

Article

Real-Time Oil Leakage Detection on Aftermarket Motorcycle Damping System with Convolutional Neural Networks

Federico Bianchi ^{1,*}, Stefano Speziali ¹, Andrea Marini ¹, Massimiliano Proietti ¹, Lorenzo Menculini ¹ , Alberto Garinei ^{1,2}, Gabriele Bellani ³ and Marcello Marconi ^{1,2}

¹ Idea-Re S.r.l., 06128 Perugia, Italy

² Department of Engineering Sciences, Guglielmo Marconi University, 00193 Rome, Italy

³ Department of Industrial Engineering, Alma Mater Studiorum Università di Bologna, 40126 Bologna, Italy

* Correspondence: fbianchi@idea-re.eu

Abstract: In this work, we describe in detail how Deep Learning and Computer Vision can help to detect fault events of the AirTender system, an aftermarket motorcycle damping system component. One of the most effective ways to monitor the AirTender functioning is to look for oil stains on its surface. Starting from real-time images, AirTender is first detected in the motorbike suspension system, simulated indoor, and then, a binary classifier determines whether AirTender is spilling oil or not. The detection is made with the help of the Yolo5 architecture, whereas the classification is carried out with the help of a suitably designed Convolutional Neural Network, OilNet40. In order to detect oil leaks more clearly, we dilute the oil in AirTender with a fluorescent dye with an excitation wavelength peak of approximately 390 nm. AirTender is then illuminated with suitable UV LEDs. The whole system is an attempt to design a low-cost detection setup. An on-board device, such as a mini-computer, is placed near the suspension system and connected to a full hd camera framing AirTender. The on-board device, through our Neural Network algorithm, is then able to localize and classify AirTender as normally functioning (non-leak image) or anomaly (leak image).



Citation: Bianchi, F.; Speziali, S.; Marini, A.; Proietti, M.; Menculini, L.; Garinei, A.; Bellani, G.; Marconi, M. Real-Time Oil Leakage Detection on Aftermarket Motorcycle Damping System with Convolutional Neural Networks. *Sensors* **2022**, *22*, 7951. <https://doi.org/10.3390/s22207951>

Academic Editor: Jianbo Yu

Received: 12 September 2022

Accepted: 15 October 2022

Published: 18 October 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: sensors; computer vision; convolutional neural networks; AirTender; fluorescence

1. Introduction

Nowadays, automotive diagnostics plays a significant role in ensuring vehicle driving safety and comfort. In particular, to guarantee the driver's safety, it is crucial to monitor constantly the chassis system components, such as the damping apparatus. The most immediate and coarse methods to check the health state of the damping system are provided by the driver's perception and, more generally, human inspection. Other more quantitative approaches for fault detection and isolation are classified as reliability-based, model-based, signal-based or statistical-based. See [1] for details.

Having an automated system that takes care of the health state of a vehicle is important also because of the increasing demand of public services such as shared mobility or car rentals where human inspection is not always possible. Traditionally, automotive diagnostics has often relied on "on-board diagnostics", where sensors allow generation of data within a vehicle which are then processed for diagnostic purposes. See [2] for a survey on connected vehicles.

Sensors can record signals, such as vibrations or noise, and algorithms suitably designed should process such data to benchmark the current status of the vehicle against standard conditions previously defined [3]. This approach has many advantages and, in general, its purpose is to help detect and isolate fault events. When failures are not properly detected by on-board diagnostics (or are detected with delay), one could also consider off-board diagnostics [4], which is interesting in its own right. We will not be exploring off-board diagnostics in this paper. Rather, we focus our attention on an on-board

diagnostic system suitably designed to detect oil leakages from a special type of motorcycle suspension system introduced shortly.

An outstanding paradigm for detecting faults is that of Machine Learning (ML), where data coming from sensors can be fed to algorithms able to detect (with a certain degree of certainty) fallacies or anomalies. Deep Learning (DL) in machine health monitoring [5] often concerns the use of ML architectures such as Auto-Encoders, Restricted Boltzmann machines, Convolutional Neural Networks, and Recurrent Neural Networks. They all provide outstanding models according to the task at hand. We give more details and references for how DL is used in automotive diagnostics in the next section.

As mentioned above, visual inspection or the driver's perception are the first parameters taken into account for the vehicle monitoring. However, it is often the case that perception is not enough to detect minor fault events. A more systematic analysis of the suspension system behavior, based on data collected during normal functioning, may help considerably in this case. A first-class approach consists in feeding Neural Network signals coming from sensors installed on the vehicle. In particular, Convolutional Neural Networks (CNNs) turn out to be good feature-extractors and quite robust under changing conditions [6]. However, vibration analysis usually relies on the physical installation of suitable accelerometers on the vehicle. Such sensors may sometimes not be available or have an elevated cost. Moreover, in the case of suspensions such as those studied in this work, one of the main fault precursors is oil spilling, which usually occurs even before a major break takes place.

In this work, we focus our attention on oil leak detection in an aftermarket suspension system—AirTender—via Computer Vision and CNNs. AirTender is a shock absorbing system produced by UmbriaKinetics (for more details on AirTender, visit the website <https://airtender.it>, accessed on 11 September 2022). In short, AirTender is a system that combines the advantages of the traditional coil spring system with that of the air suspensions present in today's motocross. The body of AirTender constitutes the core of the system: it is made of two annular elements (cylinders) sliding one inside the other and forming an oil chamber, pressurized by the compressed air present in the external cylinder.

The strategy we follow for leak detection is rather simple and can in principle be applied to other suspension systems as well. Our goal is to determine whether a motorbike suspension is experiencing oil leakage close to the main damper by exploiting UV oil fluorescence. As an aside, oil leakage detection from various types of equipment and starting from oil fluorescence has been already widely discussed in the literature. See, for example [7–10]. For our purposes, we set up two Convolutional Neural Networks (CNNs) working in sequence, one that localizes the core of AirTender in the suspension system by drawing a bounding box around it, and the second one which classifies the content of the bounding box as a leak/non-leak AirTender.

From a practical point of view, the idea is to place an on-board device such as a mini-computer near the suspension system. A full hd camera connected to the on-board device frames AirTender and the surrounding environment (tarmac, wheels, vehicle body, etc.). The on-board device, through our Neural Network algorithm, should then be able to detect/localize AirTender. After AirTender has been detected correctly, it undergoes a classification process as normally functioning (non-leak) or anomaly (leak) and, in case of an anomaly, informs the driver of the fault.

The setup is rather easy to put together and is illustrated in the main body of the paper. The final goal is to have an automatic detection system, part of the on-board diagnostics, able to spot oil leaks on the suspension system. As this is a first study, all tests have been made on the test bench showing promising results.

This paper is organized as follows. In Section 2, we briefly describe the broad domain of vehicle diagnostics and the problem of detecting suspension system failure. We also give a short review of ML-based fault diagnosis methods. Section 3 is the core section of this paper. We first briefly summarize the basics of Convolutional Neural Networks, with focus on their application in computer vision. Then, we present the experimental

setup and its main features along with the details of how we prepared our own dataset, needed to train our algorithm to spot oil leaks. Finally, we lay out the method we used to train our algorithm and we spell out in detail the network architectures we employed. In Sections 4 and 5, we give the results and conclusions of our analysis.

2. Related Work

Over the years, system health monitoring has been part of the set of activities performed on a system to keep it in good working condition [1].

System monitoring can either be limited to the observation of the current status of a system or can be assisted by predictions of future operating states as well as future failure states. In the former case, maintenance comes to play after an observation suggests it, while in the latter case, it can also play a preventive role.

In general, predictive diagnosis and prognosis are motivated by the need for manufacturers to optimize system performance and reduce extra costs due to unwanted disruptions. Thus, it does not come as a surprise that system health monitoring has been the focus of research in the recent years.

In this paper, our goal is to discuss an approach to monitor the health of a suspension system. Standard approaches in monitoring the health state of the damping system of a vehicle are often model-based [11–13] or signal-based [14]. A model-based approach usually relies on the comparison between the measured signal (e.g., periodic vibration signals captured by the accelerometers) and the estimated values generated by a mathematical model of the system, while a signal-based approach focuses on detecting changes or variations in a signal with subsequent diagnosis of the change.

However, one crucial problem in automotive diagnosis is robustness with respect to vehicle configuration and changing conditions (mass variations, different tire characteristics, etc.), usually captured as driving data. As a matter of fact, model-based and signal-based approaches do not immediately take advantage of such data, if not with a fine-tuning method. Therefore, in the last few years, data-driven approaches leveraging deep learning models have come to play a central role in current research [5].

As mentioned above, when dealing with machine health monitoring systems, it is customary to use signals coming from sensors, such as accelerometers or potentiometers and give them as input to Neural Networks. Some authors operate directly on raw-data (i.e., no pre-processing on measured signals), while others implement some transformation, such as scaling, denoising, Fourier transformation, etc.

For example, in [15], a CNN consisting of two convolutional and pooling layers followed by a dense layer was trained on raw data coming from multiple sensors. In addition, the authors of [16] trained a CNN consisting of three convolutional layers and two sub-sampling layers followed by a dense layer taking directly raw time-series sensor data as input. In [17], the authors combined the concept of “inception” and one-dimensional deep convolutional neural networks to train a CNN to operate directly on raw vibration signals.

On the other hand, the authors of [18] performed a frequency analysis before feeding data to a CNN architecture. Several types of bearing faults, along with a healthy bearing and rotor imbalance, were considered. For each condition, several bearings were tested.

Another common approach is to create grayscale images (2D) by reshaping time signal vectors to 2D tensors. For example, the authors of [19] proposed a new CNN, based on LeNet-5 [20], for fault diagnosis. They were able to extract features of converted 2D images and eliminate the effect of handcrafted features. The proposed method is tested on three famous datasets, including the motor bearing dataset, self-priming centrifugal pump dataset, axial piston hydraulic pump dataset. In addition, ref. [21] performed preprocessing of 1D temporal vibration signals by generating 2D grayscale images. Then, classification was carried out by exploiting a CNN made of two convolutional layers followed by a pooling layer.

Other preprocessing methods that have become standard in health monitoring in Machine Learning comprise Wavelet Transform (WT), Short-Time Fourier Transform (STFT) or Hilbert–Huang Transform (HHT). See, for example [22–25], where CNN architectures are employed on preprocessed time series data.

It is worth noting at this point that, in [26], CNNs played a major role also for the diagnosis of automotive semi-active suspension components. The authors fed a CNN (described in the paper) both raw and preprocessed data coming from seven different sensors (four wheel speeds, lateral and longitudinal accelerations, and yaw rate). Inputs are given as both 1D and 2D tensors.

Even though CNNs applied to (preprocessed) actual driving data promise to give excellent results, our approach is different. As brought forward in the Introduction, the aim of the present study is to set up a convenient “visual system” able to detect fault events (through oil leakages) with a camera framing the suspension system. To this end, we have CNNs processing actual images of the suspension system described in detail in the next section.

3. Proposed Approach

In this section, we describe the proposed approach for oil leak detection. This is the longest section of the paper. We first give an overview of CNN architectures and how they work in general. Then, we describe our experimental setup and dataset and finish off by laying out the details of the method we used for classification and the CNNs we employed for the analysis of this paper.

3.1. Overview of CNNs

Convolutional Neural Networks are known to be good feature extractors, especially when dealing with multidimensional data as inputs. They came to popularity as a method to classify images, and the first working prototype was proposed by LeCun [27] for image processing. A comprehensive overview of CNN lies outside the the scope of the present article. The reader who is interested in learning the nuts and bolts of CNNs is referred to the excellent books [28,29] or the review [30] and references therein. For a more formal treatment of CNNs and deep learning in general, see [31,32]. Very useful is also the thesis [33]. Here, we just summarize their main features and how they can be useful for our purposes. The reader that already knows CNNs and their application to Computer Vision can safely skip this part.

3.1.1. Structure of CNNs

The building blocks of Convolutional Neural Networks are *convolutional* and *pooling layers*. A convolutional layer computes the convolution (a mathematical operation that we define shortly) of the input image with a stack of filters. Downsampling or pooling layers, instead, coarse-grain the input preserving locality and spatial structures.

In Machine Learning, it is customary to think of images as functions on a two-dimensional compact domain, $f : \Omega \rightarrow \mathbb{R}^C$, where C is often referred to as the number of channels and is equal to 1 for grayscale images, while it is equal to 3 for color images. Typical examples of compact domains are the Euclidean rectangle $\Omega = W \times H$ or S^2 for spherical CNNs. Here, W stands for the width and H for the height of the input image (in practice, Euclidean domains are always discretized by means of a regular grid). Such a construction naturally suggests a real vector bundle structure for input images, where the base space is the domain Ω and the fiber is simply \mathbb{R} for grayscale images or \mathbb{R}^3 for color images (here, the group of symmetry transformations on images usually corresponds to a subgroup of $GL(2, \mathbb{R})$ or the translation group in two dimensions. Elements of the fibers usually belong to the trivial representation. In addition, the bundle is often trivial [33]). Function images are often required to be square-integrable over Ω , and therefore belong to $f \in L^2(\Omega)$. In a computer vision setting, we seek for an unknown function $y : L^2(\Omega) \rightarrow \mathcal{Y}$ defined on a training set of images $\{f_i \in L^2(\Omega)\}$. For instance, when dealing with image

classification, \mathcal{Y} can be thought of as a discrete set with $\text{rank}(\mathcal{Y})$ the number of classes, whereas if dealing with object localization, \mathcal{Y} is a multi-dimensional simplex.

Let us give a little more mathematical detail about convolution and pooling, as promised. Here, we heavily draw from [31,32]. A convolutional layer is a layer of the form $g = C_{\Gamma}(f)$. Essentially, we act on a p -dimensional input image $f(x)$ with a bank of filters $\Gamma = (\gamma_{k,k'})$, producing a q -dimensional output g ,

$$g_k(x) = \zeta \left(\sum_{k'=1}^p f_{k'} \star \gamma_{k,k'}(x) \right). \quad (1)$$

Here, $k = 1, \dots, q$ and $k' = 1, \dots, p$ while ζ is a nonlinear pointwise activation function, such as the often employed ReLU activation function, $\zeta(x) = \max(0, x)$ [34]. The output g is usually called *feature map*, which, roughly highlights those areas of the input that most activate the filter (γ). The convolution operation is defined for a planar CNN as

$$(f_{k'} \star \gamma_{k,k'})(x) = \int_{\Omega} dx' \gamma_{k,k'}(x') f_{k'}(x - x'). \quad (2)$$

The filters γ correspond to the *shared weights*. In a convolutional network, these weights correspond to the free parameters that are tuned by the learning algorithm in order for the network to extract the most important features of the input image. This allows the network to focus on—and learn—the important features of the input image.

Besides convolutional layers, we have pooling layers. They correspond, roughly, to layers which reduce the size of feature maps, decreasing, in turn, also the model parameters without losing too much information about the input. Their goal is to sub-sample the input image in order to reduce the computational burden. Aside from that, they also introduce to some level a notion of *translational invariance*, one of the hallmarks of CNNs. Pooling layers can be defined as

$$g_k(x) = P\{f_k(x') : x' \in \mathcal{N}(x)\}, \quad k = 1, \dots, q, \quad (3)$$

where $\mathcal{N}(x)$ is a neighborhood of x in Ω and P is a permutation-invariant function. Common pooling operations are *max pooling* or *average pooling* [35,36].

A convolutional network is then put together by assembling a stack of convolutional and pooling layers, $U_{\Theta}(f) = (C_{\Gamma^k} \dots P \dots C_{\Gamma^1})(f)$, where $\Theta = \{\Gamma^1 \dots \Gamma^K\}$ is the vector of parameters of the network (shared weights) (pooling layers are not characterized by learning parameters).

In a supervised setting, the CNN parameters can be learned by asking for the minimization of some loss function measuring the “distance” between the outputs ($U_{\Theta}(f_i)$ ’s) and the labels y_i ’s, $\mathcal{L} = \sum_{i \in \mathcal{I}} L(U_{\Theta}(f_i), y_i)$. If the model is complex enough and the dataset representative, the algorithm is expected to generalize outside the training set [37].

Usually, convolutional and pooling layers are followed by dense (fully connected) layers and a classifying activation function, such as softmax or sigmoid, depending on the goal of the CNN. We employ precisely such a structure (con-pool layers followed by dense layers) later in the paper when classifying the leak/non-leak status of AirTender. More modern architectures, especially when performing segmentation or object detection, contemplate the use of Fully Convolutional Networks (FCNs) [38].

In the present paper, we mainly deal with two CNN-type architectures, the Yolov5 [39] (introduced later) for object localization and a suitably designed CNN (referred to in the following as OilNet40) created to classify the status of AirTender. Full details are given in the following sections.

We would like to stress at this point that CNNs have not only been successful to process images, but have also been used to address sequential data, including Natural Language Processing and speech recognition. See, e.g., [28] for pedagogical details and

further references. However, given the goal of this paper, we focus more on their power to process physical images.

3.1.2. CNNs for Computer Vision

Computer Vision deals with how computers can derive high-level meaningful information from digital images or videos. The synergy with Deep Learning, through DL architectures such as CNNs, has produced dramatic results in recent years. This is essentially due to the application of CNNs through 2010 to 2017 to the ImageNet Large Scale Visual Recognition Challenge, or ILSVRC [40]. Tasks of ILSVRC usually comprise *image classification*, *image localization*, and *object detection*.

In computer vision, classification usually refers to predicting the classes of objects present in an image. The difference between object localization and detection is a little subtle. Object localization aims at locating an object in an image, while object detection tries to find out all the objects and their boundaries. In this paper, the terms localization and detection are used interchangeably as they define a common task in our algorithm, even though this is not standard practice.

Over the last few years, different convolutional CNN architectures have been proposed. Among these, those that came to prominence for their degree of accuracy are LeNet-5 [20], AlexNet [41], GoogLeNet [42], VGGNet [43], ResNet [44], Inception-v4 [45], Xception [46], GBD-Net [47], SENet [48]. They constitute a milestone in the field of Deep Learning applied to Computer Vision. For a detailed account on the genesis of all these architectures and their structure see, for example, [28]. By leveraging CNN architectures such as those just introduced, people were able to design Neural Networks for object detection as well. Famous deep learning architectures for object detection are: Yolo [49–51], SSD [52], Faster R-CNN [53], Mask R-CNN [54]. For a detailed account on the different object detection architectures, see [55]. In this paper, we use a Yolo architecture for our object detection task.

3.1.3. CNNs and Transfer Learning

It is often the case that training a deep neural network takes a huge amount of time and computational power, and very few companies have such considerable resources. Moreover, it is often wise to store the knowledge gained, so that we re-use a trained model or build on that model. This is in essence what *transfer learning* is for.

Many famous CNN models, such as AlexNet, GoogLeNet, ResNet, InceptionNet, VGGNet, etc., have been released as part of useful packages such as the Torch Vision library in Pytorch or Keras in Tensorflow. These models can be loaded directly to perform inference on the same (or similar task) they were trained for, or they could even be used as baseline for training in related problems. This is in essence what we do when training the Yolov5 to detect AirTender: we load the Yolov5s network with pre-trained weights and train it to identify AirTender. On the other hand, we are not using transfer learning for the classification task, as random weight initialization is sufficient.

The next section is related to the experimental setup.

3.2. Experimental Setup

The problem with oil spilled on a opaque surface is that it is oftentimes hardly visible, even to a human eye. To make it more easily detectable, we employed a method often used to spot oil leaks: observation of emitted fluorescence when the oil is irradiated with ultraviolet (UV) light. In order to amplify the UV fluorescence, we also diluted the oil with fluorescent dye. Following the specifications on the mixing of the dye (max percentage of 0.7% of dye), oil mechanical features are not compromised. This, in fact, turned out to be a crucial step to make the oil more clearly visible by the camera and, in turn, more detectable.

We also assembled a simple setup with 4 UV LEDs emitting light at of 385 nm along with other 5 UV LEDs emitting at 400 nm wavelength. The LEDs we used emit directional light, with an opening angle of 15 degrees. They were powered with a 12 V battery, consuming 0.6 W.

The detection of fluorescence emitted by oil leaks irradiated with UV light has been discussed before in the literature. See, for example [56], (and references therein) for oil leak detection with solar irradiation and image processing using deep learning. It is often the case that observing fluorescence in conditions of high illumination is not an easy task, essentially because the noise produced by daytime light makes fluorescence hardly visible. However, AirTender is hardly ever exposed to direct sunlight when fully assembled on a motorcycle. Thus, we reckon that in condition of moderate illumination (50–2000 lux), oil fluorescence from UV light should be clearly visible.

The remaining part of the experimental setup consists of an OrangePi 3 mini-computer, with a 64-bit 1.8 GHz ARM processor, a multi-core GPU Mali T720 and 2GB of RAM. A full-HD camera is connected via USB to the OrangePi. The UV LED box is connected indirectly to the 12V battery via a relay, driven by the OrangePi. When collecting photos, the relay is turned on and the suspension system is enlightened with UV light (Figure 1). The OrangePi runs a python-based algorithm to detect and classify the AirTender, exploiting the TensorFlow Lite runtime, a library specifically designed to fit small computers, as mobiles and edge computing devices (see <https://www.tensorflow.org/lite>, accessed on 11 September 2022, for full details on TF Lite). Detection and classification algorithms elaborate real-time images acquired by the camera.

We now proceed to discussing how we created the dataset to train our networks to distinguish a leaking AirTender from a non-leaking one.

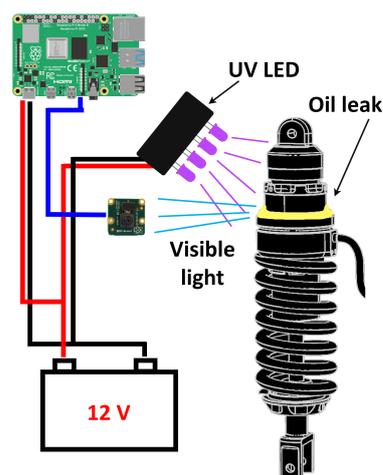


Figure 1. Setup to find oil leaks on the AirTender suspension system. A camera connected to an on-board device records the core of AirTender, while UV LEDs make oil fluoresce. The LEDs and the on-board device are powered by the on-board battery. The on-board device elaborates the images recorded by the camera through the neural networks trained for this study.

3.3. Dataset

In this section, we give some details on how the dataset for AirTender detection/classification was created.

As is well-known, CNNs require a large number of images for their training. In addition, the dataset should be as diverse as possible, in order for the CNN to generalize well outside the training set. At the same time, in a supervised learning setting, such images require labels. Producing labels for computer vision tasks such as object detection, classification or even segmentation can easily become challenging and time consuming. In these cases, techniques such as transfer learning or data augmentation can come to the rescue, and simplify the process of creating a representative dataset. We discussed briefly what transfer learning is in Section 3.1.3, while we touch on data augmentation in Section 3.4.4.

We made our own dataset for AirTender detection as well as for classification (leak or non-leak images). In an object localization task, labels correspond to the bounding boxes

around the object we want to locate, while in a binary classification task such as ours, one would simply associate one of the two classes to each image, leak or non-leak in the present case.

We generated the dataset by extracting photograms from a number of videos of AirTender as well as by taking pictures of AirTender directly. The collected images are aimed at representing AirTender in different conditions: different motorcycle setups, different backgrounds as well as different conditions of lighting, orientations, etc. More importantly, some of the collected images should simulate oil leakages on AirTender.

Eventually, we collected 1634 different images which were divided into a training, validation and test set for the training.

We ended up having a training set made of 1044 images, a validation set made of 413 images, and a test set made of 177 images. All images have dimension $1300 \times 1300 \times 3$, meaning that they are color images. In addition, they correspond to images taken in condition of illumination in the range 50–2000 lux. As this is a first study, we collected our dataset trying to represent AirTender in as many different conditions as possible, but of course, a more refined dataset could be taken by considering even more different setups, different image quality, and so on. We leave the creation of a larger dataset for the future, limiting ourselves to having a fully working algorithm.

Labels for object detection can be created using standard tools such as LabelImg, see, e.g., [57]. Alternatively, one can rely on web User Interfaces such as that provided by Roboflow (<https://docs.roboflow.com>, accessed on 11 September 2022). (The script we used is available at https://github.com/ideare-ds/tools_for_Computer_Vision, accessed on 11 September 2022). When producing labels for a Yolov5 network, we have to make sure to save the labels (one for each image) as a text file (.txt). The annotation file consists of a string of five numbers. The first number identifies the class the object to detect belongs to. The remaining four numbers identify the coordinates of the center of the bounding box (x and y) as well as the width and height of the box. All numbers are normalized to lie between 0 and 1.

Notice that we aim at building a classifier that distinguishes a leaking AirTender from a normally functioning one. Thus, we should prepare a dataset with a balanced number of images of Air Tender on full display with either spilled oil on it (faulty AirTender) or just a normal Air Tender with no oil leakage and thus working well. Indeed, when training a classifier, leak or non-leak images are associated to two different classes.

Let us move on to discussing the algorithm methodology to classify leak/non-leak images.

3.4. Method

In this section, we spell out the details of our method to detect oil leaks on AirTender. We first lay out the steps we followed to train our algorithm and then discuss three methods to improve training: image preprocessing, data normalization, and data augmentation.

3.4.1. Workflow

To determine whether the AirTender is leaking or not, we want to propose a two-step process. We first spatially localize the AirTender in the suspension system of the motorcycle: a camera placed near the suspension system will, in general, record images of AirTender and its background and, thus, it is important to first detect and isolate AirTender from its surrounding. Then, the resulting detected images are passed through a classifier which labels leak or non-leak images.

The steps we followed to have a fully working algorithm for oil leak detection are as follows. We start with our dataset, the creation of which has been discussed in the previous section. We first apply some data augmentation (see Section 3.4.4 for details) in order to have a larger and more diverse dataset. Images are then passed through the Yolov5 (training) which should localize AirTender within the image by enclosing it in a bounding box.

The content of the bounding box (supposedly AirTender) is then cropped and preprocessed. The resulting image passes through the binary classifier (OilNet40) which outputs the leak/non-leak status of AirTender.

3.4.2. Image Preprocessing

In the previous sections, we discussed a method to amplify oil fluorescence. We might wonder whether there is more we can do to make oil leakages even more visible. It turns out that some image preprocessing can be helpful to highlight the spilled oil, especially if we could improve contrast to highlight oil fluorescence.

In this work, we relied on the application of Contrast Limited Adaptive Histogram Equalization (CLAHE) to input images. CLAHE is a variant of the Adaptive Histogram Equalization (AHE) which prevents over-amplification of the contrast, and operates on small regions of the image, also called tiles, rather than the entire image.

It turns out that histogram equalization techniques, such as CLAHE, are implemented in OpenCV [58]. It is possible to choose parameters such as the clip limit, i.e., the value at which the histogram is clipped, and the tile size.

In order to apply CLAHE to all input images, we first convert RGB images to HSV (Hue, Saturation, Value) space. We then apply CLAHE to the channel H (Hue) and convert back to RGB colors. This helps highlight fluorescence of oil leaks making them more visible.

Besides color images, we also applied CLAHE before and after converting color images (RGB channels) to grayscale (1 channel) images, producing two different datasets for a total of 4 different presentations of the same dataset. See Figure 2 for the four different preprocessing methods applied to a normally functioning AirTender (left) and a faulty AirTender (right). Eventually, we test our classification algorithm on all datasets.

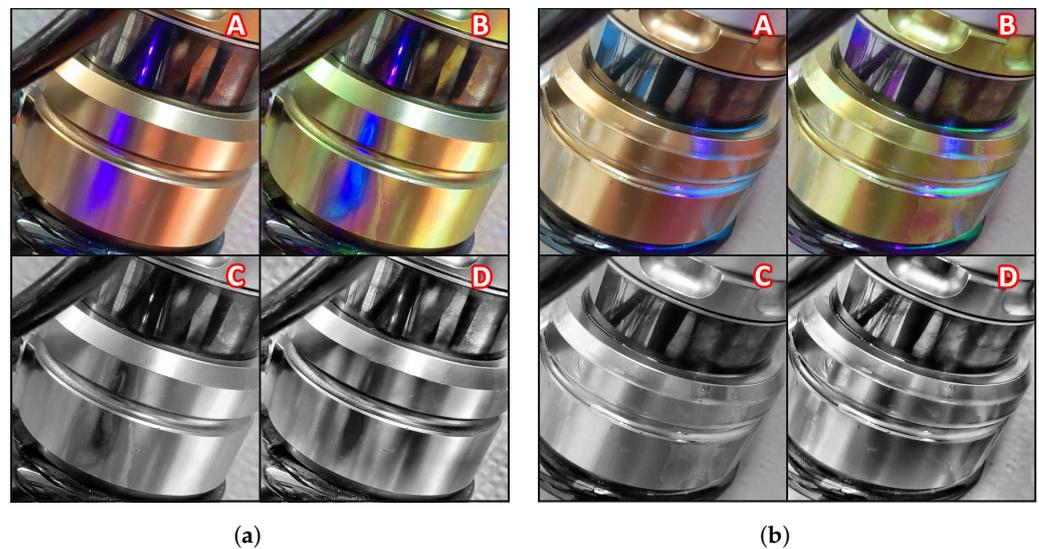


Figure 2. (a) Normally functioning AirTender processed in four different ways: original image (A), CLAHE filter (B), gray + CLAHE (C) and CLAHE + gray (D). (b) Faulty AirTender: original image (A), CLAHE filter (B), gray + CLAHE (C) and CLAHE + gray (D). Oil fluorescence is clearly visible on the core of AirTender.

3.4.3. Image Normalization

For most images, the pixel values are usually given in terms of integers in the range 0 to 255. On the other hand, neural networks process input data using small weights and biases. It is often the case that neural networks work better when the pixel values are comparable with weights and biases. This, in general, helps the algorithm converge faster.

Thus, a step often taken in computer vision is to normalize the input data, rescaling the pixel values to lie in the range 0 to 1. This is readily done by dividing each pixel of each color channel by 255.

3.4.4. Data Augmentation

Data augmentation is a useful technique in Machine Learning used to increase the size of the training set. This is particularly useful when creating a fully representative dataset. It turns out to be a challenging task. In addition, CNNs by construction leverage translation invariance and scale separation (as explained above), but do not explicitly take into account different kinds of transformations that do not spoil information about the image. Examples of such transformations are rotations, dilations, different conditions of lighting, and so on. Thus, data augmentation can be a useful tool to manually include such a transformation without changing the network architecture.

For this work, we relied heavily on data augmentation for both tasks, AirTender localization and classification. In more detail, we considered random rotation of input images, random zoom, shifts and flips both vertically and horizontally as well as random change of brightness conditions. The Yolov5 architecture also contemplates a mosaic method for data augmentation, where random images are put together (mosaic) and, therefore, more objects of the same type can appear in the same input image.

3.5. Model Architectures

In this section, we summarize the model architectures we employed for detection and classification, respectively. We begin with a brief summary of the Yolov5 architecture and the setup we used for the training. Then, we move on to discussing our architecture for AirTender classification.

3.5.1. Model Architecture for AirTender Detection: Yolov5

For object detection, we relied on the Yolov5 architecture [39]. Yolov5 is a family of object detection architectures pretrained on the COCO dataset and based on the Pytorch framework [59]. Full details on the model architecture as well as tutorials on how to train the network on new datasets and deploy the model can be found on GitHub [39].

Within the Yolov5 family, there are 10 different architectures, pretrained on the COCO dataset, which differ in size and performance. As explained on the GitHub page, a Yolov5 model can be exported to different formats, including TensorFlow SavedModel, TensorFlow Lite, and TensorFlow.js (and many others). This is a useful feature if one wanted to deploy a model to a desktop or a web app.

For our algorithm, we used the Yolov5s architecture, a light version with less parameters than its siblings Yolov5m, Yolov5l or Yolov5x but with a shorter inference time. This choice was mainly dictated by the fact that we want to deploy the trained model to single-board computers such as OrangePi or RaspberryPi, which should be able to detect AirTender in real time.

3.5.2. CNN Model for Binary Classification: OilNet40

In this paper, we use our own deep CNN called OilNet40. The network was developed using Keras [60], the high-level API of TensorFlow 2 [61]. The purpose of OilNet40 is to perform binary classification on images of AirTender. In other words, it should distinguish between leak and non-leak images of AirTender. Such a network was designed following standard rules of building CNNs, see, for example, [28,29]. The reason for designing a specific CNN, rather than using some well-established model and leveraging transfer learning, lies in the ability of its customization: the setup of very complex networks performing convoluted and diverse tasks is less prone to modifications and is less malleable to our purposes.

The structure of the network is shown in Figure 3. Input images are first resized to $240 \times 240 \times 3$ and then passed through a stack of convolutional and dense layers.

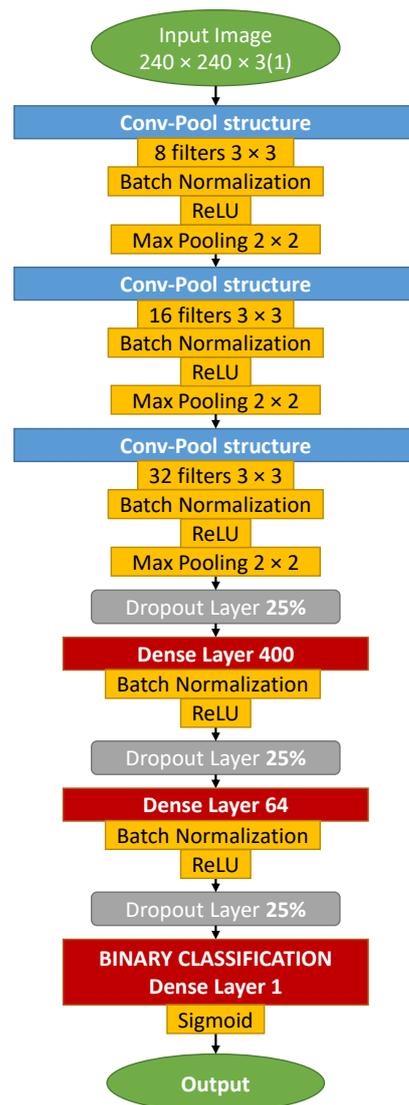


Figure 3. OilNet40 CNN diagram architecture.

In particular, we have three stacks of convolutional layers, with 8, 16, and 32 3×3 filters, respectively. Convolutional layers are all followed by batch-normalization [62] and ReLU activation function [34]. Here, batch normalization is used after the convolution. The ReLU activation function introduces nonlinearity to the network and is one of the most commonly used activation functions. Convolution is always performed with 3×3 filters. Some authors consider also filters with different dimensions. In general, the dimension of filters is seen decreasing along the neural network. It is not uncommon to use 5×5 or even 7×7 sized filters, for the first convolutional layers for large input images. Finally, 2×2 max-pooling is considered after each convolutional layer.

After convolutions and max-pooling, we included a stack of two dense layers. Indeed, little experimentation showed that one stack of dense layers is not enough to process the features extracted with convolutional filters, resulting in worse classification. On the other hand, three stacks of neurons introduce too many parameters, at the risk of overfitting. We introduced a dropout (25%) before each dense layer. Dropout is a regularization technique often used to reduce overfitting. It selects randomly neurons to be removed during each training, reducing in turn the number of parameters of the network. The first dense layer has 400 neurons, while the second one has 64 neurons. Each dense layer is followed by batch normalization and a ReLU activation function. At the top of the neural network, we

have a single neuron where a sigmoid gives back the prediction of the network, i.e., leak or non-leak Airtender with 0.5 as threshold.

For the training, we used the Nadam optimizer, i.e., Adam with the Nesterov trick [63]. In addition, we tested different CNN architectures, changing each time the number of filters, number of neurons in the dense layers and learning rate of the optimizer. All this was done with the help of Keras Tuner [64] Random Search. In particular, we set up a grid search in order to tune the following parameters: number of neurons in the dense layers and the learning rate. Finally, we selected the model performing best on the validation set. Of all the architectures, we explored that the best was the 40th network in the list, hence the name OilNet40.

4. Results

In this section, we collect the results of our classification algorithm. As explained before, we trained two different neural networks, the Yolov5s for the detection of AirTender and the OilNet40 network for the actual classification.

The classification is performed on four different datasets, each corresponding to a different preprocessing technique. In particular, the first dataset is the original dataset, with no preprocessing. In the second dataset, images are processed with a CLAHE filter, while the third and fourth correspond to grayscale images obtained by conversion to grayscale before and after applying the CLAHE filter, respectively (see Section 3.4.2). Results for the training of OilNet40 on all four datasets are presented below.

4.1. AirTender Detection

We begin by summarizing the results for the detection of AirTender. To this end, we used the Yolov5s architecture [39]. As anticipated, we trained the small Yolov5 architecture because the model eventually is to be deployed to our mobile system, and a network with a relatively small number of parameters is easier to export and use, besides offering faster performance.

As mentioned above, the dataset comprises 1634 images, split into a training, a validation, and a test set. The dataset we work with is not made of a very large number of images. This is essentially due to the many technical difficulties that arise when producing one. Nonetheless, we tried to make it as diverse as possible, using images of AirTender in different conditions of lighting, orientation, positioning, etc. In addition, we relied heavily on data augmentation, as offered by the Yolov5 system [39]. This helped us work with an even more diverse dataset. In particular, random rotation in the range -90 to 90 degrees along with vertical and horizontal flipping allows to have in the dataset images of AirTender framed from all different angles. This is particularly useful when, in the final setup, one does not have much control on the relative orientation of the recording camera and AirTender. Moreover, the mosaic data augmentation feature allows to have images of AirTender at different distances from the recording camera.

In Figure 4, we show the result of the training. In particular, we show the mean Average Precision mAP 0.5:0.95 against the number of epochs. It is used in many benchmark challenges such as Pascal, VOC or COCO, and corresponds to a number normalized between 0 and 1. Roughly, the closer it is to 1, the easier it is for the system to detect the AirTender at the correct position. See, e.g., [28] for details.

The actual training was performed on a PC equipped with a GPU Nvidia RTX 3060 with 12 GB of dedicated memory and 3584 CUDA cores. We see in Figure 4 that the model is learning throughout and achieves a mAP of about 85% only after 300 epochs.

Overall, we find that the Yolov5 architecture is rather easy to train and works reasonably well, at least in the case presented in this paper. Once AirTender is correctly localized, its status can be classified by a second network. Thus, we can now proceed to discussing the more interesting part of the algorithm, i.e., classification of leak or non-leak images of AirTender.

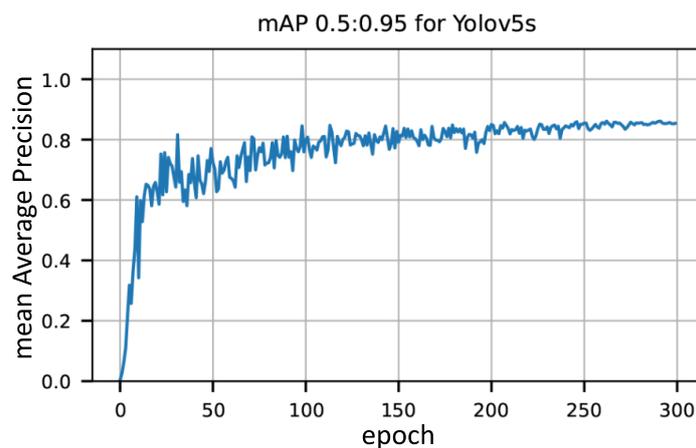


Figure 4. Mean Average Precision (mAP 0.5:0.95) for AirTender detection.

4.2. AirTender Classification

Classification is the second task of our algorithm. Once the AirTender is correctly located in the suspension system, the aim is to spot oil leakages, if any. To this purpose, as anticipated before, we set up a convolutional network (OilNet40) for classification.

The network should be fed the resulting image of the object detection. At a practical level, this is done by cutting out what is inside the bounding box of the object detection. The classifier, in turn, will then assign a class to the AirTender: Normal or Anomaly, according to the status of Airtender. See Figure 5.

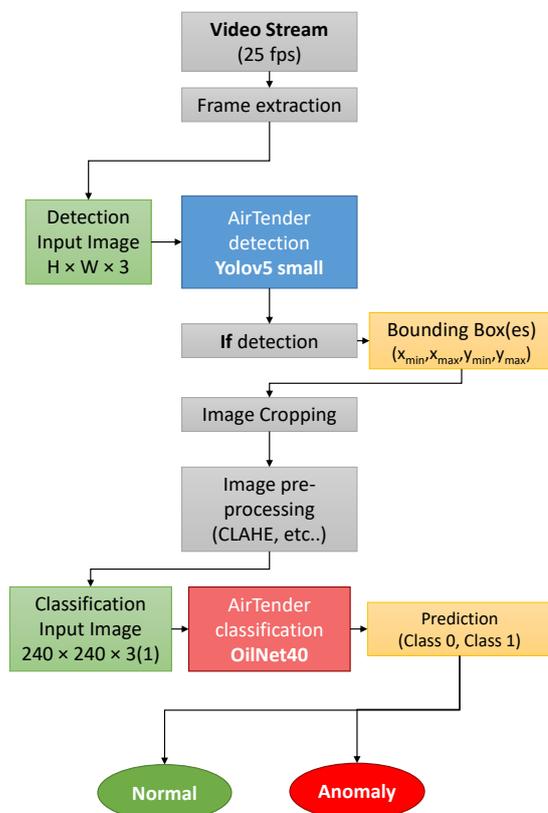


Figure 5. Summary of the key processes to spot oil leaks.

As mentioned above, we trained OilNet40 on four different versions of our original dataset. The results of the training are shown in Figure 6a,b. In each case, we plot the accuracy of prediction as well as Precision and Recall versus the number of epochs, for both training and validation. All metrics (Accuracy on the left plot and Precision and Recall on the right plot) are plotted on the y axis. Notice that we have always split the dataset in 50% leaks and 50% non-leaks, thus a randomly guessing algorithm is expected to give the correct answer 50% of the time, while a well-trained Neural Network is expected to give results with confidence well above that threshold.

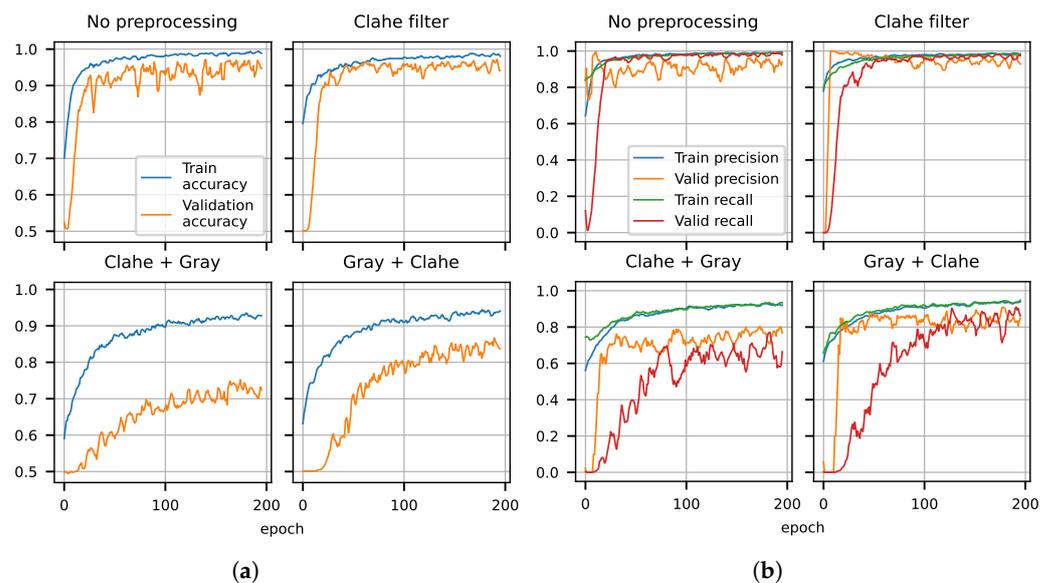


Figure 6. (a) Training and validation accuracy for OilNet40 trained on four different versions of the dataset. The model trained on the dataset preprocessed with CLAHE shows more stability and reaches the highest accuracy. (b) Precision and Recall for training and validation of OilNet40. The model performs best on color images. The model trained on the dataset filtered with CLAHE has the smallest number of false positive and negatives.

From the plots in Figure 6a,b, we see that all metrics improve with the number of epochs, and the better performing algorithm is that trained on images where the CLAHE filter was applied (see Section 3.4.2). The Network trained on grayscale images does not seem to perform as well, reaching a level of accuracy of about 80% in both cases (CLAHE + Gray and Gray + CLAHE). The grayscale datasets were introduced with the goal of having a more universal algorithm, working independently of the color of AirTender and emitted fluorescence. The main reason for which we are not able to train an algorithm that generalizes well is presumably due to a small dataset. A larger and more representative dataset is expected to lead to a generalizing algorithm also for the grayscale images of AirTender.

Besides training and validation, we tested our network on a separate test set, for which we plotted confusion matrices. See Figure 7. As mentioned above, the test data are never introduced when training the OilNet40 and correspond to images taken independently of the training and validation sets and at a different time of the day. On the vertical axis, we have the actual labels, normal or anomaly (i.e., non-leak or leak image). We can see that the models performing best are those trained on images with no preprocessing or on images where CLAHE was applied, as expected. In particular, in the first case (no preprocessing), we have zero false positives but 3 false negatives, while in the second case (CLAHE), we have one false positive and one false negative. In the case of grayscale images, OilNet40 trained on CLAHE + gray dataset seems to perform better than the same model trained on the gray + CLAHE dataset, even though the training suggested the opposite. In the former case, OilNet40 achieves Precision 0.96 and Recall 0.97, while in the latter, it achieves Precision 1 and Recall 0.77, finding more normal configurations than the actual ones.

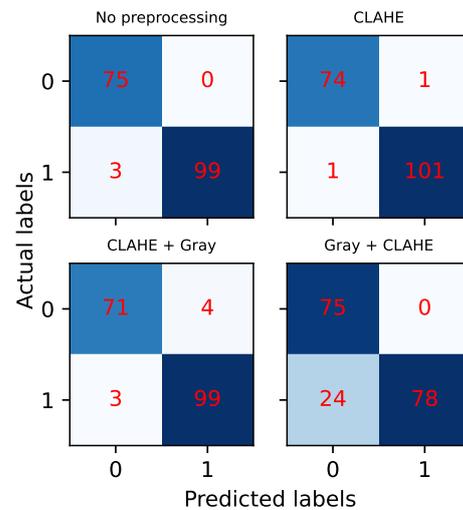


Figure 7. Confusion Matrices for OilNet40 tested on four different versions of the test dataset. The outcomes confirm the results of the training for the color images. In the case of grayscale images, OilNet40 trained on CLAHE + gray dataset seems to perform better than gray + CLAHE even though the training suggested the opposite.

It is worth noting at this point that, differently from other architectures, CNNs are not quite black boxes as one might think. In fact, in CNNs, it is possible to visualize the feature maps, i.e., what a CNN sees after each convolution. This step turns out to be useful to understand whether the filters highlight the regions of interests for a classifier such as ours. This gives a view into how an input is decomposed in the classification process.

In Figure 8, we show every channel (32 filters in total) of the third layer activation on both a normally functioning AirTender (above) and a faulty AirTender (below). In Figure 8, image below, red contours highlight the feature maps related to the filters which mainly react to the presence of spilled oil on the surface of AirTender core. The same filters are not activated on a similar image of AirTender with no spilled oil, as shown in the upper image.

At this point, it could be nice to test the robustness of our results, making the two networks (Yolo and OilNet40) work sequentially, one after the other. Let us see how this is the case.

4.3. Robustness of the Results

In order to check the robustness of our results, we tested the complete algorithm, illustrated in Figure 5, on a live video stream. We installed the TF Lite version of the trained networks on the OrangePi board, to check its actual performance on such a pipeline. We set up a python script following the flux diagram in Figure 5, to have a working algorithm which first detects AirTender from a live camera and then classifies the image as leak/non-leak.

To record the video, the AirTender was mounted on a pedestal together with the whole suspension system (spring, damper, etc.) and placed indoor in varying light conditions. To test the classification of anomalous functioning, the AirTender was stained with oil+dye and enlightened with UV light. A one-minute video stream was fed to the OrangePi.

We found that it can process video streams with a frequency of 0.25 fps. To check the anomaly-detection performances, we recorded a video of both leak/non-leak events and manually labeled each frame, then fed the video-dataset to the OrangePi. The video-dataset consists of 1517 frames, 651 normal and 866 anomalous. It turned out that the embedded system can classify images correctly (accuracy) 94% of the times. We can consider this a good result for a preliminary study. The same test should be run, in the future, on a real motorcycle.

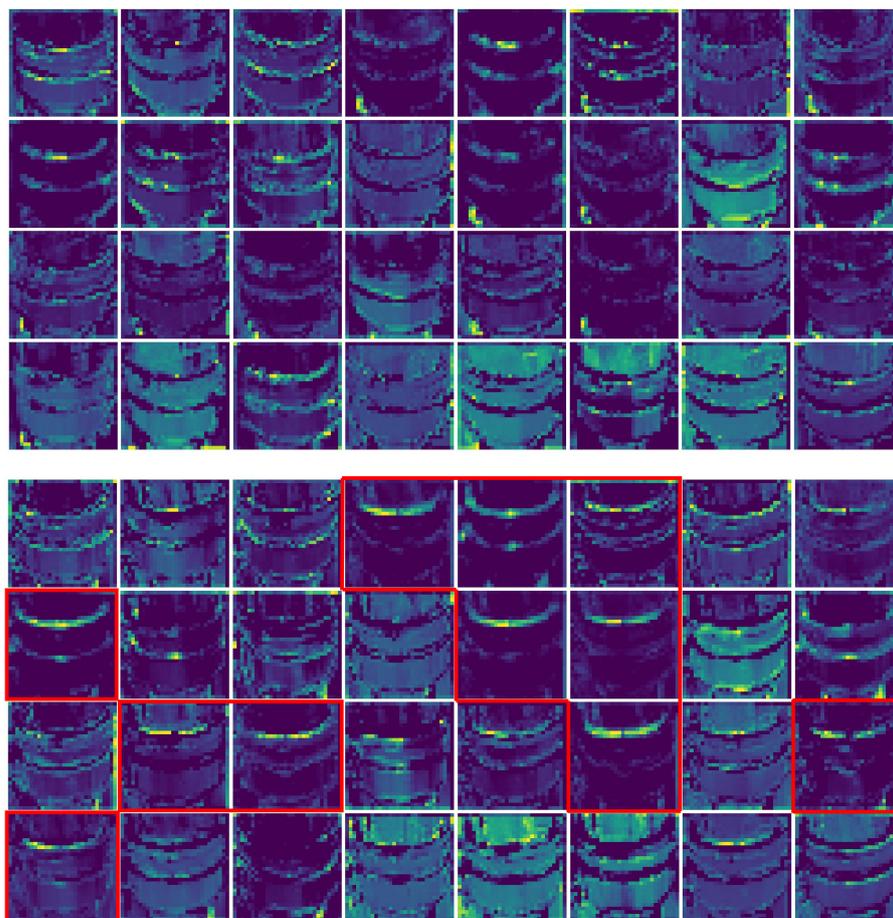


Figure 8. The 32 channels of the third convolutional layer activation on both a normally functioning AirTender (**above**) and a faulty AirTender (**below**). In the image below, red contours highlight the feature maps related to the filters which mainly react to the presence of spilled oil on the surface of AirTender core. The same filters are not activated on a similar image of AirTender with no spilled oil, as shown in the upper image.

5. Concluding Remarks

Our work illustrates a new and efficient way of using on-board devices to detect fault events of a modern suspension system. In particular, we relied on deep learning techniques to detect and classify the status of a special type of motorbike suspension system. The procedure is fast, flexible and easy to improve.

The algorithm presented in this paper is shown to be efficient on images and video-frames of AirTender that make up our dataset, and consists of two steps. We first focus on the detection of AirTender in the motorbike suspension system. This step turns out to be necessary because a camera placed next to the chassis system of a generic motorbike is not expected to frame AirTender only. It will, in general, frame AirTender and its surrounding environment which is different from motorbike to motorbike. The second step of our algorithm consists in classifying AirTender as normally functioning or not: a binary classifier is fed with the results of the detection, i.e., the core of AirTender.

As explained in the text, a broken AirTender is very likely to spill oil. In order to make oil leaks more evident, we set up a small circuit where UV LEDs are pointed at the core of AirTender and are supposed to make the oil fluoresce. All this is amplified by diluting the mechanical oil with fluorescent dye.

Our work can be easily extended by further enlarging the dataset in order to represent the diversity of leaks that can occur when AirTender breaks. This would allow to train an algorithm that better generalizes to real cases.

The accuracy of our algorithm depends on a number of factors. First, we considered images of AirTender with a decent resolution. An algorithm operating on very low quality images or videos of Airtender can have poor performance. In addition, if the oil leak is not visible enough, or if the emitted fluorescence is not of sufficient intensity, the algorithm can misclassify the status of AirTender. To this end, it is necessary to have the UV LEDs directed towards the leaking area. We hope to improve our algorithm so to include more general setups in future studies.

Finally, we would like to make a few comments on other deep learning techniques that could be used for anomaly detection. To spot oil leaks, we relied on binary classification using a home-made Convolutional Neural Network. It would be interesting to explore other architectures as well. In particular, Convolutional Autoencoders could be employed to spot oil leaks for conditions “far from the normality”. In addition, oil leaks could be detected by means of semantic segmentation. For example, a network such as the U-Net [65], introduced for biomedical image segmentation, is sometimes also used to spot oil leaks on metallic surfaces [56]. We will attempt to explore all these different architectures in future research.

Author Contributions: Conceptualization, F.B., S.S. and A.G.; Data curation, F.B. and S.S.; Formal analysis, F.B. and S.S.; Funding acquisition, G.B.; Investigation, F.B. and S.S.; Methodology, F.B., S.S. and A.G.; Project administration, A.G., G.B. and M.M.; Resources, G.B.; Software, F.B., S.S. and A.M.; Supervision, A.G. and M.M.; Validation, F.B. and S.S.; Writing—original draft, F.B. and S.S.; Writing—review & editing, F.B., S.S., A.M., M.P., L.M., A.G., G.B. and M.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by INVITALIA bandi BREVETTI+, grant number BRE0000396, as part of the INNOSUSP project.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author. The data are not publicly available due to non disclosure agreement.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kothamasu, R.; Huang, S.H.; VerDuin, W.H. System health monitoring and prognostics—A review of current paradigms and practices. *Int. J. Adv. Manuf. Technol.* **2006**, *28*, 1012–1024. [\[CrossRef\]](#)
2. Siegel, J.E.; Erb, D.C.; Sarma, S.E. A survey of the connected vehicle landscape—Architectures, enabling technologies, applications, and development areas. *IEEE Trans. Intell. Transp. Syst.* **2017**, *19*, 2391–2406. [\[CrossRef\]](#)
3. Delvecchio, S.; Bonfiglio, P.; Pompoli, F. Vibro-acoustic condition monitoring of Internal Combustion Engines: A critical review of existing techniques. *Mech. Syst. Signal Process.* **2018**, *99*, 661–683. [\[CrossRef\]](#)
4. Siegel, J.E.; Coda, U. Surveying off-board and extra-vehicular monitoring and progress towards pervasive diagnostics. *arXiv* **2020**, arXiv:2007.03759.
5. Zhao, R.; Yan, R.; Chen, Z.; Mao, K.; Wang, P.; Gao, R.X. Deep learning and its applications to machine health monitoring. *Mech. Syst. Signal Process.* **2019**, *115*, 213–237. [\[CrossRef\]](#)
6. Zhao, G.; Zhang, G.; Ge, Q.; Liu, X. Research advances in fault diagnosis and prognostic based on deep learning. In Proceedings of the 2016 Prognostics and System Health Management Conference (PHM-Chengdu), Chengdu, China, 19–21 October 2016; pp. 1–6.
7. Chase, C.; Lyra, G.; Green, M. Real time monitoring—the key to effective oil spill prevention and response. In Proceedings of the Rio Oil & Gas Expo and Conference, Rio de Janeiro, Brazil, 13–16 September 2010; pp. 1–10.
8. Camagni, P.; Colombo, G.; Koechler, C.; Pedrini, A.; Omenetto, N.; Rossi, G. Diagnostics of oil pollution by laser-induced fluorescence. *IEEE Trans. Geosci. Remote Sens.* **1988**, *26*, 22–26. [\[CrossRef\]](#)
9. Fingas, M.; Brown, C. Review of oil spill remote sensing. *Mar. Pollut. Bull.* **2014**, *83*, 9–23. [\[CrossRef\]](#) [\[PubMed\]](#)
10. Sato, T.; Suzuki, Y.; Kashiwagi, H.; Nanjo, M.; Kakui, Y. A method for remote detection of oil spills using laser-excited Raman backscattering and backscattered fluorescence. *IEEE J. Ocean. Eng.* **1978**, *3*, 1–4. [\[CrossRef\]](#)
11. Ferreira, C.; Ventura, P.; Morais, R.; Valente, A.L.; Neves, C.; Reis, M.C. Sensing methodologies to determine automotive damper condition under vehicle normal operation. *Sens. Actuators A Phys.* **2009**, *156*, 237–244. [\[CrossRef\]](#)

12. Hernandez-Alcantara, D.; Morales-Menendez, R.; Amezcuita-Brooks, L. Fault detection for automotive shock absorber. In *Journal of Physics: Conference Series*; IOP Publishing: Bristol, UK, 2015; Volume 659, p. 012037.
13. Hernandez-Alcantara, D.; Amezcuita-Brooks, L.; Vivas-Lopez, C.; Morales-Menendez, R.; Ramirez-Mendoza, R. Fault detection for automotive semi-active dampers. In Proceedings of the 2013 Conference on Control and Fault-Tolerant Systems (SysTol), Nice, France, 9–11 October 2013; pp. 625–630.
14. Alcantara, D.H.; Morales-Menendez, R.; Amezcuita-Brooks, L. Fault diagnosis for an automotive suspension using particle filters. In Proceedings of the 2016 European Control Conference (ECC), Aalborg, Denmark, 29 June–1 July 2016; pp. 1898–1903.
15. Xia, M.; Li, T.; Xu, L.; Liu, L.; De Silva, C.W. Fault diagnosis for rotating machinery using multiple sensors and convolutional neural networks. *IEEE/ASME Trans. Mechatron.* **2017**, *23*, 101–110. [[CrossRef](#)]
16. Eren, L.; Ince, T.; Kiranyaz, S. A generic intelligent bearing fault diagnosis system using compact adaptive 1D CNN classifier. *J. Signal Process. Syst.* **2019**, *91*, 179–189. [[CrossRef](#)]
17. Zilong, Z.; Wei, Q. Intelligent fault diagnosis of rolling bearing using one-dimensional multi-scale deep convolutional neural network based health state classification. In Proceedings of the 2018 IEEE 15th International conference on networking, sensing and control (ICNSC), Zhuhai, China, 27–29 March 2018; pp. 1–6.
18. Janssens, O.; Slavkovikj, V.; Vervisch, B.; Stockman, K.; Loccupier, M.; Verstockt, S.; Van de Walle, R.; Van Hoecke, S. Convolutional neural network based fault detection for rotating machinery. *J. Sound Vib.* **2016**, *377*, 331–345. [[CrossRef](#)]
19. Wen, L.; Li, X.; Gao, L.; Zhang, Y. A new convolutional neural network-based data-driven fault diagnosis method. *IEEE Trans. Ind. Electron.* **2017**, *65*, 5990–5998. [[CrossRef](#)]
20. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [[CrossRef](#)]
21. Zhang, W.; Peng, G.; Li, C. Bearings fault diagnosis based on convolutional neural networks with 2-D representation of vibration signals as input. In Proceedings of the MATEC Web of Conferences, Seoul, Korea, 22–25 August 2017; EDP Sciences: Les Ulis, France, 2017; Volume 95, p. 13001.
22. Liao, Y.; Zeng, X.; Li, W. Wavelet transform based convolutional neural network for gearbox fault classification. In Proceedings of the 2017 Prognostics and System Health Management Conference (PHM-Harbin), Harbin, China, 9–12 July 2017; pp. 1–6.
23. Verstraete, D.; Ferrada, A.; Droguett, E.L.; Meruane, V.; Modarres, M. Deep learning enabled fault diagnosis using time-frequency image analysis of rolling element bearings. *Shock Vib.* **2017**, *2017*. [[CrossRef](#)]
24. Zhang, W.; Zhang, F.; Chen, W.; Jiang, Y.; Song, D. Fault state recognition of rolling bearing based fully convolutional network. *Comput. Sci. Eng.* **2018**, *21*, 55–63. [[CrossRef](#)]
25. Ding, X.; He, Q. Energy-fluctuated multiscale feature learning with deep convnet for intelligent spindle bearing fault diagnosis. *IEEE Trans. Instrum. Meas.* **2017**, *66*, 1926–1935. [[CrossRef](#)]
26. Zehelein, T.; Hemmert-Pottmann, T.; Lienkamp, M. Diagnosing automotive damper defects using convolutional neural networks and electronic stability control sensor signals. *J. Sens. Actuator Netw.* **2020**, *9*, 8. [[CrossRef](#)]
27. LeCun, Y.; Boser, B.; Denker, J.; Henderson, D.; Howard, R.; Hubbard, W.; Jackel, L. Handwritten digit recognition with a back-propagation network. *Adv. Neural Inf. Process. Syst.* **1989**, *2*, 296–303.
28. Géron, A. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2019.
29. Chollet, F. *Deep Learning with Python*; Simon and Schuster: New York, NY, USA, 2021.
30. Mehta, P.; Bukov, M.; Wang, C.H.; Day, A.G.; Richardson, C.; Fisher, C.K.; Schwab, D.J. A high-bias, low-variance introduction to machine learning for physicists. *Phys. Rep.* **2019**, *810*, 1–124. [[CrossRef](#)]
31. Bronstein, M.M.; Bruna, J.; LeCun, Y.; Szlam, A.; Vandergheynst, P. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Process. Mag.* **2017**, *34*, 18–42. [[CrossRef](#)]
32. Bronstein, M.M.; Bruna, J.; Cohen, T.; Velicković, P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv* **2021**, arXiv:2104.13478.
33. Cohen, T.S. Equivariant Convolutional Networks. Ph.D. Thesis, Faculty of Science (FNWI), Amsterdam, The Netherlands, 2021.
34. Nair, V.; Hinton, G.E. Rectified linear units improve restricted boltzmann machines. In Proceedings of the International Conference on Machine Learning (ICML), Haifa, Israel, 21–24 June 2010.
35. Nagi, J.; Ducatelle, F.; Di Caro, G.A.; Cireşan, D.; Meier, U.; Giusti, A.; Nagi, F.; Schmidhuber, J.; Gambardella, L.M. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In Proceedings of the 2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), Kuala Lumpur, Malaysia, 16–18 November 2011; pp. 342–347.
36. Lin, M.; Chen, Q.; Yan, S. Network in network. *arXiv* **2013**, arXiv:1312.4400.
37. Abu-Mostafa, Y.S.; Magdon-Ismail, M.; Lin, H.T. *Learning from Data*; AMLBook: New York, NY, USA, 2012; Volume 4.
38. Long, J.; Shelhamer, E.; Darrell, T. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 3431–3440.
39. Jocher, G.; Nishimura, K.; Mineeva, T.; Vilariño, R. yolov5. Code Repository. 2020. Available online: <https://github.com/ultralytics/yolov5> (accessed on 11 September 2022).
40. Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis. (IJCV)* **2015**, *115*, 211–252. doi: 10.1007/s11263-015-0816-y. [[CrossRef](#)]

41. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 84–90. [[CrossRef](#)]
42. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 1–9.
43. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
44. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
45. Szegedy, C.; Ioffe, S.; Vanhoucke, V.; Alemi, A.A. Inception-v4, inception-resnet and the impact of residual connections on learning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017.
46. Chollet, F. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 1251–1258.
47. Zeng, X.; Ouyang, W.; Yan, J.; Li, H.; Xiao, T.; Wang, K.; Liu, Y.; Zhou, Y.; Yang, B.; Wang, Z.; et al. Crafting gbd-net for object detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *40*, 2109–2123. [[CrossRef](#)]
48. Hu, J.; Shen, L.; Sun, G. Squeeze-and-excitation networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 7132–7141.
49. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 779–788.
50. Redmon, J.; Farhadi, A. YOLO9000: Better, faster, stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 7263–7271.
51. Redmon, J.; Farhadi, A. Yolov3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.
52. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. Ssd: Single shot multibox detector. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 21–37.
53. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *Adv. Neural Inf. Process. Syst.* **2015**, *28*, 1137–1149. [[CrossRef](#)]
54. He, K.; Gkioxari, G.; Dollár, P.; Girshick, R. Mask r-cnn. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017; pp. 2961–2969.
55. Zhao, Z.Q.; Zheng, P.; Xu, S.t.; Wu, X. Object detection with deep learning: A review. *IEEE Trans. Neural Netw. Learn. Syst.* **2019**, *30*, 3212–3232. [[CrossRef](#)]
56. Lu, L.; Ichimura, S.; Yamagishi, A.; Rokunohe, T. Oil film detection under solar irradiation and image processing. *IEEE Sens. J.* **2019**, *20*, 3070–3077. [[CrossRef](#)]
57. Tzutalin. *Labellmg, Free Software*; MIT License: West Hollywood, CA, USA, 2015.
58. Bradski, G. The OpenCV Library. *Dr. Dobb's J. Softw. Tools Prof. Program.* **2000**, *25*, 120–123.
59. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*; Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2019; pp. 8024–8035.
60. Chollet, F. Keras. 2015. Available online: <https://keras.io> (accessed on 11 September 2022).
61. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Available online: <https://tensorflow.org> (accessed on 11 September 2022).
62. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the International Conference on Machine Learning (PMLR), Lille, France, 7–9 July 2015; pp. 448–456.
63. Dozat, T. Incorporating Nesterov Momentum into Adam. 2016. Available online: <https://openreview.net/pdf?id=OM0jvwB8jIp57ZJjtNEZ> (accessed on 11 September 2022).
64. O'Malley, T.; Bursztein, E.; Long, J.; Chollet, F.; Jin, H.; Invernizzi, L. KerasTuner. 2019. Available online: <https://github.com/keras-team/keras-tuner> (accessed on 11 September 2022).
65. Ronneberger, O.; Fischer, P.; Brox, T. U-net: Convolutional networks for biomedical image segmentation. In Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention, Munich, Germany, 5–9 October 2015; Springer: Berlin/Heidelberg, Germany, 2015; pp. 234–241.