs depicted in Figure 6.3, mainly focusing the analysis on two known problems: the upper ROI addresses the weakness regarding the combination of several colors close to each other, while the central ROI bounds high contrast areas and often misunderstood shapes. Both the ROIs are set to the *boundary* type, meaning that the detection should be fired as soon as the airbag enters the selected region. The bottom-right ROI is meaningless for the current analysis, since it is necessary for logging purpose in some production samples, containing the timestamp for each frame.

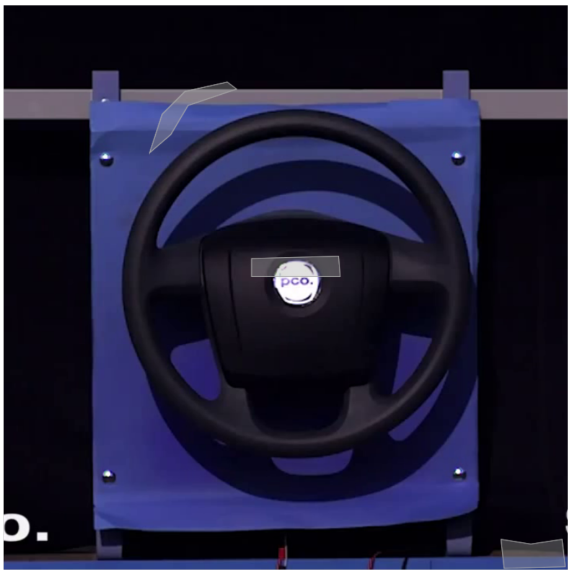
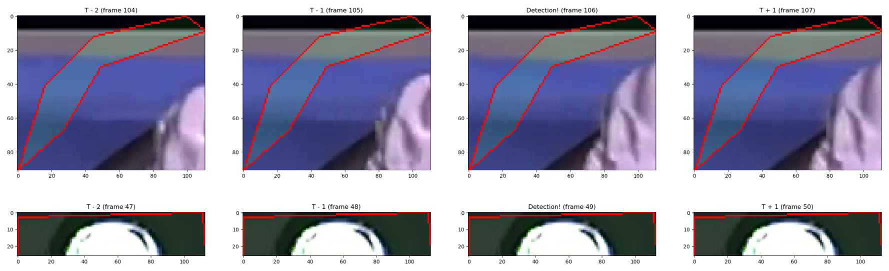


Figure 6.3: Starting image for the first empirical test: in overlay it is possible to see the polygonal ROIs drawn by the user through the UI. The first one covers a section of the support structure of the airbag while the second one is in the steering wheel logo area.

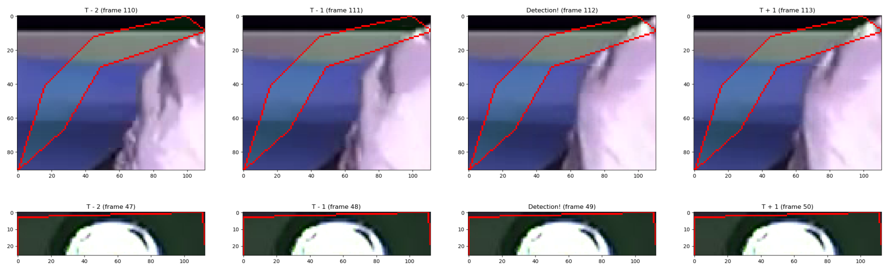
Figure 6.4 returns the same images as the ones that are presented in the report generated by the UI of the tool. From this sample on, the detection instant corresponds to the third frame in each sequence, since 2 prior and 1 following frames are also depicted. The comparison shows how the different solutions behave diversely in the same condition. Specifically, for what concerns the upper ROI, the default approach fails to reach an acceptable result, firing the detection some instants before the airbag even gets close to the ROI. However, for the aforementioned ROI, both the modified version of said approach and the one based on the classifier result in an acceptable detection instant, with the classifier output being the most precise in recognizing the exact frame that depicts the airbag touching the upper ROI boundary.

Figure 6.4 also enables a comparison on the boundary ROI located on the steering wheel logo. In this case both the segmentation based methodologies fail to detect the correct instant of invasion of the ROI by the airbag. Since the specified ROI is objectively complex to analyze due to the misleading high contrast areas, the correct prediction of the pipeline based on the classifier is due to the implementation of the "backup" control layer discussed in Section 5.4, which substitutes the simple label prediction and provides a detection based on the comparison of the probability prediction series corresponding to the ROI frames range.

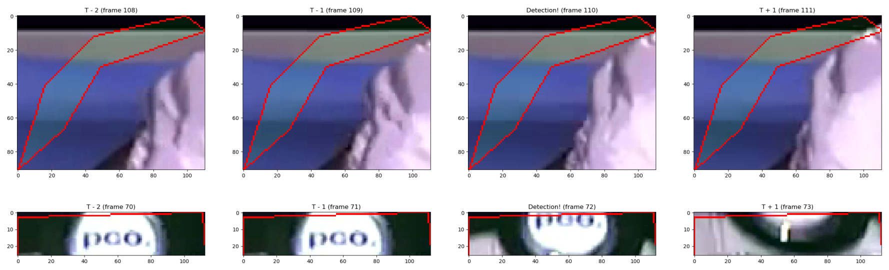
Second test The second experiment is performed by specifying the ROIs as in Figure 6.5. In this case both the regions bound sections that are characterized by shadows that alternate with the less dark color of the support frame and small sections of the steering wheel rim. The peculiarity of the choice lays in how the airbag flows in those areas, presenting poorly illuminated sections of the expanding air chamber in the lower ROI and high contrast effects in the side one. The lower ROI type is set to *all_area*, in order to focus the analysis on the sensitivity of each solution to low contrasts between the bag and



(a) Detection performed with the default approach.



(b) Detection performed with the modified approach.



(c) Detection performed with the Classifier with focused attention.

Figure 6.4: First comparison between the detection performed by means of the three solutions on the user-defined ROIs, as in Figure 6.3

the background while the ROI on the side keeps the *boundary* type.

The output of the detection process in Figure 6.6 offers an interesting picture of the general difference in the behavior of the two types of approach, in terms of addressed Machine Learning task. Indeed, far what concerns the side ROI (which is the vertical one in the aforementioned figure), the default solution misses the actual detection instant by a considerable margin, driven by the known misleading background color and lighting condition. Instead, both the Dec. Detector and the Classifier perfectly succeed in the goal of firing the detection with the correct frame.

The most interesting results, however, are given by the lower ROI detections in Figure 6.6: in this case the correct firing instant must be the one that corresponds to a full coverage of the ROI by the airbag. The default Detector and the Dec. Detector output the same instant, while the pipeline based on the Classifier detect the full coverage in the subsequent frame. By making a close analysis, the ideal prediction is the one resulted by the Classifier, since there is no empty pixel with the ROI boundary but there is a reason for the Detectors to share the same selected instant. Both the default solution and the

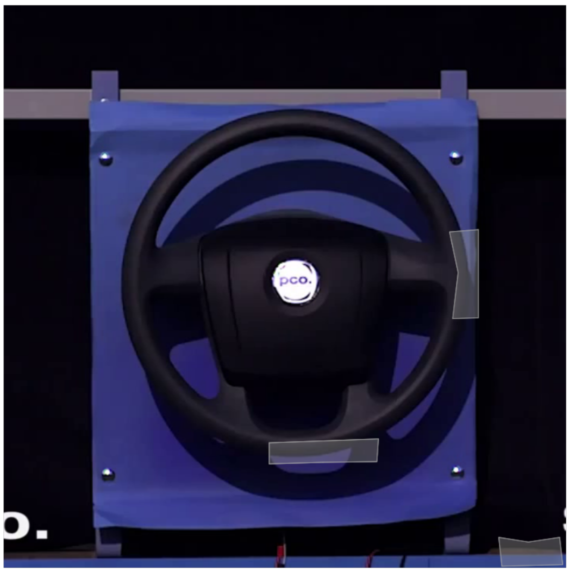


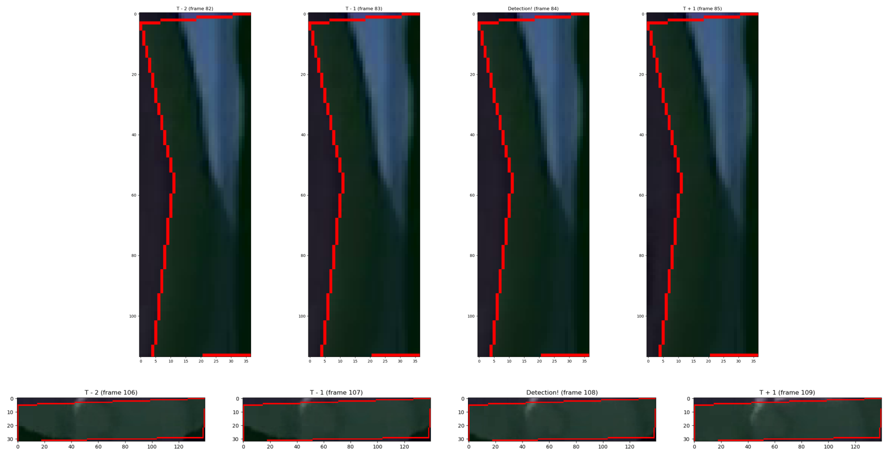
Figure 6.5: Starting image for the second empirical test. The first drawn ROI is on the right side of the steering wheel rim, characterized by a better light condition with respect to the second ROI, which is in the lower part of the steering wheel.

modified one include in their detection algorithm the usage of some pre-defined detection thresholds, which are set to fire the detection with almost the entire ROI covered by the airbag, while the classifier is trained to detect either full or partial coverage. Since said thresholds can be modified, the highlighted difference is not a problem due to the Machine Learning model but it can result in wrong detection in case the Segmentation process generates noisy binarized masks.

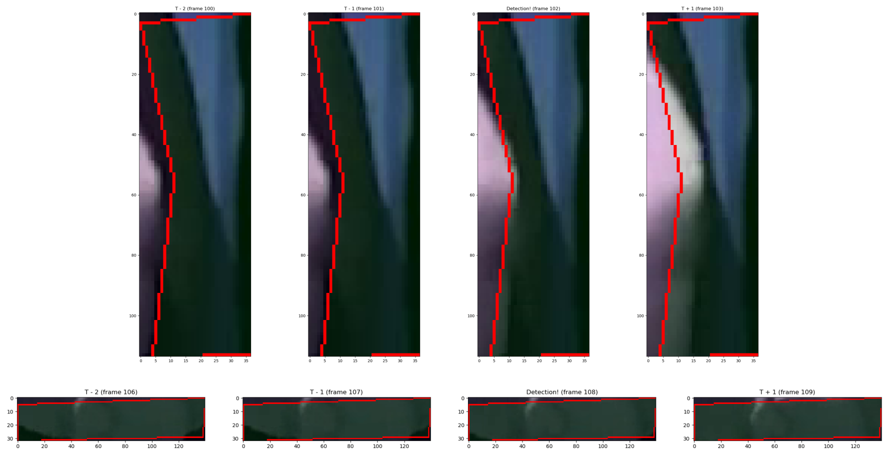
Third test The third and last experiment is based on the ROIs showed in Figure 6.7: both of them cross different section of the image, including the steering wheel and the support frame. They are placed in a symmetrical way with respect to the center of the steering wheel itself, represented by the logo, in order to address two different lighting conditions: in the upper left section the airbag is directly hit by the light, so the expanding end is clear and easily recognizable, while in the bottom right region the airbag is poorly illuminated, making it more difficult to recognize its edges. Specifically, the upper left ROI has type *boundary*, while the bottom right one has *all_area*.

As in the previous experiments, the resulting detection frames are collected in a report delivered through the UI and said frames are showed in Figure 6.8. For what concerns the results corresponding to the first ROI, the default Detector still fails to reach an acceptable range of detection instants, probably due to the false positive areas generated by the default model based to the brightness of some surfaces. On the other hand, both the updated solution and the one based on the classifier return a reasonable range of frames, with the first solution firing the detection in the previous frame compared to the second approach.

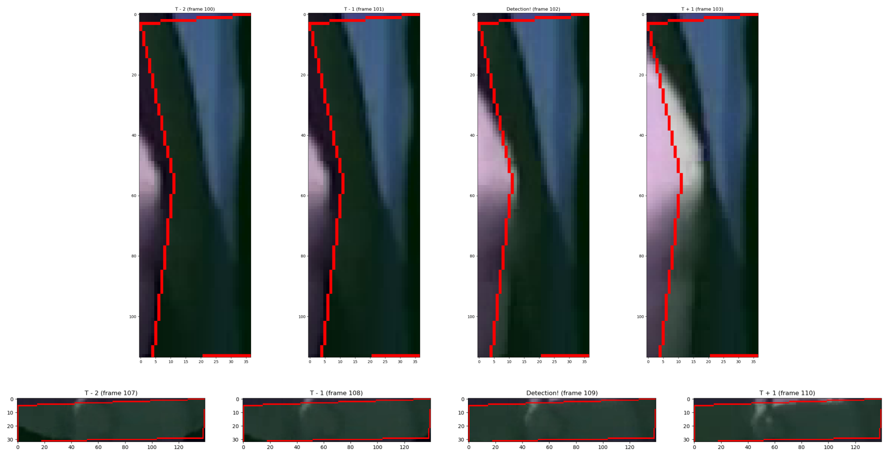
A similar analysis can be carried out regarding the second ROI results: in this case, the coupling between the type of ROI and worse light conditions compared to the previous ROI, represent a reason for a variety of scene interpretations. Again, the first solution seems not to perfectly recognize the bounds of the airbag due to the shadows on its



(a) Detection performed with the default approach.



(b) Detection performed with the modified approach.



(c) Detection performed with the Classifier with focused attention.

Figure 6.6: Second comparison between the detection performed by means of the three solutions on the user-defined ROIs, as in Figure 6.5

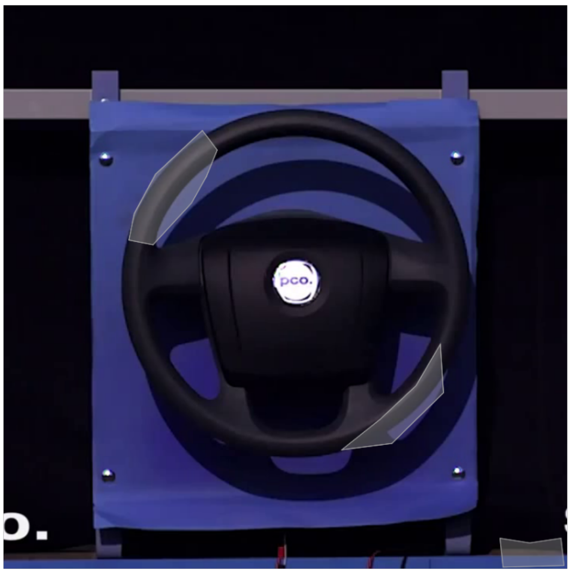
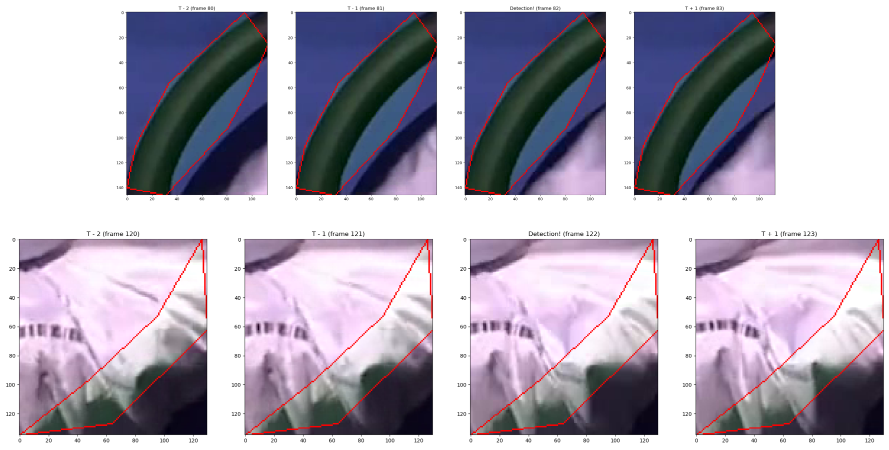
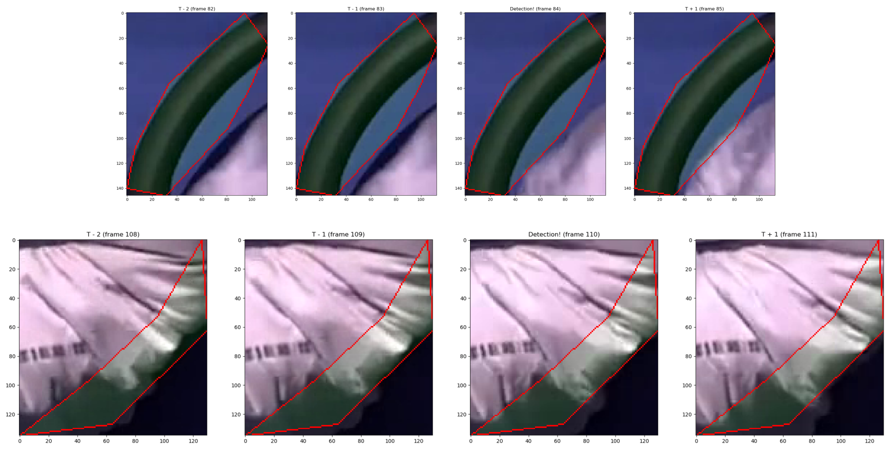


Figure 6.7: Starting image for the third empirical test. The ROIs are both positioned on the steering wheel rim, symmetrically with respect to the logo. The difference lays in how the light hits the airbag in those places as it is directed from the upper left to the lower right corner.

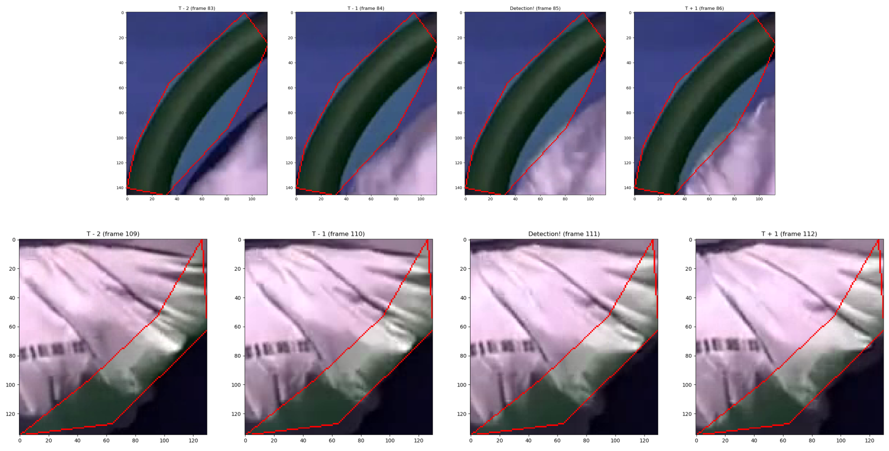
edges, definitely exceeding in terms of detection time if compared to the other approaches. Indeed, as for the previous ROI, the introduced solutions deviate for a single frame in their detection instants, which is probably due to the previously mentioned thresholds employed in the pipeline of the Detectors methodologies.



(a) Detection performed with the default approach.



(b) Detection performed with the modified approach.



(c) Detection performed with the Classifier with focused attention.

Figure 6.8: Third comparison between the detection performed by means of the three solutions on the user-defined ROIs, as in Figure 6.7

Chapter 7

Conclusion and Future Works

7.1 Conclusion

The premise of this Thesis work corresponds to the will of improving two aspects of the the current production version of the ***AI***$^{\boldsymbol{rbag}}$ tool, specifically regarding the robustness of the predictions and the performances in terms of inference time, as explained in Section 1.3. The tool is built in order to leverage on a Machine Learning backend, which represents the perimeter of the analysis and of the changes made in the presented work.

The default solution employs a Deep Convolutional Neural Network with two branches, namely Encoder and Decoder, aiming at generating a binarized version of the input image, which presents active pixels only for the ones that correspond to the airbag in the original image, following a Semantic Segmentation paradigm.

Given the limited cardinality of the original training dataset, the default model suffers from a lack of generalization, since it is applied with a wider range of light conditions and color combinations with respect to the ones known through the dataset.

The first improvement attempt regards the pre-existing model and presents a pre-processing operation that extends the augmentation phase, removing color aberrations and differences given by the variety of light conditions. Moreover, the model is modified in order to perform a real binary segmentation task: instead of learning both a background class and the airbag one, the updated version learns only the airbag class, reducing the dispersion of the learning capacity and the possibility to learn biases linked to known background conditions. The introduced innovations bring an improvement according to both an empirical segmentation analysis and to a repeated test based on the IoU metric.

The second proposal is completely detached from the existing solution, introducing a drastic difference in the working paradigm, since the entire pipeline is changed and optimized to work with a classifier, instead of including a segmentation phase. In order to support the training of this specific model, a translation module from the original dataset to a compatible version with the classifier has been presented. The classification is based on both an image extracted from an input video and a user-defined ROI, which are merged inside the model in order to get a class label that defines the absence, the presence or the complete coverage of the airbag inside the ROI. This solution introduces a relevant boost in terms of time performances, while showing good generalization capabilities.

When compared in the same classification task challenge, both the proposed solutions perform better than the original one. In particular, the optimized model based on the original solution is the best one in terms of classification accuracy, when applied on an augmented validation dataset with randomly proposed patches, while slightly increasing the inference time with respect to the default solution. The classifier proposal represents a good trade-off between classification accuracy and time performances, being the fastest algorithm when integrated in a production pipeline and providing an increased robustness level thank to the double prediction mode introduced in its detection process.

7.2 Future Works

Both the proposed approaches can be further studied from an optimization point of view, since the presented solutions are strictly bounded with the physical limitations of the project such as the time availability and the hardware employed for the training processes.

For what concerns the first proposal, the concept of a semantic segmentation task as the paradigm for the specified goal is still an effective idea, since it paves the way for a lot of additional features, such as the analysis of the maximum extension of the airbag. Given that, the training phase, together with the runtime data augmentation, is a challenging operation when combined with the availability and power of the used hardware and it required an intensive optimization work. By using a better hardware, it would be possible to study the behavior of more advanced and elaborated Deep Neural Models having the same goal as the employed ERFNet.
Moreover, another fundamental limitation to the work of improving the performances is the availability of significant data. The proposed study focuses on augmenting the current dataset and trying to make the model robust to the change of some patterns that, instead, are recurring in the dataset, such as background colors and light conditions. However, in the case much more data were available, the ideal procedure would be the one of training a specific version of the model for each environment in which it has to perform predictions, in order to specialize the model for each individual condition.

On the other hand, the Classifier with forced attention proposed in Chapter 5 depends on some innovations introduced in the training context, such as the Translation module and the Valve filters. The Translation module implies a random sampling operation of the ROIs' coordinates, meaning that this specific step represents the bottleneck of the entire training process. As in the previous proposal, the ideal solution would be performing the entire training process on ad-hoc datasets, characterized by the architecture described in 5.2 and dropping the actual translation step.
Furthermore, being the classifier solution studied and trained on the same hardware as the modified segmentation model, even in this regards the limitations given by the hardware prevent to employ more efficient and sophisticated CNN backbones, which could bring an improvement in the classification accuracy. Besides the architectural and data issues, the training process can potentially be improved by using multiple parallel tasks, introducing unsupervised algorithms such as solving Jigsaw Puzzles on input images [3] with the aim of improving the retention of information coming from the background even while forcing the attention in a specific ROI.

Bibliography

[1] SHUBHRA AICH, WILLIAM VAN DER KAMP, AND IAN STAVNESS, *Semantic binary segmentation using convolutional networks without decoders*, 2018.

[2] NANDITA BHASKHAR, *Interpreting logits: Sigmoid vs softmax*, Blog: Roots of my Equation (web.stanford.edu/ nanbhas/blog/), (2020).

[3] SILVIA BUCCI, ANTONIO D'INNOCENTE, YUJUN LIAO, FABIO MARIA CARLUCCI, BARBARA CAPUTO, AND TATIANA TOMMASI, *Self-supervised learning across domains*, 2021.

[4] MARIUS CORDTS, MOHAMED OMRAN, SEBASTIAN RAMOS, TIMO REHFELD, MARKUS ENZWEILER, RODRIGO BENENSON, UWE FRANKE, STEFAN ROTH, AND BERNT SCHIELE, *The cityscapes dataset for semantic urban scene understanding*, 2016.

[5] JIA DENG, WEI DONG, RICHARD SOCHER, LI-JIA LI, KAI LI, AND LI FEI-FEI, *Imagenet: A large-scale hierarchical image database*, in 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248–255.

[6] SAGI EPPEL, *Setting an attention region for convolutional neural networks using region selective features, for recognition of materials within glass vessels*, 2017.

[7] ——, *Classifying a specific image region using convolutional nets with an roi mask as input*, 2018.

[8] ALBERTO GARCIA-GARCIA, SERGIO ORTS-ESCOLANO, SERGIU OPREA, VICTOR VILLENA-MARTINEZ, AND JOSE GARCIA-RODRIGUEZ, *A review on deep learning techniques applied to semantic segmentation*, 2017.

[9] ROSS GIRSHICK, JEFF DONAHUE, TREVOR DARRELL, AND JITENDRA MALIK, *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2014.

[10] BEN GRAHAM, *Kaggle diabetic retinopathy detection competition report*, (2015).

[11] KAIMING HE, XIANGYU ZHANG, SHAOQING REN, AND JIAN SUN, *Deep residual learning for image recognition*, 2015.

[12] ALEX KRIZHEVSKY, ILYA SUTSKEVER, AND GEOFFREY E HINTON, *Imagenet classification with deep convolutional neural networks*, in Advances in Neural Information Processing Systems, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds., vol. 25, Curran Associates, Inc., 2012.

[13] JONATHAN LONG, EVAN SHELHAMER, AND TREVOR DARRELL, *Fully convolutional networks for semantic segmentation*, 2015.

[14] ADAM PASZKE, ABHISHEK CHAURASIA, SANGPIL KIM, AND EUGENIO CULURCIELLO, *Enet: A deep neural network architecture for real-time semantic segmentation*, 2016.

[15] ADAM PASZKE, SAM GROSS, FRANCISCO MASSA, ADAM LERER, JAMES BRADBURY, GREGORY CHANAN, TREVOR KILLEEN, ZEMING LIN, NATALIA GIMELSHEIN, LUCA ANTIGA, ALBAN DESMAISON, ANDREAS KOPF, EDWARD YANG, ZACHARY DEVITO, MARTIN RAISON, ALYKHAN TEJANI, SASANK CHILAMKURTHY, BENOIT STEINER, LU FANG, JUNJIE BAI, AND SOUMITH CHINTALA, *Pytorch: An imperative style, high-performance deep learning library*, in Advances in Neural Information Processing Systems 32, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds., Curran Associates, Inc., 2019, pp. 8024–8035.

[16] SHAOQING REN, KAIMING HE, ROSS GIRSHICK, AND JIAN SUN, *Faster r-cnn: Towards real-time object detection with region proposal networks*, 2016.

[17] EDUARDO ROMERA, JOSÉ M. ÁLVAREZ, LUIS M. BERGASA, AND ROBERTO ARROYO, *Erfnet: Efficient residual factorized convnet for real-time semantic segmentation*, IEEE Transactions on Intelligent Transportation Systems, 19 (2018), pp. 263–272.

[18] NIKOLAS VINCI AND DAVIDE CASINI, *Method and system for verifying the correct deployment of an airbag device*. Priority Date: 13 August 2018, Publication Date: 20 February 2020, Application number: PCT/IB2019/056836, Publication number: WO 2020/035772 A1, IPR Dossier n.A08.

[19] SERGEY ZAGORUYKO AND NIKOS KOMODAKIS, *Wide residual networks*, 2017.