Master Degree in Computer Engineering

Master Thesis

3D Printing Anomaly Detection: Implementation of a ML system using YOLOv5 and EfficientNet-Lite

Supervisor

Prof. Tania CERQUITELLI

Candidate

Elisa CENEDESE

Company Tutor

Giorgia FORTUNA (ML Reply)

Davide VENA (ML Reply)

Academic Year 2021/2022

December 2022

"To all those who do not have the strength or the possibility to change their destiny and dreams. Be resilient and have the courage to adjust your path."

Summary

In recent decades there has been an increasing shift to prototyping objects digitally using Computer Aided Design (CAD). While this software is great for rapid prototyping and is adopted in many contexts, turning the digital object into a real object has been an open problem for several years. Before the advent of 3D printers, the main technique used was Subtractive Manufacturing (SM), which consists of modeling a 3D object starting from a large block and removing material until it reaches the final shape. While this technique works, it is highly inefficient, for several reasons. Lately, the situation has changed with 3D printers becoming mainstream. Modern 3D printers use a technique, called Additive Manufacturing (AM), to create a physical object, starting from the digital counterpart. This process works by dividing the digital design into multiple layers and printing the physical object layer by layer, extruding new material from time to time (usually ABS or PLA). The whole procedure typically takes a long time and is not uncommon for the printing process to give an unsatisfactory result. Specifically, it may happen that, for various reasons, the final printed product contains one or more anomalies. When this happens, depending on the type of anomaly that has occurred and the context of use of the final product, the object could be discarded, leading to a considerable loss in terms of time and cost. Therefore, the faster the anomaly is detected, the better.

Carrying out monitoring of a 3D printing process is a very tedious and time-consuming task, which makes manual handling in large contexts, with multiple machines, impractical. Starting from these considerations, the aim of the thesis is to identify an innovative solution to perform the anomaly detection task for a 3D printing process, with the aim of identifying the occurrences of printing defects in near real time. The proposed solution receives a real-time video stream of the 3D printing process from a webcam and uses Computer Vision (CV) techniques for image analysis. The video stream is received as input to the Anomaly Detection model using the Real Time Stream Protocol (RTSP) and each frame is analyzed individually using a specific pipeline: first, a detection task is performed to identify the nozzle of the 3D printer and the area underneath, then the detected area is analyzed using a classification task, which results in the presence or absence of

anomaly for that frame. Both steps of the pipeline have been implemented using Machine Learning libraries specifically adapted and trained for this use case. The Detection of the 3D printer nozzle is performed using You Only Look Once v5 (YOLOv5), which is a Machine Learning framework that employs convolutional neural networks (CNNs) to provide real-time object detection. The output of this first step is the identification of the Area Below the Nozzle (ABN) containing the portion of the 3D object currently being printed. The ABN is then passed to the second step which consists of a classification task implemented using TensorFlow-Lite library and EfficientNet-Lite as the backbone network. The output of this step is a binary classification label which holds the information 'Anomaly' or 'No Anomaly' for the frame analyzed. The obtained anomaly detection model can find several types of 3D printing defects like First Layers issues, Stringing, Layer Shifting, Under Extrusion, Over Extrusion and Detach. Since the thesis project has been performed in collaboration with Machine Learning Reply and a Dutch multinational lighting corporation, both steps have been trained using data coming from the lighting company production process of 3D printed lamps. Furthermore, the system has been optimized to work in tiny arm edge-devices like the Raspberry Pi.

In conclusion, the proposed solution can perform near-real time anomaly detection of a RTSP video stream showing a 3D printing process. With this strategy, if an anomaly is detected the printing process can be stopped before completion, thus resulting in a save of printing time and material for the company.

Acknowledgements

I would like to thank my academic supervisor, Tania Cerquitelli, and my company tutor, Giorgia Fortuna, for the opportunity and support provided to me during this experience. I would also like to express my gratitude to Davide Vena who guided me during these months in the development of the thesis.

My final words are dedicated to my family, who always encouraged me to study, and to Fabio, who helped me believe in myself and supported me emotionally during this journey.

Table of Contents

Chapter 1

Introduction

While Computer Aided Design (CAD) software has been used for decades for rapid prototyping, turning the resulting digital object into a real one has always been a challenge. However, this has changed with the advent of 3D Printing technology, which is still rapidly evolving today. The first commercialized 3D Printing process was released by Charles Hull in the 1980s [1], whose main functioning consisted in building up the object layer by layer, using a UV light to cure and bond a photopolymer resin. Nowadays, thanks to its versatility and continuous innovation, 3D printing technology is used by companies to improve manufacturing productivity and it is applied in many fields such as agriculture, healthcare, automotive industries and many others, to build any kind of open source design [2].

Modern 3D Printers use a technique called Additive Manufacturing (AM) in order to create a physical object, starting from the digital counterpart. In 2021, the International Organization for Standardization (ISO) created the standard ISO/ASTM 52900, containing terms definition in the field of AM. According to [3], AM can be defined as:

> *"[The] process of joining materials to make parts from 3D model data, usually layer upon layer, as opposed to subtractive manufacturing and formative manufacturing methodologies."*

This means that, differently from other technologies such as Subtractive Manufacturing (SM) (see Chapter 2), 3D printing proceeds, starting from the CAD drawing, by dividing the digital design into several layers and printing the physical object layer by layer, depositing new material each time. Some of the advantages of 3D Printing, with respect to traditional manufacturing methods include: the possibility to create complex designs; the ability to create objects within hours (depending on the object design and complexity), together with the possibility to obtain fast design thanks to CAD files; the little wastage of material, compared to other methods such as SM. Despite the advantages mentioned above, overall

the 3D Printing process requires a certain amount of time (which varies depending on the design complexity), during which problems may occur. Specifically, it can happen due to several factors, that the object is printed with one or more defects (or anomalies) in it (see Section 2.2). Depending on the context of use of the object and the severity of the anomalies it contains, the final product could be rejected, resulting in a loss in terms of production time and printed material. If an irremediable defect occurs in the first printing layers and it is not identified as soon as possible by the 3D machine operator, waste increases as the process continues unnecessarily. Therefore, to reduce the damage it is important to carry out continuous monitoring of the process of print and to act as soon as possible. However, this manual monitoring is a very laborious and time consuming procedure, as well as being unfeasible in large scale settings, with many machines.

In order to alleviate the problems described above, deriving from the introduction of defects during the 3D Printing process, the thesis focuses on creating an automated system for detecting defects using Machine Learning (ML) techniques, with the aim to provide, as main feedback to the machine operator, a textual warning as soon as the introduction of an anomaly in the object being printed happens. All the thesis work has been done in partnership with Machine Learning Reply [4] and a Dutch multinational lighting corporation. ML Reply is the Reply group company specialized in ML & AI solutions and mainly devoted to enhance existing processes or introducing new AI based applications, supporting its customers in a end-to-end development. The project developed, for the 3D Printing Anomaly Detection, falls under the scope of the European Institute of Innovation and Technology (EIT) Digital [5] program, whose mission is to increase competitiveness in the digital sector in Europe, valuing the cooperation between leading business organisations, education and research institutes, as well as promoting digital innovation and entrepreneurship by creating suitable environments for the development of creative and innovative proposals.

Given the introduction above about the high level context and main objective of the thesis, the remaining chapters contain all the details on the thesis work and are divided as follows:

- **Chapter 2**: This chapter first covers the main concepts about 3D Printing technology, also known as Additive Manufacturing (AM), as well as briefly describing the main differences from Subtractive Manufacturing (SM) technology. Next, it introduces the concepts of *Print Quality* and *Quality Control*, in the context of 3D printing, as important aspects to take into consideration during the production of a 3D printed object, together with the concept of *Anomaly*, which is responsible for the print quality degradation. Lastly, the chapter presents a brief overview about the most frequent types of anomalies in 3D printing processes, together with some examples of Computer Vision

approaches available in literature, as regards detection and classification of anomalies in 3D Printing context.

- **Chapter 3**: This chapter presents and analyses the main concepts about Computer Vision, paying particular attention to the problem of *Visual Anomaly Detection* and its use case concerning *Defects Detection*, applied to the industrial sector. As regards *Visual Anomaly Detection*, both supervised and unsupervised approaches are highlighted, with more emphasis to the supervised setting which is the one chosen for the case of study under analysis. The last two sections named YOLO (see Sec. 3.3) and EfficientNet (see Sec. 3.4), contain the basic principles about the two Deep Learning strategies, which are the ones used to create the ML system for 3D Printing Anomaly Detection.

- **Chapter 4**: This chapter introduces the *3D Printing Anomaly Detection Model* proposed for the thesis work, explaining the high level pipeline implemented to perform anomaly detection at inference time. Each pipeline component is described in more detail in a dedicated section, as follows: the *Input* (see Sec. 4.2) refers to the video stream showing the 3D printing process; the *Detection task* (see Sec. 4.3) is devoted to the identification of the nozzle for each frame in the video stream; the *Classification task* (see Sec. 4.4) is devoted to the classification of the Area Below the Nozzle (ABN) into "anomaly" or "no anomaly"; the *Output* (see Sec. 4.5) gives the warning if an anomaly is found for the frame being analyzed.

- **Chapter 5**: This chapter describes from a technical point of view the implementation of the *3D Printing Anomaly Detection Model* and its application to the 3D printing processes of the lighting company. Specifically, the *Data Acquisition and Preparation* and the *DL Model Training* phases are presented both for *Nozzle Detection* (see Sec. 5.1) and *ABN Classification* (see Sec. 5.2), the outputs of which consist of two trained models to be used for inference on new data. Lastly, Section 5.3 explains, at a lower level with respect to Chapter 4, how *Nozzle Detection* and *ABN Classification* phases are applied in cascade to perform anomaly detection at inference time.

- **Chapter 6**: This chapter discusses the performance results obtained for the *3D Printing Anomaly Detection Model*. The evaluation is split in two phases: one related to *Nozzle Detection* (see Sec. 6.1), which uses the metrics most common in the context of object detection and one related to *ABN Classification* (see Sec. 6.2), which employs the metrics for binary classification, taking into account the imbalance problem among the class "anomaly" and "no anomaly". Section 6.3 includes some optimization techniques that are applied to optimize the model performance in terms of latency, allowing to obtain a

ML system able to accomplish the anomaly detection task in near real time, when a 30 FPS video stream is provided as input.

- **Chapter 7**: The last chapter contains a synthesis of all the thesis work, starting from the objective of the thesis and continuing with the results obtained for the solution, implemented with the aim to alleviate the problems highlighted in Chapter 2. Then, some of the limits of the proposed approach are presented, together with possible future additions to improve the actual ML system.

Chapter 2

3D printers

This chapter introduces the main concepts about 3D printers, together with some of the most common anomalies that can happen during the 3D printing process. The last part is devoted to a brief description of the applications that can be found in literature for automated detection and classification of defects, in the context of 3D printing.

2.1 Overview

Over the years, two kind of manufacturing technologies have drastically transformed the way products, parts and prototypes are produced: Additive Manufacturing (AM) and Subtractive Manufacturing (SM). Traditionally, Subtractive Manufacturing has been used in order to bring a 3D object to life. This method builds the final model starting from a large initial solid block and removing parts from time to time. SM process can be fulfilled by hand or, more commonly, by using a computer numeric control. Today, the most common SM process is Computer Numerical Control (CNC) machining. The other technology that has taken hold is Additive Manufacturing, which is widely known as *3D Printing*. Differently from SM, AM process consists in building an item, under automated control, by depositing repeated layers of solidifying material, until the final shape is assembled. The term AM reflects the fact that new material is added layer by layer and joined to create a 3D envelope. Although AM and SM use different approaches, they have an overlapping range of applications. In fact, they are often used side by side to optimize production, choosing the specific type of process based, for example, on production volume or stage of development. Below some of the differences between the two technologies are summarized:

- **Available materials**: SM uses a great range of materials such as stone, wood, glass, foam, plastics and (soft/hard) metals. AM instead works mainly with

plastics, plastic derivates, resins and some metals.

- **Design complexity**: AM is more suitable to build small intricated and complex shapes with respect to SM, which instead is used especially for manufacturing voluminous objects.
- **Accuracy**: SM reaches a higher accuracy level with respect to AM which often needs to perform surface machining after printing, for the parts that need high accuracy characteristics.
- **Surface finishing**: Due to the layering process, products obtained with AM may contain small pores and this can also lead to structural fragility in the final item. For this reason, they often need an additional phase of cleaning and finishing to obtain better product characteristics and aspect. On the other hand, SM allows to obtain more robust results with a better finish, including a great variety of surface finishing such as smooth, mottled and stepped.
- **Waste material**: AM produces less "waste material" (meaning the parts of material that are removed from the final product) with respect to SM, since it constructs the object from scratch.
- **Speed**: AM is often slower in production when dealing with large objects.

The thesis work focuses on the Additive Manufacturing process, since the objective is to perform anomaly detection in the context of a *3D Printing* process. A *Three-Dimensional (3D) Printer* is a type of material design machine, which

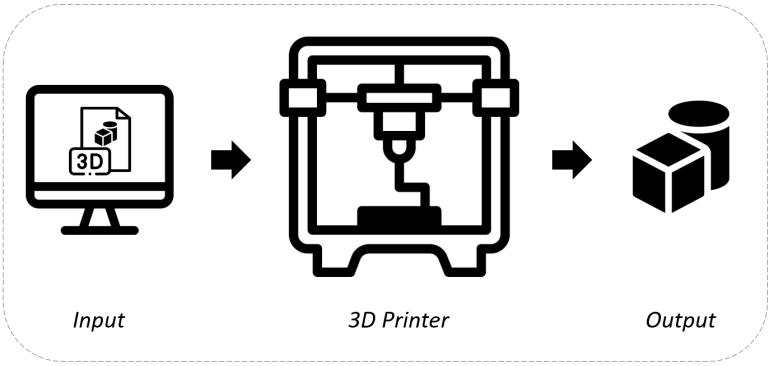

Figure 2.1: The image illustrates the main players in the 3D printing process. **Input**: The digital representation of the object to be 3D printed. **3D Printer**: The device used for 3D printing. **Output**: The physical object obtained.

allows to create physical 3D objects, starting from a 3D digital model. Similarly to a traditional two-dimensional (2D) printer, it receives a digital design as *Input*.

However, it differs in the *Output* since, while a 2D printer reproduces the digital prototype on paper, typically using ink, the 3D printer instead generates a three-dimensional product using a specific type of material (see Figure 2.1).

2.2 Printing Anomalies

In the context of 3D printers, *Print Quality* is a key aspect to consider. It refers to the goodness of the final product, which can be affected by various external factors that must be taken into consideration during a 3D Printing process.

Newer 3D printers, based on Fused Filament Fabrication (FFF), usually implement sensors that are capable to monitor different parameters in order to guarantee a better quality of the final product. However, these checks generally are limited to parameters such as nozzle temperature and build platform and are not capable of performing an exhaustive defects detection. This means that, in many cases, if the process leads to the formation of defects, printing is not interrupted until the operator intervenes, leading to wasted material, effective machine operation time, power, as well as potential generation of malfunctions in machine parts (e.g., nozzle clogging). Since 3D printing is subject to multiple production errors, from small inaccuracies to complete build failures, a skilled employee is necessary to recognise a failure and react accordingly to the type of error. This process is human error prone and requires each worker to constantly supervise the process. In large settings, where several printers operate simultaneously, it is unfeasible to perform a continuous monitoring by means of an operator, since each employee can be assigned to several machines. Furthermore, due to safety concerns that arise from particles and volatile organic compounds (VOC) emitted by the FFF 3D printers, it is mandatory to minimize the physical presence of humans near the printers. In this setting, remote supervision using cameras feeds of the printing process can be used to alleviate the safety problem during the quality monitoring.

One of the main problem is how to determine when a 3D printed product is considered "acceptable". The concept of "acceptable" is not easily standardized as it strongly depends on the product itself and also on its context of use. Since quality is very flexible, *Quality Control* is not easy to fully automate; despite that, some degree of automation could be very useful to speed up the production and save material, time and personnel.

A 3D object can be affected by anomalies, which lead to a degradation in the *Print Quality* of the final product. An *Anomaly* indicates the presence of an element in the form of an irregularity that cannot be traced back to the desired prototype, the triggering cause of which can be multiple. Many different anomalies can be detected during the 3D printing process, in different stages of the operation. The type of anomalies mainly depends on the nozzle system (type of object to be

obtained, nozzle and material) and each of them has different characteristics that can lead to a different reaction by the operator/system. Nevertheless, it is possible to identify a list of the most common anomalies that can be found during a 3D printing process [6]. The list is presented below, together with Figure 2.2 showing a graphical example for each of the indicated anomalies:

1. **Stringing And Oozing**: This type of defect, also known as "hairy prints", consists in small strings of filament left on a printed model. It can be seen as a line of filament that connects different parts of the object to be built and that forms when the nozzle moves but the filament continues to flow from it. Usually it is due to very high printing temperatures and/or the application of an incorrect setting for the retraction mechanism. A stringing defect can be removed manually quite easily, without altering the structure of the object, once its creation is finished. For this reason, it often does not require the stop of the printing process as reaction from the operator/system, but a notification is sufficient.

2. **Spaghetti**: Its name derives from its resemblance to spaghetti food. It occurs when the filament starts to extrude creating a nonsensical pile of product instead of the desired object. Sometimes the print can be saved as the subsequent operations after the point of defect occur correctly and the excess material can be removed at the end of the print.

3. **Nozzle blob**: Nozzle blobs, meaning mass of plastic accumulated around the hotend, often start with a poor initial layer that detaches and starts to spaghetti. Since the hot filament starts to spaghetti so close to the bed, it starts to ball up and become a blob that can even get big enough to cover the nozzle and some of its inner and surrounding areas.

4. **Under Extrusion/Over Extrusion**: In both cases, the resulting print structure resembles the desired model, but the surface has an undesirable texture. Concerning over extrusion the result is to give a melted or uneven finish to the surface. Differently, under extrusion can cause holes, meaning that the material is missing in some print layers. Objects subjected to under extrusion tend to be fragile and are prone to break ups. There can be several factors that lead to under extrusion issue. To mitigate this problem good practices include:

    - Calibrating the first layer;
    - Setting up the slicer to determine the temperature, speed, and how much filament the 3D printer should extrude;
    - Reinstalling correctly the nozzle, when it is replaced, to avoid clogging and leaks;

- Checking for any traces of dirt in the extruder gears that may obstruct the passage of the filament;
- Checking the filament "melt-zone" to avoid it melting too far away from the nozzle.

5. **First Layer Issues**: it is one of the most common 3D printing problems and one of the first you come across. The first layer is essential, as it is the first one you encounter and a mistake on this can easily lead to problems in subsequent layers. However, an object with a poor initial layer, depending on the context of use, could be considered acceptable if the irregularity is slight, if it does not create damage in subsequent layers and if the final product is placed in a context such that the irregularity is not easily visible. Depending on the issue that caused the defect and the shape object being built, a bad first layer can look very different, but in general, the layer looks uneven with possible small pieces of "spaghetti" mixed in.

6. **Layer Shifting**: This issue causes the layers to be shifted with respect to their intended position. It consists in an unexpected movement along the X and/or Y axis that leads the printer to randomly start printing all of its new layers some distance away from the prior layers. The parts of the object above and below the shift point look structurally compatible, but shifted over either right, left, forwards or backwards. Layer shifts are mainly due to incorrect tension of the printer belts or pulleys that are not safe. This kind of anomaly can be considered of high priority since it leads to a result that can hardly be fixed in post production. In this case it is advisable to identify the problem as soon as possible and abort the printing process to avoid wasting resources.

7. **Warping**: Sometimes, especially on large objects, the corners can lift off the bed. This is usually due to a sudden difference that occurs between the nozzle melting temperature and the ambient temperature, which causes shrinkage that lifts and deforms the layers.

8. **Poor Bridging**: The term bridging refers to the printing of layers over the air, without the use of any physical support. In some cases it can be avoided if the model is oriented in a specific way, otherwise gradual overhangs must be created to deal with this situation. Often, these supportless overhangs cause poor bridges with sagging or dropping filaments.

9. **Detach**: This issue happens when a major portion or even the entire print come loose from the surface of the build plate. This can lead to other failures as the part moves around too much for the filament to be accurately positioned for future layers. When the object suffers from this error it is no longer easily recoverable and therefore printing should be interrupted.

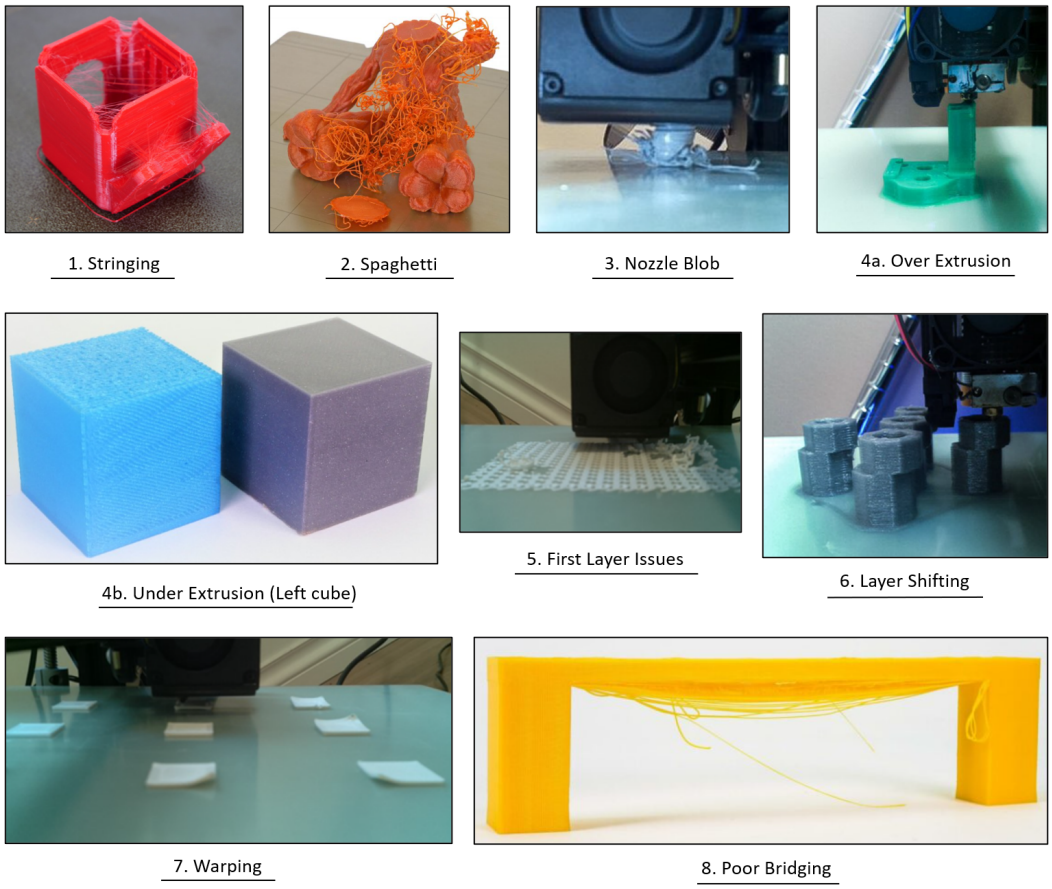

Figure 2.2: Image references: [6], [7]

The vulnerability to errors to which 3D printing is subject has led to the search for new strategies to monitor these processes in a automated way. Thanks to the expansion of Computer Vision in industry applications, some of those approaches have been applied for automated detection and classification of anomalies, also in the context of 3D Printing. In literature, some camera-based approaches can be found such as [8], which basically consists in an algorithmic implementation of error detection, made up of a single camera mounted before the 3D printer together with traditional computer vision and image preprocessing techniques, able to detect three classes of errors (detachment from print bed, discontinuous material flow from nozzle and printed object deformation). Another example [9], proposes a shallow learning image-based classification method to diagnose specific defects, being integrated with a feedback quality control mechanism for automatic machine parameters adjustment. This strategy requires handcraft feature extraction techniques which are hardly able to generalise to different printing conditions, material and setups.

Always in the context of shallow learning, [10] creates a supervised ML method using Support Vector Machines to categorize the printed parts into either 'good' or 'defective', requiring an already successful print to provide comparisons for failure detection.

Next, Deep Learning methods come in allowing to obtain a set of features learned directly from observations of the input images, using a general-purpose learning procedure. This approach is based on expressing complex representations by means of other simpler representations. Some works in the context of Additive Manufacturing have been applied using Deep Learning techniques, such as [11], which presents a real time monitoring system, with a deep learning model (ResNet 50 backbone) able to classify three classes (Under-extrusion, Good-quality, and Over-extrusion) and a feedback loop to modify 3D Printing parameters. Another use case is introduced by [12] which proposes a system made up of one moving camera, together with an adaptive shooting position planning algorithm and a CNN-based model to accomplish efficient defect classification. Other works are available in literature such as [13] which develops a warping detection system using Convolutional Neural Networks (CNN) and [14] which deploys a deep neural network for the recognition of the stringing defect. Another approach is done by [15] which creates a 3D printing error detection and correction system, training a multi-head neural network with image labels constructed in terms of deviation from optimal printing parameters. This study allows to create real-time extrusion AM error detection and correction, being generalisable to several parts, geometries, materials, printers and extrusion methods.

From these considerations, the thesis work proposes an alternative camera-based strategy, based on deep learning techniques, to perform defects detection as soon as possible, during the 3D Printing process. In this case, the camera is fixed and the algorithm takes care of detecting the area of interest below the nozzle for each frame, which will be the only area taken into consideration for the search for any anomalies. The high level overview of the solution is presented in Chapter 4, while the implementation details are specified in Chapter 5. The next chapter introduces a brief recap on Computer Vision and Deep Learning techniques that were used for the development of the anomaly detection model.

Chapter 3

Computer Vision

This chapter presents an introduction to the field of Computer Vision, with a brief review of some of the main tasks solved by applying Machine Learning techniques. In Section 3.2 the Visual Anomaly Detection problem is described, together with several approaches that can be applied to solve it. Lastly, Section 3.3 and 3.4 introduce the two Machine Learning models that have been used in the thesis work to create an "intelligent" system able to perform defects detection in 3D printing processes.

3.1 Background

As described in the book "Computer Vision: Models, Learning, and Inference" by Simon J.D. Prince [16]:

> *"The goal of computer vision is to extract useful information from images. This has proved a surprisingly challenging task; it has occupied thousands of intelligent and creative minds over the last four decades, and despite this we are still far from being able to build a general-purpose "seeing machine."*"

In other words, *Computer Vision* (CV) can be defined as the field of study that aims to develop techniques to help computers and systems understand the content of digital visual inputs, deriving relevant characteristics from different types of sources such as images and videos and taking actions depending on the obtained information. Typically, these methods consist of developing strategies that seek to reproduce the capabilities of human vision. However, while understanding the content of digital images is a trivial task for humans, it still remains an open challenge for machines today.

Technically speaking, it is possible to make a distinction between traditional Computer Vision methodologies, in which both the features and the algorithm are

handcrafted, classic Machine Learning algorithms ("shallow learning"), which take extracted features and use Machine Learning to build a model, and Deep Learning networks, which build an end-to-end pipeline, starting from the training data, as pixels, to the output (see Figure 3.1).

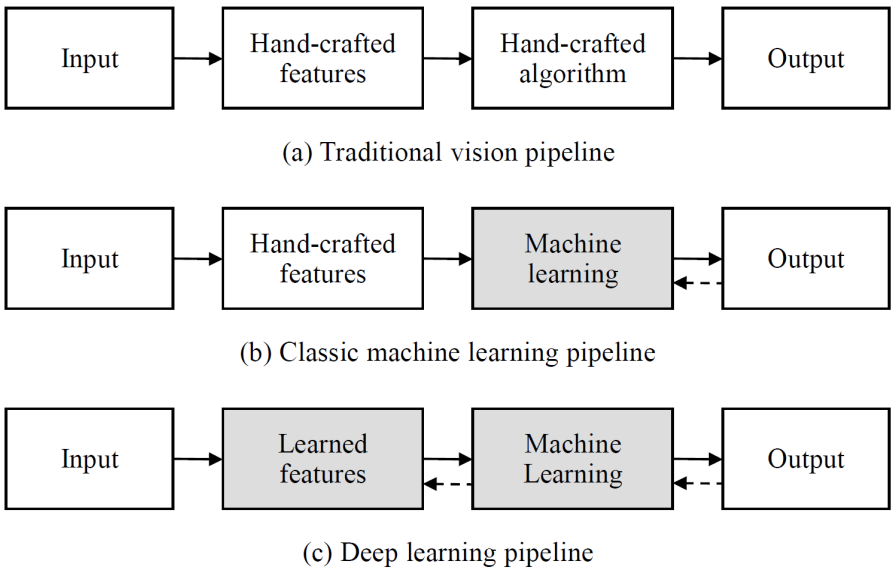

Figure 3.1: Image reference: [17](2022, Figure 5.2).

In recent years, thanks to the considerable availability of images on the Internet, together with the progress in computational efficiency, a large variety of Computer Vision applications have been solved by ML techniques, which require an important unbiased representative training data to obtain good results on real-life inputs. Specifically, deep convolutional architectures have undergone a dramatic acceleration over the past decade, becoming the primary choice for many tasks.

In general, Machine Learning algorithms can be categorized as either *supervised*, where each instance, belonging to a collection of input data $\{x_i\}$, is paired with the corresponding target label from $\{t_i\}$, or *unsupervised*, in which data come without labels. Another possible setting is *semi-supervised* learning, in which the target labels are provided only for a small subset of data. As regards *unsupervised learning*, the main goal is to find interesting patterns, regularities, subgroups among the observations, without any supervision in the learning process. Differently, in *supervised learning* the pair training inputs and outputs is given as input to the learning algorithm, to estimate the best model parameters in terms of prediction ability (training phase). Once the model is trained, the best parameters previously learned are frozen and new unseen inputs are fed to the model to generate a prediction with a corresponding output value (inference phase). The output can

be either a discrete label for classification tasks, where the aim is to predict the class membership from a set of possible classes, or a scalar/vector real value, for regression tasks. The whole *supervised learning* procedure explained above is illustrated in Figure 3.2.

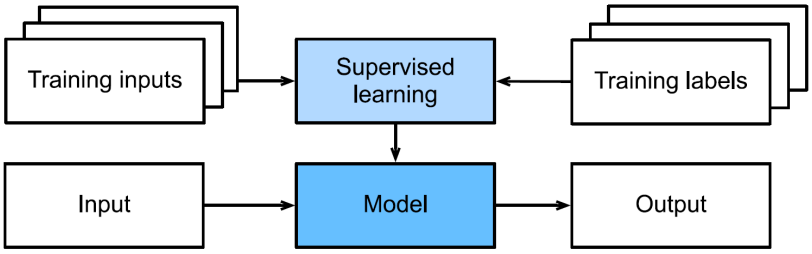

Figure 3.2: Image reference: [18](2021, Figure 1.3).

Among the various Computer Vision topics, *Visual Recognition* has had a great development in the last decade, thanks to the availability of large quantities of labeled images and to the step forwards obtained in the field of Deep Learning; this has meant that many modern techniques in this field are nothing more than natural applications of deep neural networks. *Visual Recognition* includes different tasks such as:

- *Class Recognition*: also known as *Image Classification*, which consists in categorizing each input image into the appropriate class. In literature both classic handcrafted feature-based approaches (with optional ML to perform the final classification) and modern deep neural networks systems have been applied.
- *Object Detection*: whose main objective is to locate and classify several categories of objects in an image, delimiting them with bounding boxes. In this task images can contain multiple objects, with different sizes and positions, which eventually overlap.
- *Semantic Segmentation*: which consists in delineating the objects at the level of pixels, meaning that each pixel is assigned to a class (per-pixel class labeling). This task includes some variants such as *Instance Segmentation*, where different labels are given for pixels being part of different instances of the same object type, *Panoptic Segmentation* and *Pose Estimation*.

By combining two or more of the tasks introduced above, it is possible to define ML pipelines that can be helpful for solving different industrialization problems like the *Visual Anomaly Detection* of the final product during the production stage.

The next sections give a general overview of this application in the context of Computer Vision, together with the two deep learning strategies that have been applied in the thesis work to solve the main problem proposed.

3.2 Visual Anomaly Detection

Anomaly Detection is a well-known problem that still presents many challenges in the area of Machine Learning applications. It generally refers to distinguish among abnormal patterns (anomaly) that deviate from the common behavior and normal samples that represent the norm in the data. When dealing with images, the problem in literature is framed as *Visual Anomaly Detection* or *Image Anomaly Detection* [19].

Concerning *Visual Anomaly Detection* a large variety of methods can be developed to solve practical use cases, an example of which regards *Defects Detection*, applied to the manufacturing sector. Depending on whether supervised information, including both normal and abnormal samples, is available or not, it is possible to divide the Visual Anomaly Detection in two different approaches: *supervised*, where sufficient availability is required of both normal and defective annotated samples, and *unsupervised*, where models are usually trained using only normal images. Another possible distinction refers to the labels granularity, that can be at image-level or pixel-level. Specifically, in image-level detection the objective is to identify if the entire image is normal or not, while in pixel-level detection the model tries to localize inside each image specific abnormal regions.

More in detail, *Supervised Visual Anomaly Detection* requires a large amount of annotated examples, especially when Deep Learning approaches are used. When dealing with large-scale industrial applications, a good amount of data is often available; however, a problem still remains, due to the nature of the task in question, consisting in the fact that the amount of data relating to samples with defects is considerably less than that without defects, which leads to a class imbalance situation. In literature, some supervised examples, applied to industrial anomaly detection, include: Lin et al. [20], which propose a multi-scale cascade CNN to perform surface defect detection, with a two stage augmentation method and a asymmetric loss function to mitigate the imbalance between positive and negative samples; Huguan et al. [21], which develop a compact CNN, consisting of a lightweight bottleneck and a decoder, for surface defect inspection on low-frequency CPUs; Božič et al. [22] which present an end to end training scheme for a two stage deep learning architecture, with a dynamically balanced loss between segmentation and classification networks, as well as a gradient flow adjustment. Other possible industrial applications include: [23] and [24], which create two different methods for defects detection, by considering only image-level labels during the learning

process; Božič et al. [25] which, again in the context of surface-defect detection, propose a deep learning technique using mixed supervision to alleviate the cost of labelling, present in fully supervised settings. Looking in more detail, this solution is obtained by implementing two subnetworks trained simultaneously using respectively pixel-level annotations in the first and image-level annotations in the latter. Another work, this time concerning cracks detection in aircraft structures, is presented by [26], which basically consists in a lightweight version of YOLOv3, named YOLOv3-lite, which allows fast and accurate crack detection. A last example is the one presented by [27], where a defect detection system is build as a three stage cascaded architecture, including two detectors, with the aim of sequentially localize the cantilever joints and the associated fasteners, and a classifier for fasteners defects diagnosis.

Differently, *Unsupervised Visual Anomaly Detection* helps in reducing the need for a large amount of labelled data, since the model is trained using only defect-free images, but it does not take advantage of the information about defective samples during training if available. This setting is very useful when anomalous items are very rare, variable in patterns and difficult to collect.

Since the thesis work aims to create a camera-based "intelligent" system, helping in *Defects Detection* during a 3D printing process, the main target in this case is expressed in terms of visual data, presented as video frames. Considering the fact that the 3D printing process takes place in a controlled environment, where it is possible to identify a certain type of systematic errors and that a good amount of both types of data (normal and defective) will be available once the camera-based system will be mounted, the design choice fell on a image-level supervised setting, consisting of two deep learning strategies applied in cascade, the main aspects of which are introduced in the following two sections.

3.3 YOLO

Current mainstream object detection frameworks can be categorized in two different types, *two stage* and *single stage* object detectors, both of which are based on deep learning strategies. *Two stage* solutions follow a Region Proposal based approach, which consists in generating the Regions of Interest (RoIs) at first and then classifying each proposal into a category, by applying a location refinement on the previously generated RoIs. Some examples include: R-CNN, Fast R-CNN, Faster R-CNN, Cascade R-CNN and FPN. These methods usually outperform single stage object detectors in terms of detection accuracy, but their complex architecture, made up of different components, may become a bottleneck in terms of inference time for real time applications. *One stage* solutions help reducing the time needed for computation, framing object detection as a regression/classification

problem, mapping directly from image pixels to bounding boxes coordinates and class probabilities. Main examples in this setting include: You Only Look Once (YOLO) [28] and Single Shot MultiBox Detector (SSD).

As regards the thesis work, a one-stage detector is preferable as one of the requirements is the model to be able to identify anomalies in near real time, processing frames from a streaming video. Therefore, the choice fell on YOLO, which is a popular real-time one-stage object detection system, consisting of a single CNN trained on full images to simultaneously predict multiple bounding boxes and class probabilities for those boxes. More in detail, YOLO constructs a detection pipeline by applying a single network, which reasons globally about the entire visual input and all the included objects, to make a prediction. The basic idea behind the method is the following:

- The input image is split into a SxS grid of cells, where each cell is responsible for predicting an object, if the center of the object falls within the cell.
- Each grid cell predicts B bounding boxes, with the corresponding confidence scores. Formally, the bounding box confidence score is defined as $Pr(Object) * IoU_{pred}^{truth}$ where, $Pr(Object) \geq 0$ reflects how likely the box contains an object (objectness) and IoU_{pred}^{truth} is the Intersection over Union, between the predicted box and the ground truth, which indicates how accurate is the bounding box prediction. If no object is present in the cell, the confidence should be set to 0.
- Each bounding box prediction involves: (x, y) coordinates representing the center of the box, computed as offsets to the corresponding cell; width and height of the box, normalized by image width and height; bounding box confidence.
- A class prediction is also based on each cell. The conditional class probability ($Pr(Class_i|Object)$) is the probability that the detected object belongs to a particular class where, $Class_i$ identifies the specific class of interest among all the C classes.
- At test time, the class-specific confidence score is computed multiplying the conditional class probability and the box confidence as follows:

$$Pr(Class_i|Object) * Pr(Object) * IoU_{pred}^{truth} = Pr(Class_i) * IoU_{pred}^{truth}$$

  This gives a measure, for each box, of the confidence on both classification and localization. The final predictions are encoded as a $SxSx(B * 5 + C)$ tensor.

To measure the accuracy of each bounding box, YOLO uses the Intersection over Union (IoU), also known as Jaccard index or Jaccard similarity coefficient, which

is the most common metric for comparing the similarity between two arbitrary shapes. Formally, the IoU is calculated considering the predicted (B_{pr}) and the ground truth (B_{gt}) bounding boxes for an object and computing the ratio between their area of intersection and their area of union:

$$IoU = \frac{B_{pr} \cap B_{gt}}{B_{pr} \cup B_{gt}}$$

Once the object detector has produced a fixed number of bounding boxes per image, these regions are run through a Non-Maximum Suppression (NMS) step to remove duplicates for the same object, which have a lower confidence. At high level the algorithm can be explained with the following steps:

- First, all the boxes with a confidence score lower than a specified threshold are discarded;
- Then, the bounding box with highest confidence score is selected as current element and the overlap (IoU) between this box and the other bounding boxes, of the same class object, is computed. At this point, all the boxes with $IoU \geq IoU\ threshold$ are removed. This process is repeated with the remaining bounding boxes, until there is no more reduction of boxes to perform.

During training, YOLO uses sum-squared error between the predictions and the ground truth to calculate the loss, as shown in Figure 3.3. More in detail, the loss

$$\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Localization loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Confidence loss

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Class loss

Figure 3.3: Image reference: [28]. **In blue**: *Localization loss*, for bounding-box coordinates. **In green**: *Confidence loss*, for bounding-box score prediction. **In red**: *Class loss*, for class-score prediction.

formulation shows that, for a certain cell i: (x_i, y_i) represents the center for the box; (w_i, h_i) refers to width and height of the box; C_i is the confidence score; $\mathbb{1}_j^{obj}$ is an identity function, set to 1 if object is present in cell i; $\mathbb{1}_{ij}^{obj}$ denotes that the jth bounding box predictor "conduces" that prediction. In the original paper, the loss terms are modulated by two scalar meta-parameters, λ_{coord} and λ_{noobj}, which multiply the *Localization loss* and the object *Confidence loss* respectively. In detail, these coefficients are set to $\lambda_{coord} = 5$ and $\lambda_{noobj} = 0.5$, to increase the weight for bounding box coordinate predictions and to decrease the influence of no object loss.
The original YOLO framework uses a custom network based on GoogLeNet [29] as backbone, made up of 24 convolutional layers and 2 fully connected (FC) layers, of which some convolution layers use 1 $\times$ 1 reduction layers followed by 3 x 3 convolutional layers. Later, new versions have been created by Joseph Redmon, the main author: YOLOv2 [30] which proposes a new model named DarkNet-19 and YOLOv3 [31] which introduces a larger version, named DarkNet-53, to improve performances. Other newer versions include: YOLOv4 [32] by Alexey Bochkovskiy, which uses CSPDarknet53 as backbone network, PANet neck for feature aggregation, YOLOv3 head structure and it also introduces, among others, mosaic data augmentation technique; YOLOv5 by Glenn Jocher, which uses a model architecture close to YOLOv4, with some specific training procedures implemented in Pytorch. Concerning the thesis work, the choice fell on YOLOv5 which provides a framework for fast training and easy deploy.

3.4 EfficientNet

EfficientNet [33] is a family of CNN based models for image classification, released by Google in 2019. These models result in better accuracy and efficiency, on both ImageNet and other transfer learning datasets, with respect to other state of the art solutions, such as GPipe and ResNet. More specifically, a new mobile-size baseline network is designed, called EfficientNet-B0, together with an effective method for performing model scaling to any target resource constraints, allowing to obtain maximum accuracy gain, while maintaining model efficiency. The intuition for the new scaling method derives from the fact that the authors have empirically observed how the scaling dimensions are mutually dependent, suggesting the need to coordinate and balance the different contributions. Previous studies have already attempted to arbitrarily scale depth, width and image resolution, but this process requires manual tuning and often leads to sub-optimal results in terms of accuracy and efficiency. The new compound scaling method instead allows to uniformly scale all these dimensions using a fixed set of coefficients, chosen applying a small grid search on the baseline model. Figure 3.4 shows the different model scaling

approaches, where (a) represents the baseline network, (b) to (d) indicate the conventional methods that only consider one dimension of the network to scale up and (e) which is the novel structured method based on compound coefficient. The EfficientNet baseline is built performing a multi-objective neural architecture

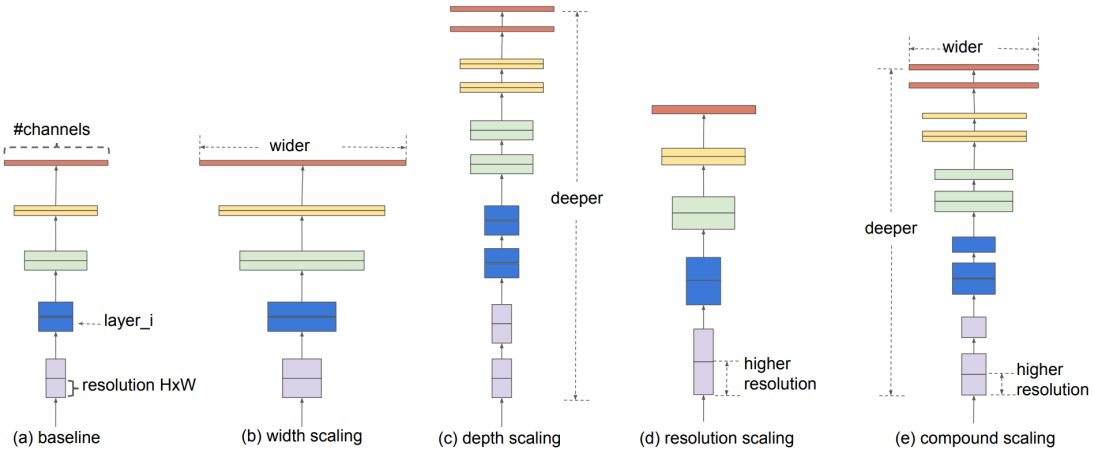

Figure 3.4: Image reference: [33](2020, Figure 2).

search to obtain an efficient network architecture, optimizing both accuracy and FLOPS (Floating Point Operations Per Second). The main network building block consists in a mobile inverted bottleneck (MBConv), similar to MobileNetV2 [34] and MnasNet [35], to which squeeze-and-excitation optimization is added. Starting from this baseline, the scaling method is applied to scale up EfficientNet-B0 and obtain the models from EfficientNet-B1 to B7.
To sum up, EfficientNet models, obtained as explained above, are able to surpass state of the art methods in accuracy beyond reducing model parameters and FLOPS by an order of magnitude. In addition, the adopted compound scaling method is an effective strategy, that can be easily applied to other baseline ConvNets. As regards the thesis work, the EfficientNet-lite model is adopted, which provides a mobile/IoT device friendly solution.

Chapter 4

3D Printing Anomaly Detection model

This chapter is focused on presenting an overview of the solution proposed for solving the main problem of the thesis work. Section 4.1 describes the high level ML pipeline used by the system, while from Section 4.2 to 4.5 more details about the components and how each of them interacts with the others are given.

4.1 Architecture

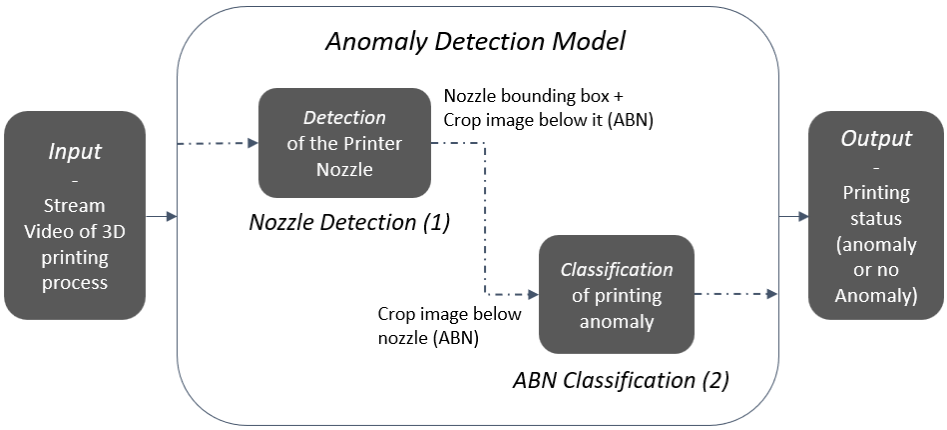

Figure 4.1: High level pipeline for Anomaly Detection (inference time)

The Machine Learning strategy proposed uses an object detection algorithm for the detection of the 3D printer nozzle, together with a classification algorithm to classify the area below the nozzle. The solution is based on the use of *You Only*

Look Once (YOLO) as real-time object detection system and *EfficientNet-Lite* as computer vision model for classification.

As shown in Figure 4.1, the *Anomaly Detection Model* is made up of two main phases, *Nozzle Detection (1)* and *ABN Classification (2)*, which are performed sequentially at inference time. The *Input* to the model is a video stream, while the high level *Output* is simply a flag, for each stream frame, indicating the absence or presence of an anomaly. Concerning the *Nozzle Detection*, the input coincides with that of the model, while the output includes a bounding box surrounding the nozzle, together with an image crop below it. The bounding box is simply a rectangle, which is used in the context of object detection to describe the spatial location of the object to detect; in this case it refers to the position of the nozzle inside the single video frame the model is processing. Once the position of the nozzle is detected for the specific frame, a square image crop is generated, which identifies the Area Below the Nozzle (ABN) where the algorithm will focus in looking for possible printing anomalies. This means that, for each frame, the solution proposed only focuses in a specific area, thus any other possible anomaly that is visible in other locations of the frame is not considered at that time. This is not a limit but simply a design choice as the goal is to detect an anomaly that occurs in that precise moment or in any case in that specific position, but maybe in some previous layer (not critical anomaly). Then, the ABN identified in the first phase is taken as input by the *ABN Classification* phase, which takes care of classifying it. The output of this second phase, which also coincides with the output of the model, is one of two possible labels: anomaly or no anomaly.

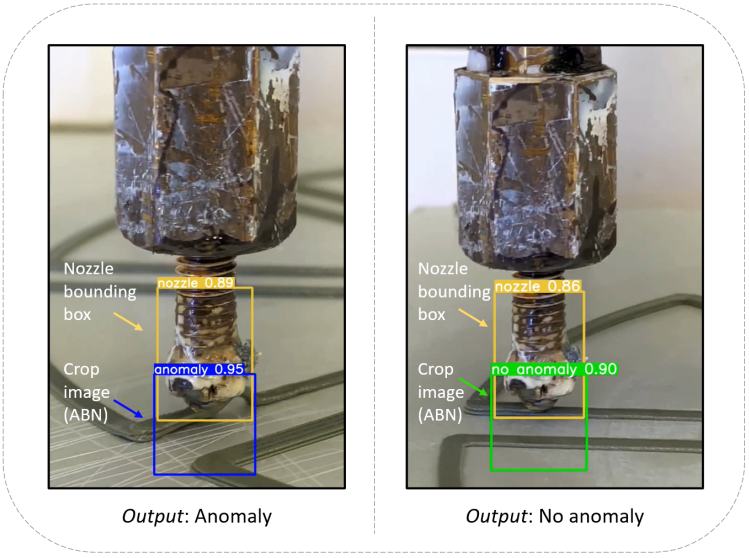

Figure 4.2: Visual Anomaly Detection output for two different frames.

In Figure 4.2 it is possible to observe a real example of visual output which takes into account two different frames of the video stream and shows the output related to both ML phases. Two sub-images are displayed having a yellow bounding box that delimits the nozzle, together with the associated confidence level. More in detail, as regards the image on the left, this indicates the presence of an anomaly (detach) delimited by a blue square, while the one on the right refers to normal behavior, which is localized by a square outlined in green.

To sum up, starting from two separated phases (*(1)* and *(2)*), which mainly consist of two well defined tasks in machine learning (respectively Detection and Classification), their integration, obtained following the steps described above, allows to create an *Anomaly Detection Model* to perform near Real-Time Defects Detection during a 3D printing process.

4.2 Input

The *Input* consists in a Video Stream showing a specified 3D printing process. Technically speaking, a video can be considered as a succession of still images (frames) displayed as a rapid sequence (see Figure 4.3). If the frequency is high enough, the images are perceived as moving and no longer as a chain of still pictures. A stream is a "channel" that allows the flow of a potentially unlimited sequence of

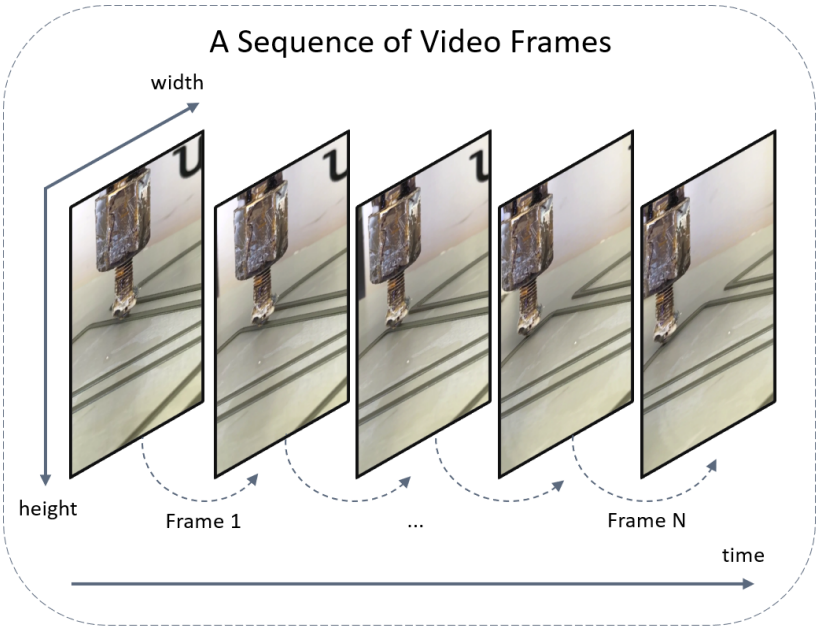

Figure 4.3: Video as a sequence of frames, showing a 3D printing process. Each frame is expressed as a grid of pixels (width x height), with the units in pixels.

data elements between the source and the destination. Video streaming involves sending video content in a continuous flow of compressed data over the network, showing the result to the client as soon as it arrives. One of the main characteristics of a video stream is the frame rate, expressed in frames per second (fps), which indicates the frequency at which successive still images are displayed.

The concept of video as sequence of frames is important since the proposed *Anomaly Detection Model* computes the anomaly prediction frame by frame, meaning that a nozzle bounding box and an output label (anomaly/no anomaly) are generated individually for each frame of the input video stream. The model can handle streams with different fps, but the ability to detect an anomaly in real time highly depends on how fast the underlying hardware is able to process a single frame. The supported protocols to handle a stream are the following:

- **Real Time Streaming Protocol (RTSP)** [36]: is an application layer network protocol that provides an extensible framework for transport control of multimedia data streams (such as audio and video) over an appropriate transport protocol. RTSP controlled streams can use the Real-time Transport Protocol (RTP) protocol, together with the Real-time Control Protocol (RTCP) protocol, for the media stream delivery, but the RTSP functioning does not depend on the specific transport mechanism used for the multimedia data.
- **Real Time Messaging Protocol (RTMP)** [37]: is a Transmission Control Protocol (TCP) [38] based protocol, originally developed as proprietary, which maintains persistent connections and helps provide a smooth low-latency streaming communication.
- **Hypertext Transfer Protocol (HTTP)** [39]: is a stateless application level protocol which requires a reliable network transport communication to exchange data between a client and a server. Usually, the communication takes place over TCP/IP connections, but this does not prevent HTTP from being implemented on top of other protocols on the Internet, or on different networks.
- **Hypertext Transfer Protocol Secure (HTTPS)**: is an extension of HTTP which provides secure communication using SSL, or its successor TLS.

4.3 Detection Task

The *Nozzle Detection* phase consists in a Detection Task which aims to identify, sequentially for each frame of the input stream, where the nozzle is located. The solution uses *YOLOv5* [40] (release 6.1 [41]) as nozzle detector, which is a specific version of *YOLO*, implemented by Glenn Jocher. *YOLOv5* is based on the Pytorch

framework [42] and is made up of a family of compound-scaled object detection models pretrained on the Microsoft Common Objects in Context (MS COCO) dataset [43]. It includes, among others, functionalities such as model ensembling, hyperparameter evolution and allows different export formats (ONNX, CoreML, TFLite).

MS COCO is composed of several large-scale datasets created in different years, each one focused on a different computer vision task such as: Object Detection, Keypoint Detection, Panoptic Segmentation and Dense Pose. The annotations file, for the Object Detection Task, contains coordinates of bounding boxes and full segmentation masks for 80 categories of objects found in everyday life in their natural environments and depicted in varied viewpoints.

YOLOv5 is available in five main versions: nano (*YOLOv5n*), small (*YOLOv5s*), medium (*YOLOv5m*), large (*YOLOv5l*), xlarge (*YOLOv5x*). The larger the model is, the more, in most cases, it produces better results, but also requires more parameters and therefore more CUDA memory [44] to train and is slower during inference phase.

The specific model used for the thesis is *YOLOv5n* [45], which is the smallest (4 MB) and faster pretrained model among the available versions and consists of about 1.9M parameters, considering weights and biases. With respect to *YOLOv5s*, nano models maintain a depth multiple of 0.33 but reduce the width multiple from 0.50 to 0.25, resulting in about 75% fewer parameters, from 7.5M to 1.9M, which is ideal for mobile and CPU solutions.

4.4 Classification task

The *ABN Classification* phase consists in a Classification Task which aims to categorize the Area Below the Nozzle (ABN), distinguishing between anomaly and no anomaly image crops. The task in question is a binary classification, as the objective is to predict a qualitative response variable for each new observation, thus dividing the data into two groups. The solution is based on *EfficientNet-Lite* [46], which is a set of image classification models suitable for mobile, microcontrollers, IoT and other types of edge devices. *EfficientNet-Lite* runs on Tensorflow Lite [47] and is designed to bring the performance of *EfficientNet*, allowing fast inference to small devices (CPU, GPU and EdgeTPU).

With respect to *EfficientNet*, the *Lite* version presents some architectural changes, mainly implemented to address the hardware heterogeneity on the edge devices. The changes are the following:

- The removal of Squeeze-and-Excite (SE) networks [48], that are not well supported for certain mobile accelerators;

- The replacing of all swish activations [49] with RELU6 [50], which notably increases the quality of post training quantization;

- The fix of the stem and head while scaling models up to reduce the size and computation and to obtain smaller and faster scaled models.

EfficientNet-Lite is available in five variants, *efficientnet-lite[0...4]*, in which the numerical index grows as the size (and latency) of the model increases, as well as the accuracy. The version chosen as backbone is *EfficientNet-Lite0*, that is the smallest model (4.7M parameters) and allows low inference time latency.

Transfer Learning technique [51] is used to adapt the classification model, pretrained on the large ImageNet [52] dataset, to the 3D Printing Defects dataset. This is useful since the training dataset, available for the task in question, is too small to train a full-scale model from scratch. Generally speaking, the Transfer Learning paradigm is based on preserving and applying previous knowledge learned from one or multiple existing sources, to improve learning on a new target domain. The use of Transfer Learning in the context of Deep Learning mainly consists in taking the feature maps, typically learned on a large-scale image classification task, and leveraging them on a new alike problem. The intuition behind is that the features extracted from a large and general enough dataset, give a good starting point for the new task to learn. The high-level Transfer Learning pipeline, in the context of Deep Learning, is the following:

1. Take the pretrained model as backbone, removing the classification head and freezing all the layers in it, in order to keep the information previously learned. This is also called base model and contains meaningful features that can be generically useful for different tasks.

2. Add a new classification head on top of the base model, made up of some trainable layers, randomly initialized. The architectural choice made for the head model is the following:

    - A Global Average Pooling Layer [53] for spatial data, to convert the block of features extracted from the base model into a single element vector per image.
    - A Dropout Layer [54], which is used as a regularization technique to prevent overfitting the training data.
    - A Dense Layer with two output units, also called Fully-Connected Layer, that converts the input features into a single prediction per image. Softmax [55] is used as activation function for the output layer of the classification network, since it allows to interpret the result as a probability distribution.

3. Train the entire model by keeping the convolutional layers frozen (use the base model as feature extractor) and update the new classification head layers on the target dataset. This process is also called feature extraction.

4. Perform Fine-Tuning, which involves unfreezing all the convolutional layers (or some of them) and retraining them together with the new head, by applying a very low learning rate. Typically, the unfrozen layers are the ones on top of the base model, since they are associated to higher-level feature representations which are more relevant to the specific task, while low level features are more general and can be "reused" as they are, without being updated.

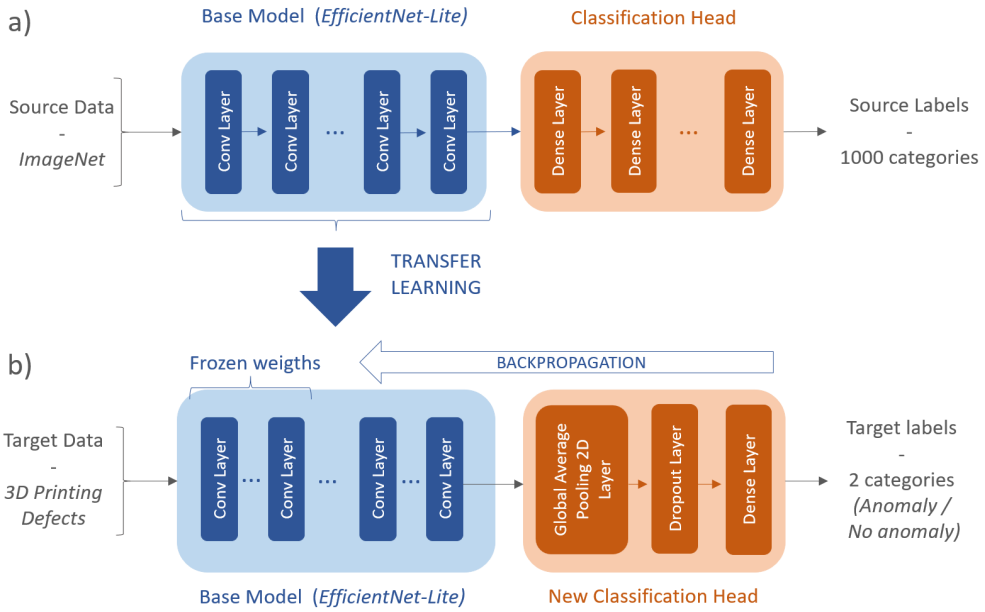

Figure 4.4: In blue: The *EfficientNet-Lite* backbone. **In orange**: The classification head.

Figure 4.4 shows the Transfer Learning workflow explained above, applied to the *ABN Classification* task. Specifically, the upper part a) is related to the EfficientNet-Lite architecture comprising of backbone and classification head, while the lower part b) corresponds to the architecture modified for the task under analysis. In particular, sub-figure b) describes the fine tuning phase, where the "BACKPROPAGATION" arrow visually explains the layers weights that are updated during training procedure, while the "frozen weights" label refers to those that remain frozen during this process. The same sub-figure also highlights how the frozen layers correspond to the first ones in the architecture structure, which is the typical approach used for the reasons explained before in this section. Looking

at the two sub-figures a) and b), another aspect to remark is the difference in the inputs given to the model and in the outputs produced. Referring to the naming convention commonly applied in the context of Transfer Learning, "source data" relates to the ImageNet data on which the model has been previously trained, while "target data" represents the 3D Printing Defect dataset, created specifically for the objective of the thesis. The details about the construction of this dataset, together with the labelling process are explained in Section 5.2.1. As for the output, the difference is that in b) the task is a binary classification that provides two categorical labels ("target labels"), while in a) the pre-trained network, including the head, is capable of classifying any image that falls into any of the 1000 available categories ("source labels") in ImageNet. All the technical specifications about the training procedure and the application of the Transfer Learning technique are available in Section 5.2.2.

4.5 Output

The main *Output* of the *Anomaly Detection Model* is simply the label predicted by the Classification Task. A class label is associated to each frame of the input video stream, among the two available classes: anomaly or no anomaly.

A single run of the algorithm corresponds to an input pair (Video Stream ID, timestamp) together with an output, which consists in a Comma-Separated Values (CSV) file containing all the results for that run. A CSV is a text file format, where each line (or record) is subdivided in a well defined number of fields, divided by a special separator character (e.g. comma). While the Anomaly Detection program is running, a line is written, in append, each time a prediction on a new frame is computed. Aside the classification labels, additional useful information are added to the CSV file:

- **Timestamp**: is a string indicating date and time related data, which help identify when a certain event took place. In this case, it is associated to the prediction of a single frame, meaning that each time a new classification label is issued, a new line, identified by the timestamp, is added to the file. Timestamp follows the ISO8601 standard [56]:

  YYYY-MM-DDTHH:MM:SS.mmm

  where,

  - YYYY-MM-DD is the date expressed in Year (YYYY) as four-digit format, Months (MM) and Days (DD).
  - T is the default separator character put between the date and time fields.

  - HH:MM:SS.mmm is the time expressed in hours (HH), minutes (MM), seconds (SS) and milliseconds (mmm).

- **Prediction label**: is the classification label expressed as string, having two possible different values: anomaly or no anomaly.
- **Prediction**: is the classification label expressed as number, having two possible different values: 1 (for anomaly) or 0 for (no anomaly).
- **Nozzle prediction**: is related to the Detection Task and indicates the class-specific confidence score, per bounding box. This gives a confidence score for each class, for each bounding box. Since in this case of study the model is trained for only one class, the Nozzle class, the classification loss is not calculated during training and is set to 0. This translates into the fact that the final confidence score is equal to the box confidence score, without considering the conditional class probability that the detected object belongs to a particular class.
- **Anomaly prediction**: is related to the Classification Task and indicates the confidence score for a single image crop. The last layer of the classification head outputs two numbers which correspond to the scores of each class, namely 0 and 1. The softmax activation function converts those scores into a probability distribution. The one with the highest value is taken as final confidence score.
- **Bounding box nozzle**: refers to the coordinates of the bounding box surrounding the nozzle, expressed in the YOLO format (see Section 5.1.1).
- **Bounding box anomaly/no anomaly**: refers to the coordinates of the Area Below the Nozzle (ABN) where the algorithm focuses in searching for anomalies.

Whenever an anomaly is detected by the algorithm, a warning is generated and the operator, assigned to the 3D printing machine, reacts accordingly. Depending on the type and severity of the anomaly, the printing process may or may not be interrupted. Since the concept of quality is flexible in the context of 3D printing (see Section 2.2), full automation is not easy to implement, at least at this early stage of the project. For this reason, the final choice is currently left to the operator, who will decide whether to continue printing or stop the process. The *Anomaly Detection Model*, in this first draft, helps to speed up the error detection phase compared to a manual control and therefore to save product and time.

Chapter 5

Implementation

This chapter focuses on describing the implementation of the *Anomaly Detection Model*, introduced at a high level in Chapter 4, and its application to printing machines that produce 3D lamps. Sections 5.1 and 5.2 illustrate the Machine Learning pipelines, respectively for the *Nozzle Detection* and for the *ABN Classification* phases, both described at a high level in Chapter 4. All the training processes are performed on hardware belonging to Google Colab (Pro plan). The CPU model is a Intel Xeon, supported by a 32 GB RAM, while the GPU is a Tesla T4 instance, with a 16 GB dedicated RAM. Section 5.3 presents the application of the *Anomaly Detection Model* in its entirety, during inference phase.

5.1 Nozzle Detection

The main objective of the Machine Learning (ML) pipeline, related to *Nozzle Detection*, is that of creating an object detection ML algorithm with the aim to identify the nozzle during a 3D printing process in the lighting company plants. To achieve this goal, two distinct phases are carried out sequentially: the *Training Phase* of the detection model and the *Inference Phase*, where the model is put into action on new data. In Figure 5.1 it is possible to observe the complete flow for the *Nozzle Detection - ML Pipeline*.

The *Nozzle Detection - Training Phase* consists of two main parts. The first one is devoted to the *Data Acquisition and Preparation*, which shows how the dataset, related to the lighting company's nozzle systems, is created. The second one instead focuses on the implementation of the *DL Model Training*, which is applied on this specific task.

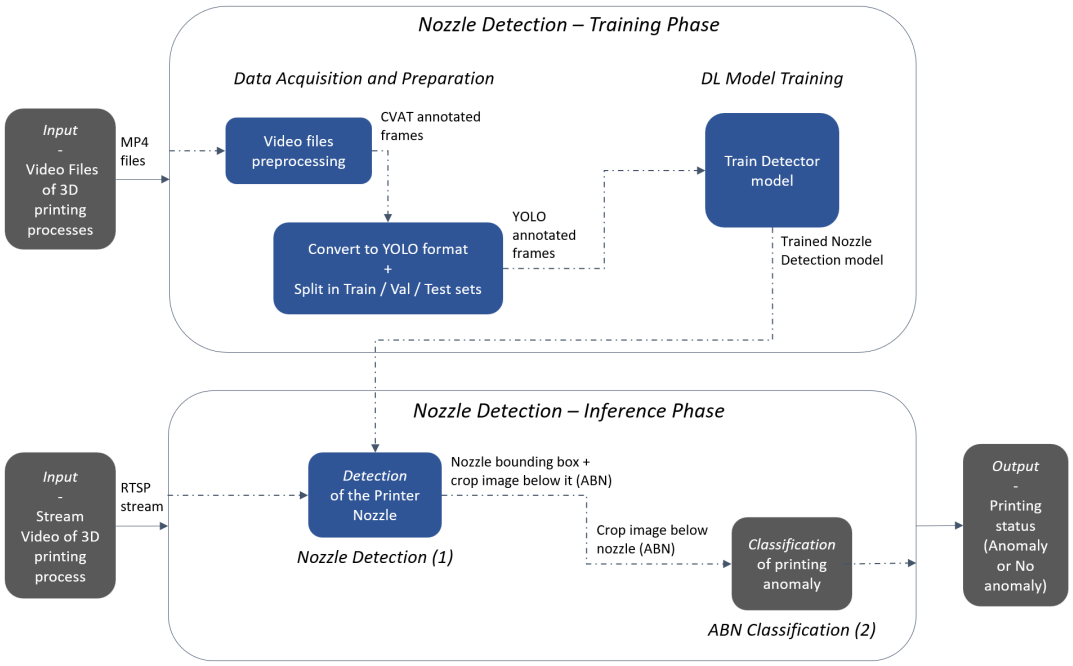

Figure 5.1: Nozzle Detection - ML Pipeline: the part relating to the Nozzle Detection (training + inference) is highlighted in blue.

5.1.1 Training Phase - Data Acquisition and Preparation

Since no datasets, for the activity in question, were made available by the lighting company, it has been necessary to include a preliminary phase of data acquisition and preparation, aimed at creating an ad hoc dataset for the printers available in their plants. The data used to train the object detection model consist of MP4 files showing the 3D printing process of different kind of lamps, produced by the company. Each file consists in a video, which can be seen as a single task, containing a specific nozzle and type of lamp to be printed. The video tasks provided do not present the entire process but only a portion: some show the first print layers where the printed product is minimal, while others contain frames with already many layers previously deposited. Figure 5.2 shows the two main types of nozzles that have been identified by manually inspecting the videos provided by the lighting company.

Every object detection algorithm needs a dataset appropriately labelled, in the form of bounding boxes. As regards the thesis work, the annotation phase consists in creating bounding boxes surrounding the nozzle for each frame of the analysed video. This is done considering two different alternative strategies, which can be seen graphically in Figure 5.3.

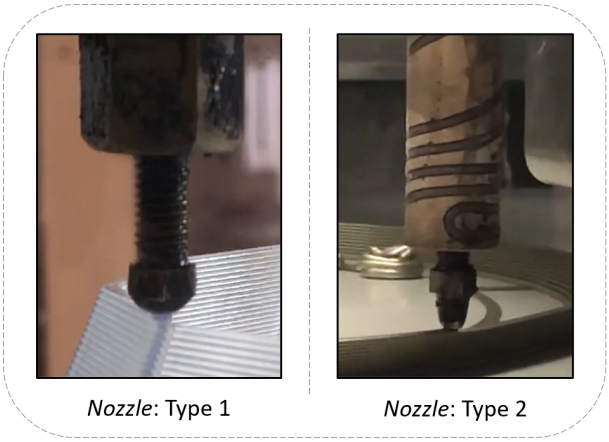

Figure 5.2: The two types of nozzles identified in the lighting company's 3D printing processes.

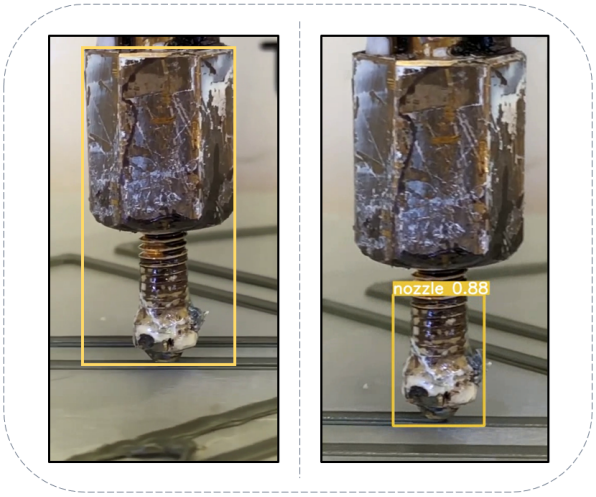

Figure 5.3: The solution (on the left) shows a bounding box in yellow enclosing the nozzle plus a portion above it. This first try leads to bad detections concerning the nozzle of type 2. The second solution (on the right) is able to better detect both types of nozzles and for this reason this is the one chosen in the final implementation.

Before performing the labelling, the videos are pre-processed to remove the frames in which the nozzle is not present and excluding one frame out of three to avoid overfitting during model training. Furthermore, the videos are trimmed in such a way that only one complete turn is considered if no anomalies occur in the following neighbouring turns.

To obtain those type of annotations, Computer Vision Annotation Tool (CVAT)

[57] is used, which is a free, open source, web-based image and video annotation tool, used for labelling data for computer vision. The output of this annotation step consists in a folder for each video task, which follows the naming convention described below:

task_<video_id>_<YYYY-MM-DD>

where,

- **task**: is a common keyword to indicate that the folder contains information about a video task.
- **video_id**: is the video identifier which should be unique and representative of the printing process.
- **YYYY-MM-DD**: is the date expressed in Year (YYYY) as four-digit format, Months (MM) and Days (DD), in which the annotation was made.

Each task follows the CVAT annotation format and contains within it a folder named "images", with all the frames using the PNG extension, and a XML file, with all the annotations. Below the directory tree structure obtained is presented.

```
/
└── raw
    ├── task_<video_id>_<YYYY-MM-DD>
    │   ├── images
    │   │   ├── frame_<frame_number>.PNG
    │   │   └── <...>
    │   └── annotations.xml
    └── <...>
```

In this structure, the name of a frame is given by a 6-digit number indicated by the label *frame_number* and prefixed by the keyword *frame*.

A script is then used to convert the bounding box labels into the format compatible with the object detection model chosen, which is *YOLOv5*. For each frame, the YOLO format uses two files with the same name but different extensions. One file is a PNG image file and the other is a TXT text file, where information about the labels within the image is stored. The number of rows indicates the number of objects present in the image. In this case, each TXT file has only one row, since each frame contains only one interesting object, that is the nozzle.

A single row is structured by five parameters, written in the exact order indicated below:

<class_id> <x_center> <y_center> <width> <height>

where,

- **class_id**: is the numerical index of the object class. An index corresponds to each class and the indexing starts from 0. In this case the class is unique (nozzle class) and refers to index 0.

- (**x_center**, **y_center**): are the coordinates that represent the center of bounding box.

- (**width** x **height**): are respectively the width and height of the bounding box.

The box coordinates are normalized between zero and one as percentage of image dimensions, in such a way to obtain bounding boxes expressed in the normalized xywh format.

The next step is to partition the data into two folders, one containing the training data and the other containing the data to validate the model. This separation is done manually at the task level, so that each task belongs to one and only one of the two groups. From the set of videos provided, twelve tasks are obtained for the training set and five for the validation. An additional test folder contains 4 tasks, not available during the training phase, that are used to test the algorithm on new instances. Table 5.1 shows some statistics on the data folders:

Dataset	**#Tasks**	**#Frames**	**Frame Res.**
Training	12	4773	(1280 x 720) or (720 x1280)
Validation	5	1119	(1280 x 720) or (720 x1280)
Test	4	1242	(1280 x 720) or (720 x1280)

Table 5.1: Nozzle Detection dataset: split in Training, Validation and Test.

The final dataset structure is the following:

```
/
└── dataset_yolo
    ├── train
    │   ├── images
    │   │   ├── <task_number>_frame_<frame_number>.PNG
    │   │   └── <...>
    │   └── labels
    │       ├── <task_number>_frame_<frame_number>.txt
    │       └── <...>
    ├── val
    │   ├── images
    │   └── labels
    └── test
        ├── images
        └── labels
```

where, the frame is identified by a 6-digit number prefixed by the keyword *frame*, in turn prefixed to the *task_number*, which indicates the task to which it belongs. The *task_number* is a numerical index greater than or equal to 0 which is assigned progressively to each task inserted in the dataset. The correspondence between the numerical index and the identifier of the task (*video_id)*, is maintained inside a dictionary saved in TXT format.

The last point concerning the *Data Acquisition and Preparation* phase is to create a data configuration YAML file, containing details about the dataset used to train the model. The parameters to be defined in this file are the paths to which to retrieve training, validation and testing data, the number of classes and the names of the classes memorized as a dictionary. The dictionary contains for each class a mapping, in terms of key-value pairs, between the index (key) and the class (value). In this case the class is unique, meaning that the number of classes is put to 1 and there is a single mapping that is between the "nozzle" class and the zero index.

5.1.2 Training Phase - DL Model Training

As already explained in Section 4.3, the model used for the thesis work is a specific version of *YOLOv5*. The choice fell on *YOLOv5n*, which is the lighter version and allows the shortest inference time. The reason relies in the fact that one of the requirements for the *Anomaly Detection Model* is to run on edge devices, which have limited computational and memory resources.

YOLOv5 is available for all model sizes (small/medium/large/extra-large) in two types of architectures: P5 and P6. *YOLOv5-P6* models contain an additional

output layer P6, beyond the three output layers P3, P4, P5 available in the *YOLOv5-P5* versions. P6 models help in the detection of extra-large objects, and benefit especially from training at higher resolution (e.g. 1280 pixels). The nano model from the P5 architecture has been chosen, as the P6 versions are a bit slower and larger than the P5, which is not good in this case of study, where the goal is to optimize the performance in terms of inference time and also to have a model that has low memory occupation.

Transfer learning technique is taken into consideration to retrain the model on the new data, starting by considering the pretrained weights from a well-known large-scale dataset, without having to retrain the entire network from scratch. Freezing a number of initial layers speeds up the training phase, but can also lead to a reduction in the accuracy of the final model. The *YOLOv5* library suggests three alternative strategies [58]:

- **Freeze the backbone**: all the layers from 0 to 9, which correspond to the model backbone, are frozen. This is obtained by setting their gradients to zero before starting training.
- **Freeze all the layers**: a freeze list of layers (from 0 to 23) is created, containing the full model, except the final output convolution layers.
- **No freeze**: all layers are retrained (no layers are frozen).

The choice is to perform training starting by considering the pretrained weights of *Yolov5nP5* model on the COCO Dataset [43] (with an image size equal to 640) and then retraining all the layers on the new task. The training procedure is performed for 300 *epochs*, with a *batch size* equal to 16. Since for best results at inference time the library suggests testing at the same or lower resolution used during training, an empirical choice is made by varying the *image training size* following the grid below:

Image training size	1280 (original resolution)	640	320	128	96

and searching for the minimum resolution that allows the fastest training and inference time, while maintaining good performance in terms of detection capabilities. The final choice fell on a size equal to 128 which gives satisfactory results despite the low resolution. This is probably due to the fact that the task being analyzed is quite simple, as the detection of a single object, the nozzle, is carried out by keeping a fixed camera frame and a distance such as to allow a sufficiently close view (extra small objects are not present). Images are resized to 128 on the long side, while the short side is handled consequentially, preserving the original aspect ratio. Grey padding technique is applied to fullfill stride constraints.

To overcome instabilities in the early training, a warmup phase [59] is added, which allows to slowly ramp up model changes. The number of *warmup epochs* is

set to 3, while the corresponding *warmup momentum* and *warmup bias lr*, which are the ones that ramp to the default values over the warmup epochs, are set respectively to 0.8 and 0.1.

YOLOv5 loss is a compound loss, which consists of three parts: classes loss (L_{cls}), objectness loss (L_{obj}), which use the BCE loss [60] and location loss (L_{loc}), which uses the CIoU loss (see Figure 5.4). Loss gains are applied to scale and balance loss contributions:

$$Loss = 0.3 * L_{cls} + 0.7 * L_{obj} + 0.05 * L_{loc}$$

where, the objectness loss is weighted differently on the three prediction layers (P3, P4, P5):

$$L_{obj} = 4.0 * L_{obj}^{small} + 1.0 * L_{obj}^{medium} + 0.4 * L_{obj}^{large}$$

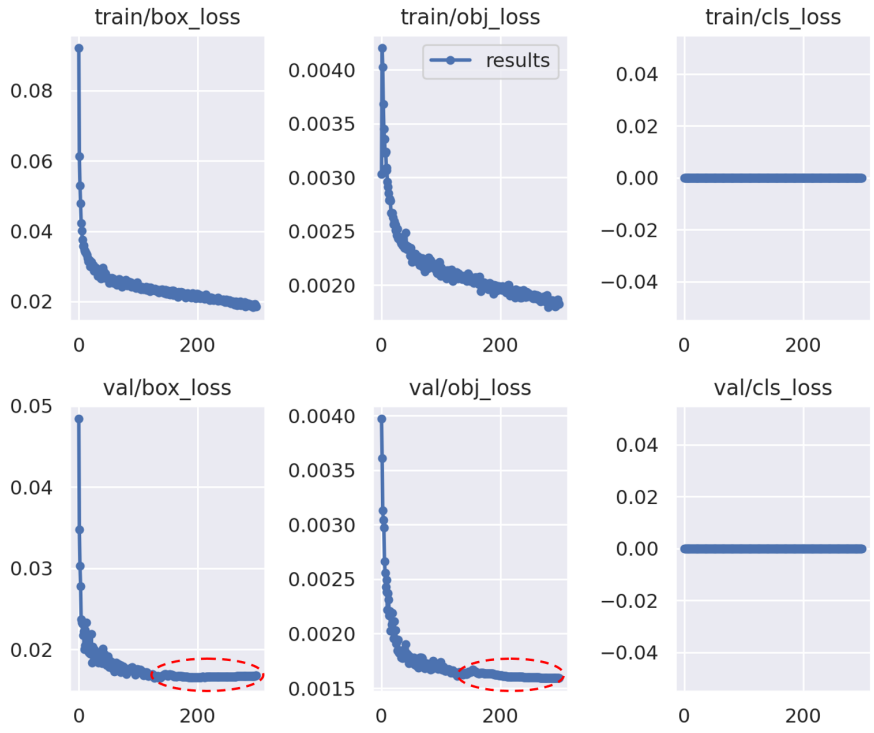

Figure 5.4: YOLOv5 loss contributions. The contribution of L_{cls} is 0 since the model is one class (Nozzle class).The red circles highlight the epoch from which some degree of overfitting starts occurring.

The optimizer chosen for stochastic optimization is Stochastic Gradient Descent (SGD) [61], with Nesterov Momentum (*momentum factor* = 0.937). The model is regularized by adding a penalty, called regularizer, to the cost function. The

parameter norm penalty applied is the L2 norm, commonly known as *weight decay*, which is set to the value of 0.0005. Three parameter groups are created and passed to the optimizer of the model. In this way, different hyperparameters can be applied to different parts of the model. In detail, Batch Normalization (BN) layers are separated into an independent parameter group to remove weight decay from applying to them, while Bias layers are added to an independent parameter group to apply higher learning rate during warmup. The instantiation of three parameters groups leads to the generation of the three learning rates shown on Figure 5.5.

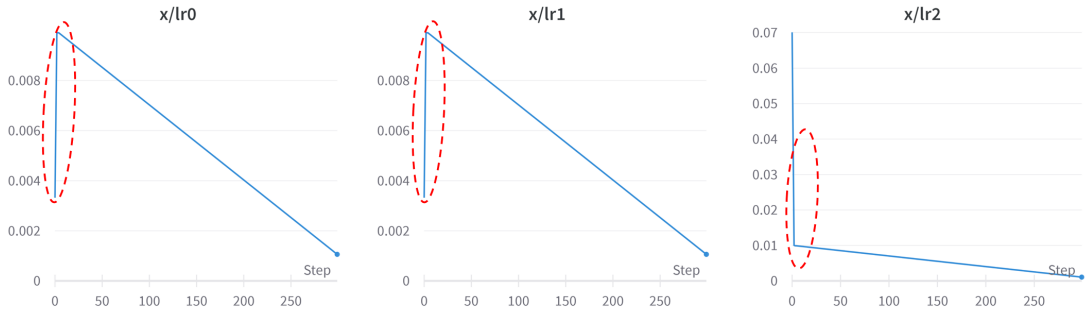

Figure 5.5: Learning rates. In red is highlighted the warmup phase.

The learning rate scheduler is designed to fall to a minimum value on the final epoch for best training results. The learning rate of each parameter group is set at each epoch to the initial lr0, times a given function, which computes a multiplicative factor given an integer parameter epoch. The equations describing this procedure are the following:

$$lambda\_function = (1 - x/epochs) * (1.0 - lrf) + lrf \tag{5.1}$$

$$lr\_epoch = lr0 * lambda\_function(x) \tag{5.2}$$

where x is the given epoch, $epochs$ is the number of total training epochs, $lr0$ and lrf are the learning rates, set respectively to the values of 0.01 and 0.1.

Since it is impossible to capture all possible real-world scenarios that a model can understand, it can be extremely useful to create alterations in the training data in order to help the model better generalize: this is the concept of Data Augmentation, which is a technique widely applied in computer vision, especially when dealing with small datasets that are easily prone to overfit. In practice, Data Augmentation is applied to make transformations to the original training data, in such a way to create synthetic images, that allow to expose the model to a greater extend of semantic variation than the training set in isolation. Within each training batch, *YOLOv5* passes training data through a data loader, which augments data

online. The data loader makes use of three main types of augmentations: color space adjustments, scaling and mosaic data augmentation.

Figure 5.6: Mosaics data augmentation.

The values of the hyperparameters used for data augmentation are listed below:

- **fliplr**: 0.5, flip image left and right randomly (probability).
- **flipud**: 0.0, flip image up and down randomly (probability).
- **hsv_h**: 0.015, image HSV-Hue augmentation (fraction) to randomly alter the colour channels of an input image. This technique is useful to ensure the model is not memorizing a given object or scene colours, helping to consider the edges and shape of objects rather than only the colours.
- **hsv_s**: 0.7, image HSV-Saturation augmentation (fraction). Adjusting the saturation level of an image can help the model to obtain better performances when colors in the wild change (e.g. different white-balance or different lighting setting). In this case of study, the model runs in a controlled environment, where there should be no major changes in color and light effects.
- **hsv_v**: 0.4, image HSV-Value augmentation (fraction).
- **mixup**: 0.1, image mixup (probability), which consists in superimposing an image over another one.

- **mosaic**: 1.0, image mosaic (probability). It is used to increase model accuracy by creating a new image from combination of multiple images, and then use the new created image for training (see Figure 5.6). Specifically, four images are combined into four patches of a random ratio. This technique helps in solving the "small object problem" (meaning the difficulty in detecting small objects with respect to the large ones), identifying objects at a smaller scale than usual.

- **perspective**: 0.0, image perspective (+/- fraction), range 0-0.001.

- **scale**: 0.9, image scale (+/- gain).

- **shear**: 0.0, image shear (+/- deg).

- **translate**: 0.1, image translation (+/- fraction).

- **degrees**: 0.0, image rotation (+/- deg).

This phase produces as output the trained Nozzle Detection model (see Figure 5.1), which consists in a checkpoint for inference containing the best weights found at a certain epoch during the training process. Concerning the name extension, the file is saved using the PyTorch convention ".pt".

5.2 ABN classification

The main objective of the Machine Learning (ML) pipeline, related to ABN Classification, is to classify for each frame the area below the nozzle, in such a way to determine, as soon as possible, if a defect is occurring during the printing of a 3D lamp. As well as for Nozzle Detection, also in this case two distinct phases are realized sequentially: the *Training Phase* of the classification model and the *Inference Phase*, where the model is fed with new data to test. Figure 5.7 gives an overview of the complete flow for the *ABN Classification - ML Pipeline*.

The *ABN Classification - Training Phase* is divided in two main parts which are executed sequentially. The first part is dedicated to *Data Acquisition and Preparation*, showing how the lighting company's 3D Printing Defects dataset is created. The second part focuses instead on the implementation of the *DL Model Training*, which is applied on this specific task.

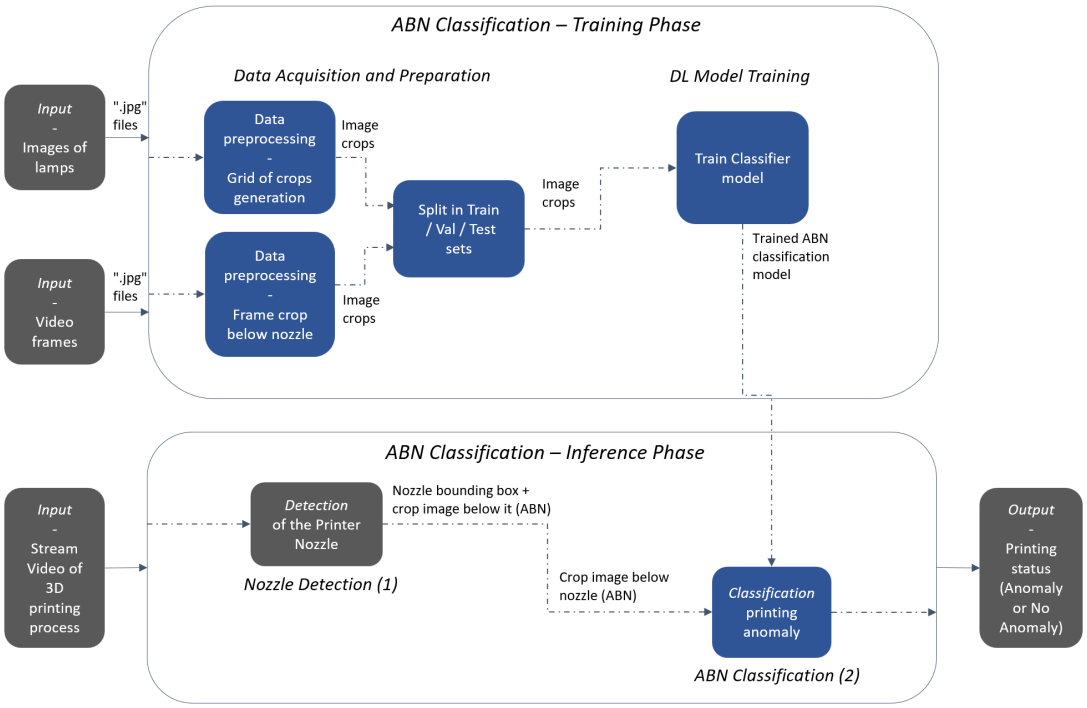

Figure 5.7: ABN Classification - ML Pipeline: the part relating to ABN Classification (training + inference) is highlighted in blue.

5.2.1 Training Phase - Data acquisition and Preparation

As well as for Nozzle Detection, also the ABN Classification requires a preliminary phase, in order to create the 3D Printing Defects dataset, which consists of square image crops showing possible defects in the production of lamps. Since the company's availability of printing processes videos was difficult to obtain, as a video surveillance process was not fully automated in their plants at the time of model development, in order to create a good representative set of anomalies, a set of pictures of printed lamps was given, from which interesting defects were extracted. In practice, the data used to train the model consist of:

- **Video files of 3D printing processes**: which coincide with the ones used to build the dataset for the detection of the nozzles (see Section 5.1). This time, starting from the 21 tasks received by the lighting company, 15 are used for training, 2 for validation and 4 for testing. The video frames, from this set of MP4 files, are extracted and processed in such a way to generate for each of them a crop by cutting a portion below the nozzle, as shown in Figure 5.8. The choice is to keep a small part of the nozzle in the crop, to better frame anomalies such as nozzle blobs.

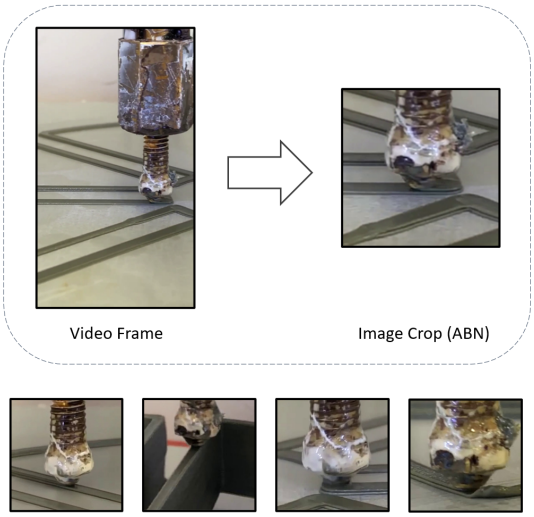

Figure 5.8: Examples of image crops (ABN) generated.

This process includes two automatic steps, one for the crops extraction and one for the elimination of those too similar to each other. The elimination step is carried out to avoid overfitting during the training process, by specifying as an input parameter to the script the step to which discarding is performed. All the generated crops are squared and with a resolution of 224 x 224 pixels.

- **Pictures of printed lamps which contain defects**: which are a source of data in addition to videos. From this set of images, 173 are used for the training set, while 68 for the validation set. All the crops deriving from a specific picture of lamp are inserted in one and only one of the two data splits. It is important to highlight that those pictures do not include all those type of anomalies that are strictly related to the printing process (e.g. nozzle blob, first layer issues) and therefore cannot be used as the sole source of data. Figure 5.9 shows pictures of lamps, some of which containing defects.

  The main technical steps for the crops generation include:

  - The automatic generation of a grid of crops (224 x 224 pixels) for each image showing a lamp.
  - A manual step to remove "not useful" crops generated.
  - A manual step to crop any anomaly not detected in the automatic step, plus an "intelligent" automatic resizing to 224 x 224 pixels. The script, to perform the resizing, brings the height to 224 (if height <= width) keeping the aspect ratio and then performs a crop on the width to bring

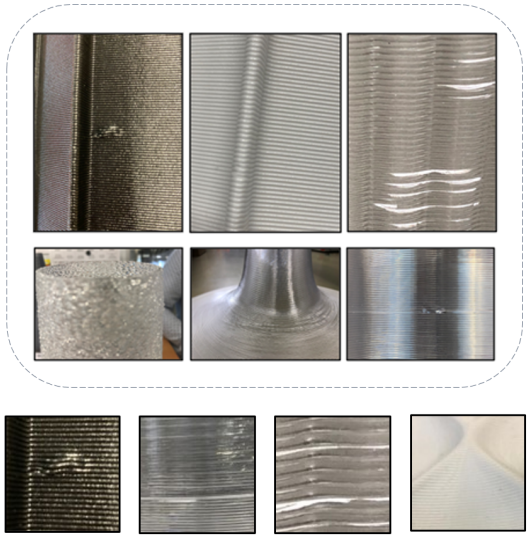

Figure 5.9: Examples of printed lamps images and generated crops.

the image to the desired dimension. If width > height the same operation is performed considering the inverted dimensions.

The following pseudocode reports the main steps to implement the automatic crop generation and the "intelligent" resize:

```
# ====== Script_1: Crop Images

target_dim = 224
width = 0
height = 0

while height + target_dim <= image_height:
    while width + target_dim <= image_width:
        original_image.cropAndSave(width,
                                   height,
                                   width + target_dim,
                                   height + target_dim)
        width += target_dim

    #Get last crop from current row
    if width - target_dim < image_width:
        width = image_width - target_dim
        original_image.cropAndSave(width,
                                   height,
                                   target_dim,
```

```
                                        target_dim)
        width = 0
        height += target_dim

    #Get last crops from current height
    if height-target_dim < image_height:
        height = image_height - target_dim
          while width+target_dim <= image_width:
            original_image.cropAndSave(width,
                                        height,
                                        target_dim,
                                        target_dim)
            width += target_dim

        if width - target_dim < image_width:
            width = image_width - target_dim
            original_image.cropAndSave(width,
                                        height,
                                        target_dim,
                                        target_dim)

    # ===== Script_2: Intelligent Resize

    target_dim = 224

    for image in frames_dir:

      if image_height <= image_width:
        image_resized = image.resize(height = target_dim)
        image_cropped = image_resized.crop(size = target_dim)

      else:
        image_resized = image.resize(width = target_dim)
        image_cropped = image_resized.crop(size = target_dim)

      image_cropped.save_to_disk(resized_frames_dir)
```

Once the image crops from both sources of data are generated, a manual step is required to divide them into two folders: "anomaly" and "no_anomaly". Each image crop follows the naming convention:

<task_name/image_name>_<crop_number>

where,

- task_name/image_name: indicates if the crop is coming from a video frame or from a picture of lamp.
- crop_number: is simply an identifier of the crop, used to distinguish different crops belonging to the same video task/picture of lamp.

Below is the directory structure where the final dataset is stored, ready to be used for training the deep learning model:

```
/
└── class_dataset
    ├── train
    │   ├── anomaly
    │   └── no_anomaly
    │       ├── <task_name/image_name>_<crop_number>.JPG
    │       └── <...>
    ├── val
    └── test
```

From Table 5.2 it is possible to observe some statistics on the two obtained data folders. The #Crops column underlines how the distribution of data is skewed

Dataset	Class	#Crops	Crop resolution
Training	no_anomaly	2298	(224x224)
Training	anomaly	763	(224x224)
Validation	no_anomaly	504	(224x224)
Validation	anomaly	180	(224x224)
Test	no_anomaly	409	(224x224)
Test	anomaly	55	(224x224)

Table 5.2: 3D Printing Defect dataset: split in Training, Validation and Test.

towards the "no_anomaly" class, for both training, validation and test sets. This may result in ML models with poor predictive performance, specifically for the minority class, since the model may tend to classify points as belonging to the majority class. In the next section some strategies will be adopted to deal with the data imbalance problem.

5.2.2 Training Phase - DL Model Training

To choose the most appropriate image classifier for the task being analyzed, some of the most popular models in literature have been tested. As already introduced in Section 4.4, the final decision fell on *EfficientNet-Lite0*, which is the smallest Lite version from *EfficientNet*, suitable for mobile and edge devices like Raspberry Pi. The machine learning library adopted for this phase is Tensorflow [62] and in

particular the Keras trainable API [63], which allows to easily implement transfer learning and fine-tuning workflows.

The transfer learning strategy applied follows the main steps highlighted in Section 4.4, where the base model is instantiated and the pretrained weights from ImageNet are loaded, by excluding the model head that is instead recreated for the new task. In practice, two training phases are performed sequentially: a *Feature Extraction* phase, where previous knowledge is used to extract significant features from new data samples and a *Fine Tuning* phase, which is used to fine-tune the high-level features in the base model to the specific task.

Both phases, depicted above, adopt the following choices in training design: Stochastic Gradient Descent (SGD) is used as stochastic optimization technique, with a *momentum* equal to 0.98. The Categorical Cross Entropy loss is chosen as target function to minimize. A *dropout rate* equal to 0.2 is applied to avoid overfitting the training data. An L2 regularization term is added in the last fully connected layer of the network, applying it to the weights and setting the regularization factor to 0.0001. Since the dataset is imbalanced, class weights strategy is applied during training to weight the loss function in such a way to pay more attention to the class that is less representative, which in this case is the "anomaly" class. The mapping between the class indices and the weights is the following:

$$weight\_0 = (1/neg) * (total/2.0)$$
$$weight\_1 = (1/pos) * (total/2.0)$$

where, *total* is the total number of training samples , *neg/pos* is the amount of negative/positive samples and labels 0/1 indicate the positive ("anomaly")/negative class ("no_anomaly"). The choice is to use a *batch size* of 128, to increase the possibility of having positive samples in a batch, compared to the case in which the default value of 32 is used. An empirical search is performed to find the smallest image size that allows for the fastest inference time, while still maintaining good classification ability. The following grid shows the different values tested:

Image size	224 (original resolution)	128	96	32

The final choice fell on a *image size* equal to 96 which gives satisfactory results despite the low resolution.

The parameters that differ for the two phases are listed below:

- **Feature extraction phase**: training is performed for 100 *epochs*, by keeping all the backbone layers frozen. The *learning rate* is set to $2 * 10^{-3}$.
- **Fine tuning phase**: to further improve performance, the best weights found on the previous phase are loaded and new training is performed just for 15

epochs. This time, a part of the backbone is unfrozen, which consists in the last top 10 layers and re-training is performed with a very low *learning rate* equal to $1 * 10^{-7}$. It is important to note that when the base model is unfrozen for fine-tuning, Batch Normalization layers are kept in inference mode, to avoid destroying the mean and variance statistics that the model had previously learned.

The *DL Training phase* produces as output the trained ABN Classification model (see Figure 5.7), which consists in the best model found to use for inference, saved in the TensorFlow SavedModel format. SavedModel is the default file format used in TF2.x to save an entire model. This method is useful for deploying (e.g. with TFLite), since it does not require the original model building code to execute.

5.3 Anomaly detection model

This section explains, at a lower level with respect to Chapter 4, how the *Nozzle Detection (1)* and the *ABN Classification (2)* are integrated at inference time to create a unique pipeline that performs anomaly detection (see Figure 4.1). Technically speaking, the main contribution to this phase consists in a modification to the "detect.py" script available in the *YOLOv5* repository, in such a way to add a second stage classifier after the detection stage. All the code is written in Python v3.10.2 and the main python libraries used are: OpenCV-Python and Pillow for image processing, Pytorch and Tensorflow for machine learning application.

The base directory tree structure of the Anomaly Detection project is the following:

```
/
└── package
    ├── aida
    │   ├── classifier
    │   │   └── classifier.py
    │   ├── detector
    │   │   ├── yolov5
    │   │   │   └── detect.py
    │   │   └── detector.py
    │   ├── image_preprocessing
    │   │   └── class_image.py
    │   ├── models
    │   ├── resources
    │   └── utils.py
    └── anomaly_detection.py
```

Starting from the bottom of the project structure it is possible to observe the "anomaly_detection.py" script. This file corresponds to the main program which basically takes care of loading the configuration file for that specific run and instantiating the two classes: *Detector* and *Classifier*, whose definitions can be found respectively in "detector.py" and "classifier.py". Its last action is to "activate" the anomaly detection by invoking the *detect* method on the detector object previously instantiated, as following:

```
detector.detect(classifier, configs)
```

where, the parameters are the classifier object and the configuration values passed as python dictionary. The *detect* method of the *Detector* class is nothing more than a wrapper which invokes the "detect.py" script, passing the proper parameters for that run as described below:

- **class_model**: the classifier object, whose task is to manage the ABN classification during inference phase.
- **imgsz**: the inference input image size (height, width) for the nozzle detection phase.
- **source**: the data source. YOLOv5 can deal with different types of sources. For this application a video stream (RTSP format) is required, but different inputs have been tried for debugging purposes: video/image formats and webcam.
- **weights**: the path to the nozzle detection model weights.
- **save_crop**: boolean indicating if the image crops (ABN), showing the presence or absence of anomalies, are saved on disk or not.
- **view_img**: boolean indicating if the results are shown on video while performing anomaly detection or not.
- **class_below_nozzle**: boolean which allows to decide if ABN Classification is performed or not (add or discard the second stage classifier).
- **nosave**: boolean indicating if the Anomaly Detection results (images/videos) are saved on disk or not.
- **project**: the path to the directory where the results are stored.
- **name**: the run name, which corresponds to the couple (Video Stream ID, timestamp).

- **max_det**: the maximum number of detections per image. In this case it is set to 1 since for each image there is always a single instance of the nozzle, which is the one the algorithm is trying to detect.
- **save_csv**: boolean indicating if the Anomaly Detection results are saved or not in a CSV file. Section 4.5 describes in detail all the fields that are stored in it.

The main integration in "detect.py" consists in adding inside the main loop, devoted to slide through the dataset entries (e.g. frames in case of a video stream), a portion of code specific to ABN classification.

Specifically, for each data entry, a inference run is computed to detect the nozzle inside the input image. Given the bounding box (bb) prediction obtained by *YOLO*, a method is invoked to retrieve the image crop bb coordinates associated to the area below the nozzle (ABN). The pseudocode of the method is the following:

```
def get_image_crop_bb_below_nozzle(xyxy,
                                   gain,
                                   pad):

  b=xyxy2xywh(xyxy)  # boxes

  sb = xyxy.clone()
  sb[:, 0]=b[:, 0]    # x center
  sb[:, 1]=b[:, 1] + (b[:, 3]/2) + 10 # y center
  sb[:, 2]=b[:, 2]   # width
  sb[:, 3]=b[:, 2]   # height

  sb[:, 2:]=sb[:, 2:] * gain + pad  # box wh * gain + pad
  xyxy=xywh2xyxy(sb).long()

  for element in xyxy:
    xyxy_.append(torch.tensor(element))

  return xyxy_, xyxy
```

where, xyxy are the bb coordinates surrounding the nozzle and detected by the detection algorithm. The image crop bb coordinates, returned by the method, are computed as follows: the width and height coincide with the width of the nozzle bb, the x coordinate of the center is the same as that of the nozzle bb, while the y coordinate is translated downwards with respect to that of the nozzle bb by a factor chosen experimentally ($y\_center\_image\_crop = y\_center\_bb/2 + 10$), which allows to frame a possible anomaly under the nozzle, keeping a small portion end of the nozzle inside the cropped image.

At this point, the method *get_image_below_nozzle* is invoked to extract the associated crop, as a cv2 image:

```
def get_image_below_nozzle(xyxy,
                           im,
                           BGR=False):

    clip_coords(xyxy, im.shape)
    crop=im[int(xyxy[0, 1]):int(xyxy[0, 3]),
            int(xyxy[0, 0]):int(xyxy[0, 2]),
            ::(1 if BGR else -1)]
    cv2_crop=cv2.cvtColor(crop, cv2.COLOR_BGR2RGB)
    return cv2_crop
```

where, *xyxy* is a variable containing the image crop bb coordinates computed in the previous step and *im* is the entire input image.

Given the *cv2_crop*, a CLImage object, from the *ClassImage* class, is created. *ClassImage* is a wrapper class for cv2 images, containing utility functions such as an "intelligent" image resizing, which allows to obtain the desired dimension as input to the classification model. The desired dimension indicates the image size with which the classification model was trained. This information is contained in the *class_model* object from the *Classifier* class, which was instantiated at the beginning in the main program and passed as parameter to the "detect.py".

The *Classifier* class takes care of managing the inference part related to ABN classification. Once the image crop is retrieved and properly resized, a method from this class, devoted to classification evaluation, is invoked on the *class_model* object:

```
preds, predicted_label, softmax_value = class_model.evaluate_image(
                                                    cropped_img)
```

The method simply computes the classification inference run on the image crop passed as parameter (*cropped_img*) and returns the results of this prediction as a triplet: confidence scores for each class (*preds*), *softmax_value* with the highest value and its associated *predicted_label* (0/1). The Rolling Prediction Average technique is applied to mitigate the so-called "prediction flickering" problem, which consists in a frequent label changing among classes during prediction. Flickering occurs because a simple image classification algorithm is used, which does not take into account the semantic correlation between frames, to solve the video classification problem. Rolling Averaging is generally used on time series data to decrease fluctuations in the output and its principle can be used to help reducing the flickering effect in video classification. In this case, the technique is applied in such

a way to maintain a queue of the last q_size=30 predictions, compute the average on them and then choose as predicted label the one with largest corresponding probability.

At this point, the Anomaly Detection prediction on a single data entry is completed and the results as bounding boxes and confidences scores are annotated on the original input image, by invoking the following method from the YOLOv5 *Annotator* class:

```
def box_label(self,
              box,
              s_box = [],
              label='',
              s_label='',
              color_bb=(128, 128, 128),
              color_img_crop=(128, 128, 128),
              txt_color=(255, 255, 255), show_crop=False):

  # Add one xyxy nozzle box to image with label
  ...

  if show_crop:
    s_p1, s_p2=(int(s_box[0]), int(s_box[1])),
               (int(s_box[2]), int(s_box[3]))

    cv2.rectangle(self.im, s_p1, s_p2,
                  color_img_crop,
                  thickness=self.lw,
                  lineType=cv2.LINE_AA)

    if s_label:
      tf=max(self.lw - 1, 1)  # font thickness
      # text width, height
      s_w, s_h=cv2.getTextSize(s_label,
                               0,
                               fontScale=self.lw/3, thickness=tf)[0]

      outside=s_p1[1]-s_h-3 >= 0  # label fits outside box
      s_p2=s_p1[0]+s_w, s_p1[1]-s_h-3 if outside else s_p1[1]+s_h+3

      cv2.rectangle(self.im,
                    s_p1,
                    s_p2,

                    color_img_crop,
                    -1,
                    cv2.LINE_AA)  # filled
```

```
        cv2.putText(self.im,
                    s_label,
                    (s_p1[0], s_p1[1]-2 if outside else s_p1[1]+s_h+2),
                    0,
                    self.lw / 3,
                    txt_color,
                    thickness=tf,
                    lineType=cv2.LINE_AA)
```

The method is modified with respect to the original to add the annotations for the image crop. More in detail, the *box/label* and *s_box/s_label* parameters are respectively the nozzle bb/label and the ABN bb/label, while *color_bb* is the color associated to the nozzle bb and *color_img_crop* is the color for the ABN box, that differs according to the classification predicted label. The parameter *show_crop* indicates if the ABN box is shown on the output image or not.

Lastly, the Anomaly Detection results, for the input data entry, are saved in a CSV file, as a single line, following the output format described in Section 4.5. The whole procedure, added inside the main loop, is repeated for each data entry, until all the input source is processed. The pseudocode containing all the steps explained above for performing Anomaly Detection on a single video frame is the following:

```
# ======= Integration inside "detect.py" by YOLOv5
# ======= Second stage classifier: classify Area Below Nozzle (ABN)
...
q_size=30
Q = deque(maxlen=q_size)
anomaly_dict ={"0": "no anomaly", "1": "anomaly"}
if save_csv:
  csv_file_path = str(save_dir / 'results.csv')
  f_csv = open(csv_file_path, 'w', 1, newline='')
  writer = csv.writer(f_csv)
...

for path, im, im0s, vid_cap, s in dataset:
  ...
  pred = model(im, augment=augment, visualize=visualize)
  ...
  pred = non_max_suppression(pred,
                             conf_thres,
                             iou_thres,
                             classes,
                             agnostic_nms,
                             max_det=max_det)
  ...
```

```
for i, det in enumerate(pred):  # per image
  ...
  # Write results
  for *xyxy, conf, cls in reversed(det):
    ...
    if class_below_nozzle:
      s_xyxy_, s_xyxy=get_image_crop_bb_below_nozzle(xyxy)
      cv2_crop=get_image_below_nozzle(s_xyxy, imc)
      CLImage=ClassImage(cv2_crop, "crop")
      target_size=class_model.img_size

      # Apply "intelligent" resize
      if (CLImage.height < CLImage.width):
        resized=CLImage.image_resize(height=target_size)
        cropped_img=resized.crop(target_img_size=target_size)

      elif (CLImage.height > CLImage.width):
        resized=CLImage.image_resize(width = target_size)
        cropped_img=resized.crop(target_img_size=target_size)

      else:
        cropped_img=CLImage.image_resize(height=target_size,
                                         width=target_size)

      if show_crop:
        preds,predicted_label,softmax_value=class_model.
                                            evaluate_image(
                                              cropped_img)
        Q.append(preds)
        results=np.array(Q).mean(axis=0)
        softmax_value=np.max(results)
        predicted_label=np.argmax(results)

        s_label = "%s %.2f" % (anomaly_dict[str(predicted_label)],
                               softmax_value)

      class_colors_v = [(0,255,0), (255,0,0)]
      if save_img or view_img:
        annotator.box_label(box=xyxy,
                            s_box=s_xyxy_,
                            label=label,
                            s_label=s_label,
                            color_bb=(63,202,233),
                            color_img_crop=
                            class_colors_v[predicted_label],
                            show_crop=show_crop)
    ...
    # Write to file
    if save_csv:
```

```
            # normalized xywh
            xywh=(xyxy2xywh(torch.tensor(xyxy).view(1, 4))/gn)
                                                .view(-1).tolist()
            now=datetime.now()
            date_time=now.isoformat(timespec="milliseconds")

            if class_below_nozzle:
              # normalized xywh
              s_xyxy_=(xyxy2xywh(torch.tensor(s_xyxy_).view(1, 4))/gn)
                                                    .view(-1).tolist()
              # label format
              data=[date_time,
                    anomaly_dict[str(predicted_label)],
                    int(predicted_label),
                    round(float(conf),2),
                    round(float(softmax_value),2),
                    str({*xywh}),
                    str({*s_xyxy_})]
              writer.writerow(data)
            ...

if save_csv:
  f_csv.close()
...
```

Chapter 6

Results

This chapter describes the performance evaluation for the *Nozzle Detection* and *ABN Classification* models respectively, which were trained using data from a lighting company, following the data acquisition/preparation steps and the training process explained in Chapter 5. Sections 6.1 and 6.2 are dedicated to the evaluation phase for the two tasks, in which the specific metrics used to estimate the performance of the model are underlined. Section 6.3 shows the performances of the trained models in terms of inference time and how they can be optimized to run on very low power edge devices.

6.1 Evaluation - Nozzle Detection

Concerning *Nozzle Detection*, a validation set composed by 1119 frames is used to monitor the model performance during the training phase and to decide which one to choose as the best model. Next, a test set with 1242 frames is applied to provide an unbiased sense of model effectiveness, seeing how the model behaves on new data. Both datasets contain real case frames, depicting the nozzle during the printing process to produce a lamp and their specifications can be retrieved in Table 5.1.

6.1.1 Metrics definition

The metrics used to validate the model are the ones commonly applied on object detections tasks [64]. The subsections below report a synthetic definition for each computed metric on the task under analysis.

Precision and Recall

In the context of object detection, the concepts of *True Positives* (TP), *False Positives* (FP) and *False Negatives* (FN) can be described as:

- *True Positive*: a correct detection, meaning a $IoU \geq threshold$;
- *False Positive*: a wrong detection, meaning a $IoU < threshold$;
- *False Negative*: a ground truth that the model was not able to detect;

where, IoU is the Intersection over the Union of the bounding box for the ground truth and the predicted bounding box (see Section 3.3) and $threshold$ is a value for the IoU which determines if the object detection is valid or not. IoU ranges between 0 and 1, where 0 means no overlap and 1 refers to a perfect match. Changing the IoU threshold changes whether a prediction is considered a TP or FP. When the threshold is near to 1 the metric is more restrictive as it needs practically perfect detections, when the threshold is set near to 0 the metric is more flexible instead, as it considers detections even boxes with small overlaps. Common values for the IoU threshold are 0.50 and 0.75.

Precision and *Recall* are the first two metrics used to evaluate an object detection model and are computed using TP, FP, FN as defined above. *Precision* indicates the model ability in identifying only the relevant objects. In other words, considering the elements predicted as a positive, it counts the percentage that is correct as follows:

$$Precision = \frac{TP}{TP+FP} = \frac{TP}{All\ detections}$$

Recall instead measures the ability of a model in finding all the relevant cases. It can be interpreted as the percentage of actually positive elements the model succeeds to find, among all relevant ground truths and its formulation is given by:

$$Recall = \frac{TP}{TP+FN} = \frac{TP}{Total\ ground\ truth}$$

Basically, having defined the two metrics above, an object detection algorithm, to be considered good, should be able to detect all ground truth objects (FN = 0 = high *Recall*), while finding only relevant objects (FP = 0 = high *Precision*).

Average Precision

Average Precision (AP) refers to the Area Under the *Precision-Recall Curve* (AUC-PR) and is one of the main metric used to estimate an object detection model. The PR curve shows the tradeoff between *Precision* and *Recall* for different confidence

score thresholds, where the confidence score is a measure of how confident the detector is about the prediction. A high area under the curve indicates a good model as both precision and recall are high, meaning that the model manages to return many of the positive results, as well as returning accurate results. The AP is calculated at a given α IoU threshold and can be formally defined as follows:

$$AP@\alpha = \int_0^1 p(r)dr$$

where, the values of AP range between 0 and 1. Since PR curves are often zigzag, a preprocessing step is performed to remove this behaviour, in such a way to obtain a monotonic trend. Mathematically, the precision value for recall r is replaced with the maximum precision for any recall $\geq r$ as follows:

$$p_{interp}(r) = \underset{\tilde{r} \geq r}{max}\ p(\tilde{r})$$

To compute the Riemann integral of the PR curve, two main approaches in terms of interpolation can be applied:

- **N-point interpolation**: AP is computed by averaging the precision at a set of N reference recall values equally spaced in the interval [0,1]:

  $$AP@\alpha = \frac{1}{N} \sum_{r \in \{0, 0.1, \ldots 1.0\}} p_{interp}(r)$$

  Common values are N=101 which is used in the COCO Detection Challenge (Bounding Box) competition and N=11 which was used in the Pascal Visual Object Classes Challenge before being changed in all-point interpolation method.

- **All-point interpolation**: in this case interpolation is done for all the recall values, computed considering all the confidence levels:

  $$AP@\alpha = \sum_i (r_{i+1} - r_i)\ p_{interp}(r_{i+1})$$

Mean Average Precision

Independently from the interpolation method used, AP is obtained separately for each class. *Mean Average Precision* (mAP) provides a unique metric able to indicate the exactness of the detections across all classes. The computation is simply obtained by averaging the AP values among all the C classes:

$$mAP@\alpha = \frac{1}{C} \sum_{i=1}^{C} AP_i$$

6.1.2 Evaluation phase

Considering the *DL Model Training* phase for *Nozzle Detection* (see Section 5.1.2), the best model found is at epoch 137 and is chosen by considering the $fitness$ as criterion, which is computed as the weighted sum of the object detection metrics:

$$fitness = \omega_1 * P + \omega_2 * R + \omega_3 * mAP@0.5 + \omega_4 * mAP@0.5 : 0.95$$

In this case, the weights $\omega_{i=1,2,3,4}$ have the following values: $\omega_1 = 0.0$, $\omega_2 = 0.0$, $\omega_3 = 0.1$ and $\omega_4 = 0.9$; this means that the greatest weight is given to $mAP@0.5 : 0.95$, while P and R do not affect in the model choice. In detail, the metrics used in the $fitness$ formulation refer to:

- **P and R**: the *Precision* and *Recall* associated to the maximum F1 confidence threshold, which should give a good balance between the two metrics;
- **mAP@0.5**: the *Mean Average Precision* with IoU Threshold = 0.50;
- **mAP@0.5:0.95**: the average among all the mAP results, which are calculated over a range of 10 IoU thresholds (t=[0.5,0.55,...,0.95]);

where, for AP a 101-point interpolated definition is used in the calculation.

Table 6.1 shows the values for the object detection metrics, which refer to the best model found and which are computed on the validation and test datasets respectively. In general, all metrics show good results and remain stable in their values when calculated on new test data not used during the training phase. This means that the model is able to accomplish the task under analysis and can generalize to unseen data. Concerning *Precision* (P) the result on the test set is

Data	**P**	**R**	**mAP @0.5**	**mAP @0.5:0.95**
Validation	0.933	0.988	0.958	0.549
Test	0.969	0.935	0.976	0.514

Table 6.1

0.969. This indicates that the model is very "pure": out of all positive predictions (TP+FP), it finds a good number of TPs, meaning correct nozzle detections, minimizing FPs. As regards *Recall* (R) the result obtained on the test set is equal to 0.935, which means that the model is able to find a good percentage of TPs, among all given ground truths (TP+FN). P and R are chosen considering the maximum score for the metric F1, which is computed as the harmonic mean of

the two, giving both the same importance (see Section 6.2.1). Figure 6.1 shows the F1 curves constructed by considering different scores, each one evaluated on a different confidence value. For the test set, the confidence value that optimizes P and R is 0.48, to which the maximum value for F1 (0.95) corresponds. Note that in this case the label "all classes" is equal to the single class label "nozzle", since the detection model is trained to detect only one class. Both P and R are important

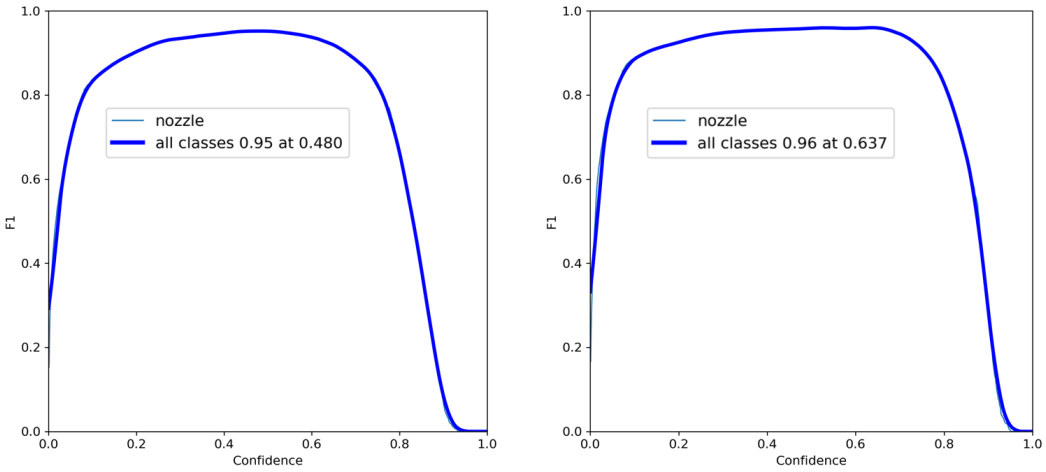

Figure 6.1: Nozzle Detection - F1 curves. Right: F1 curve generated from the Validation set. **Left**: F1 curve generated from the Test set.

to determine the goodness of the model, indeed if a model has high P but low R it means that it is able to predict samples as positive in a accurate way, but only few of them (high FN), conversely a model with high R and low P classifies correctly a high amount of positive samples, but makes a lot of incorrect predictions on the negative ones (high FP). *Average Precision* (AP) summarizes the PR curve, incorporating the trade-off between P and R for different thresholds, into a single metric and therefore is commonly used as standard metric in object detection to analyze the model "accuracy". The most challenging metric is mAP@0.5:0.95, which is computed by thresholding the bounding boxes at different IoUs. The result of this metric for this task remains stationary at a value approximately equal to 0.5 for both datasets, where the letter "m" within the acronym mAP indicates that its value is obtained by averaging over all object classes; it should be noted that in this case the mean is not calculated since this is a one-class problem. In the end, the table shows the mAP@0.5 with a fixed IoU threshold of 0.5. This is the main metric used in the Pascal VOC competition and in this case it obtains a good result on the test set equal to about 0.976. In Figure 6.2 the associated Precision-Recall curves can be seen, evaluated at 0.5 mAP. In particular, the one referring to the

test set is shown on the left. As with the F1 curves, also in this case the labels "all classes" and "nozzle" coincide, since there is only one class to be detected. Similar considerations concerning the metrics results can be done, looking at Table 6.1, for the row which highlights the results obtained on the validation set.

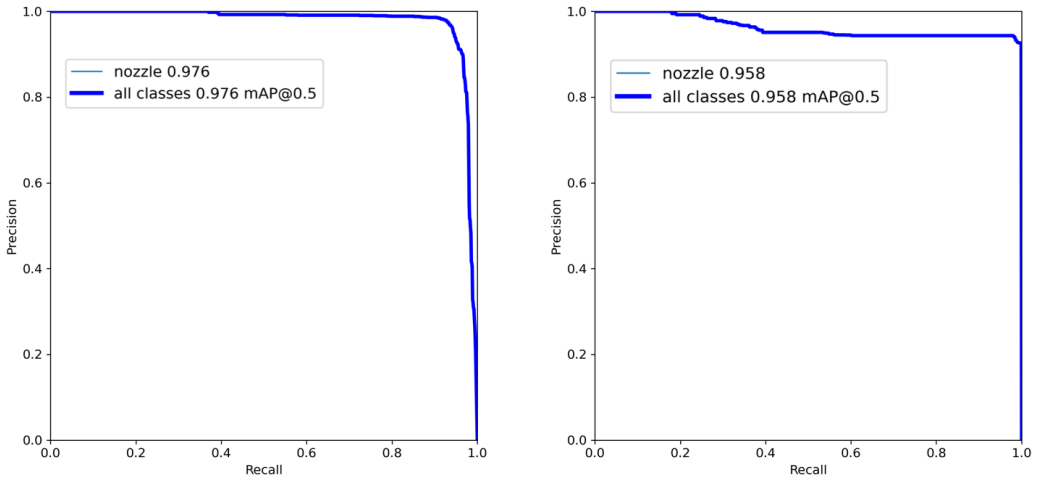

Figure 6.2: Nozzle Detection - PR curves. Right: PR curve generated from the Validation set. **Left**: PR curve generated from the Test set.

6.2 Evaluation - ABN Classification

As regards *ABN Classification*, a validation set composed by 684 image crops is used to monitor the model performance during the training phase and to choose the model with the best weights. Then, a test set with 464 unseen data is applied to provide an unbiased estimate of the model skill. Table 5.2 reports the details for these two datasets. As already explained in detail in Section 5.2.1, the 3D Printing Defects dataset is composed of images depicting the presence or absence of an anomaly. Some crops are generated extracting from a video frame the area below the nozzle, some others are obtained from already printed lamps. This second source of data is added to increase the variety of data (in shape, color, material) and to increase examples showing defects.

6.2.1 Metrics definition

In the context of classification, one of the most common metric to evaluate a model is the *Accuracy*, which can be described as the number of correct predictions over the total number of predictions. The classification task under analysis consists

in a binary classification, aiming at distinguish between the class "anomaly", also referred as the positive class and the class "no_anomaly", also referred as the negative class. In binary classification, the *Accuracy* metric can be expressed in terms of positives and negatives as follows:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

where, TP = True Positives, TN = True Negatives, FP = False Positives and FN = False Negatives.

However, since in this case of study the dataset is biased towards the "no anomaly" class, *Accuracy* is not a good choice as it can lead to misleading high results, even if the model performs poorly on the minority class, by simply assigning all the samples to the majority class. For this reason, the *F1* score, which takes into account not only the number of prediction errors but also the type of errors that are made, is considered as the main metric in the model evaluation and in the choosing of the best model. This metric is defined as the harmomic mean of *Precision* and *Recall*, where a score reaches its best value at 1 and worst at 0 and its formulation is given by:

$$F1 = 2 * \frac{Precision*Recall}{Precision+Recall}$$

F1 score gives the same weight to *Precision* (measure of exactness) and *Recall* (measure of completeness) and its value is high when both metrics are high.

6.2.2 Evaluation phase

As concerns the *DL Model Training* phase for *ABN Classification*, two consecutive phases are carried out as described in detail in Section 5.2.2: Feature Extraction and Fine Tuning. They are part of the entire Transfer Learning process to adapt the pretrained model to the 3D Printing Defects task and both consider the *F1* score as measure to monitor the best model choice.

Table 6.2 shows, both for feature extraction and for fine tuning, the results obtained on the validation set, using the best model found at a certain epoch. As for the feature extraction phase, in which the backbone is kept frozen, the 42nd training epoch out of 100 provides the best model in terms of F1-score. In general, the metrics give promising results, bearing in mind the difficulty of the task. More in detail, the *Accuracy* of the model takes on a value equal to 0.89. This metric measures the ratio of the number of correct classifications (TP+TN) to the total population (TP+TN+FP+FN), considering both "anomaly" (positive) and "no anomaly" (negative) instances. As mentioned in the previous section, *Accuracy* can be misleading when dealing with imbalanced datasets, which is why other metrics are also considered in the evaluation. The aim is to build a model which is

Training Phase	Epoch	Accuracy	F1-score	Precision	Recall
Feature Extraction	42	0.89	0.80	0.78	0.82
Fine Tuning	13	0.90	0.81	0.87	0.76

Table 6.2: Results on Validation set.

particularly skilled in detecting defects and therefore more sensitive to errors for the minority class than for the other class. For this reason, the following metrics are computed on the "anomaly" class, that is the one for which the model wants to optimize its predictive performances. Concerning *Precision* (P), it can be seen as the "accuracy" of minority class predictions. In other words, in this context it is computed as the ratio of correctly predicted anomalies (TP), divided by the total amount of samples predicted as anomalies (TP+FP) and it reaches on the validation set a value equal to 0.78. As regards *Recall* (R), it refers to the model ability to cover all positive cases, meaning that it quantifies the number of correct anomaly predictions (TP) made out of all anomaly samples in the data (TP+FN). The value obtained in this case for the metric is 0.82. Given the imbalanced data setting, both P and R are relevant in the sense that the goal is to maximize R (minimize FN), without damaging P too much. For this reason, F1 metric is used since it helps in combining P and R into a unique measure, in such a way to balance both contributions (see Section 6.2.1). In particular, for this task F1 obtains a score equal to 0.80 on the validation set. This is quite a good value, which reflects the good results obtained both on P and R. Similar considerations about the metrics can be done for the fine tuning phase, which is performed to "finetune" the higher level convolutional layers of the base model to the new task. The best model for this phase corresponds to the 13th epoch out of 15 in total and the results obtained on the validation set, using the best model, are overall slightly improved compared to the feature extraction phase, apart from the *Recall* which drops from 0.82 to 0.76.

At this point, the best model obtained at epoch 13, during the fine tuning phase, is used to assess the skill of the model on the test set, which results can be retrieved in Table 6.3. Also in this case, the metrics F1, P and R are computed on the minority class, corresponding to the label "anomaly". Compared to the values obtained on the validation, the final model obtains on the test set a similar F1 score, equal to 0.80, but with inverted contributions concerning P and R: indeed, while the P drops from 0.87 to 0.79, the R rises from 0.76 to 0.82. The confusion matrices in Figure 6.3 explain more in detail the individual scores for TP, FP, FN

Accuracy	F1-score	Precision	Recall
0.95	0.80	0.79	0.82

Table 6.3: Results on Test set.

and TN. In particular, the sub-figure on the right shows that out of 55 anomalies in the test set, 45 are correctly classified (TP) while the remaining 10 are FNs, meaning that the model wrongly categorizes them as normal behaviour. For this task it is important to minimize the FNs, since the objective is to be able to find all the anomalies without ideally losing any. The model can instead tolerate some FPs, which indicate a normal pattern for the printing process that is erroneously classified as an anomaly. In fact, when this happens, the model issues an incorrect warning that can simply be ignored by the 3D printing machine operator. The confusion matrix shows that, for the test set, out of 409 "normal" samples, 12 are incorrectly categorized as defects (FP). Similar considerations can be done for the confusion matrix generated from the validation set, taking into account that the overall model results more precise (higher P), but less complete in finding the relevant cases (lower R), than when evaluated on the test set.

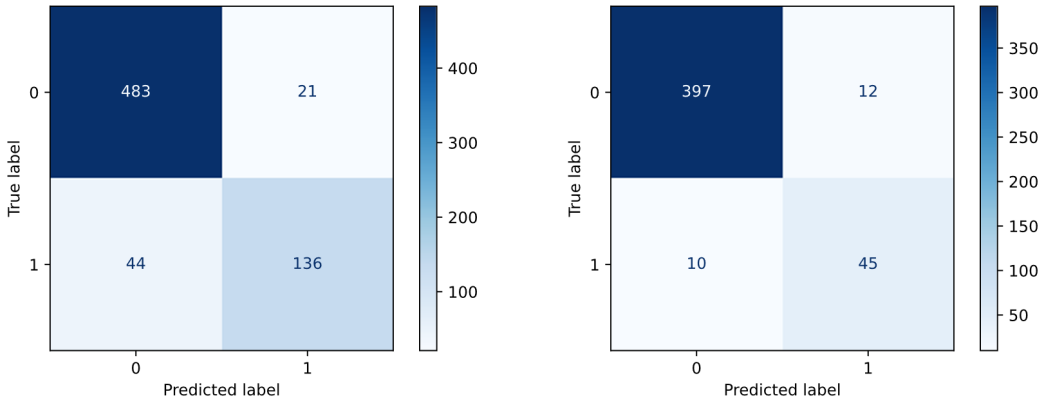

Figure 6.3: ABN Classification - Confusion matrices. Left: Confusion matrix generated from the Validation set. **Right**: Confusion matrix generated from the Test set.

From the results obtained and explained above, it is important to underline some aspects concerning the difficulty of the task. First of all, the cardinality of the data, for both training and validation/test phases, is not huge and fully representative of all the possible input designs and type of anomalies. Thus, to increase both the model ability in dealing with different inputs and to get a more

precise estimate of its goodness on new data, it would be advisable to have more data with different lamp designs, colours and defects. Secondly, the imbalance problem poses a challenge for the predictive performances, as it is harder for the model, with respect to a balanced data context, to learn the characteristics of samples from the minority class, due to their minor contribution in the learning process and thus it is harder to learn to differentiate well among "anomaly" and "no anomaly" examples. Given these difficulties, the model obtains promising results that in the future can hopefully be improved by providing more data.

6.3 Optimization for edge devices

The specifications from the lighting company require each *3D printer* to be equipped with a *Rasperry Pi* [65] device on which the Machine Learning model is deployed and a *Camera* that takes care of streaming the printing process and providing this input to the edge device (see Figure 6.4). This means that an important requirement

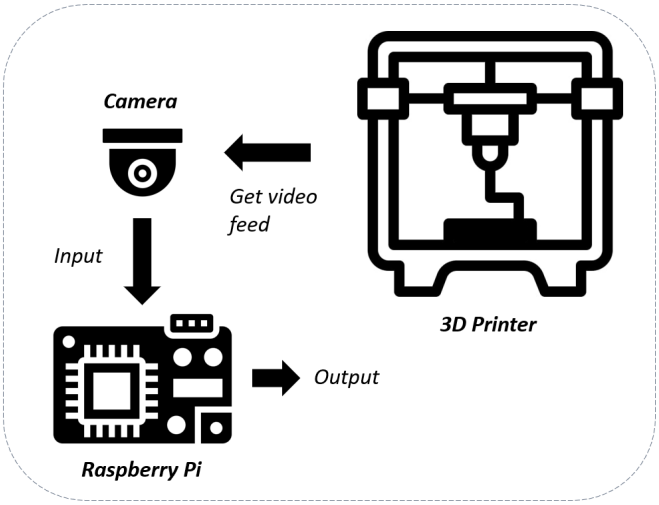

Figure 6.4: The image illustrates the main components to perform Anomaly Detection through the proposed ML system, during inference time. **3D Printer**: The device used for 3D printing. **Camera**: The video camera used to digitize, as input stream, the printing process. **Raspberry Pi**: The device used to run the Anomaly Detection ML algorithm for that specific printing process.

for the Anomaly Detection model is to be able to run on tiny edge-arm devices like the Raspberry Pi, while maintaining good response time in prediction. To allow this, some optimizations are performed separately on the trained models for *Nozzle Detection* and *ABN Classification* respectively. To compare the performance of the Anomaly Detection model in terms of inference time, according to the model

optimization strategy chosen, a Raspberry Pi 4B [66] device is used, whose main technical specifications consist in a Quad core Cortex-A72 (ARM v8) 64-bit System on Chip (SoC) @ 1.5GHz and a 8GB LPDDR4-3200 SDRAM.

Concerning the *Nozzle Detection* task, benchmarks for CPU inference are performed on the PyTorch trained model obtained in Section 5.1.2, considering Open Neural Network Exchange (ONNX) [67] and Tensorflow Lite [47] export formats by *YOLOv5*. Benchmarking is computed by selecting as data input the validation set from the Nozzle Detection dataset, which is made up of 1119 video frames (see Table 5.1). Table 6.4 reports the results obtained on the Raspberry Pi 4B, in terms of export model format and mean inference time per image. The model export is performed, for each type of format, considering an image size for the inference equal to 96, which is a lower resolution than the one used during the training but which still provides good results in terms of detection capabilities. Among the three tested formats, ONNX is the one which gives best results in

Format	**Inference time (ms)**
PyTorch	45.9
ONNX	19.2
TensorFlow Lite	21.7

Table 6.4: Nozzle Detection - Benchmarks results for CPU inference on Raspberry Pi 4B.

terms of mean inference time per image, obtaining a value equal to 19.2 ms, which consists in about 2.4x CPU inference speedup, with respect to the native Pytorch format. ONNX is an open source format for ML models, which provides great interoperability between multiple platforms and hardware and allows some optimization to accelerate inference. It gives the possibility to convert models and then use the ONNX Runtime as inference engine, which basically applies a variety of optimization and partitions on the model graph, thus acting as a cross-platform machine-learning model accelerator. Concerning the thesis work, given the Pytorch model, obtained as output from the Nozzle Detection training phase, the Pytorch documentation directly provides some methods to convert it to ONNX format and then run it on the associated ONNX Runtime.

As regards the *ABN Classification* task, the model trained in Section 5.2.2 and saved as TensorFlow SavedModel is converted into a TensorFlow Lite model. Tensorflow Lite is a Machine Learning library which helps running models on embedded devices, being optimized for on-device machine learning. To ensure minimum load and reduce execution latency, it uses an Interpreter with a static graph ordering and a custom memory allocator. Various optimization techniques can

be applied to reduce the size of the model and the latency (amount of computation required to perform the inference), at the expense of a potential loss in accuracy. Among the types of optimizations supported by the library, quantization is one of them. Quantization technique is the process of approximating a neural network that is obtained by reducing the precision of the computations and of the numbers representing the model parameters. In detail, it consists in converting 32-bit floating point numbers, which is the default, to a smaller precision, like 8-bit integer values, in such a way to obtain model compression and latency reduction. Two main types of quantization are available in TFLite:

- **Post-training quantization**: is a post-training conversion technique that can be applied to an already trained Tensorflow model when converting it to TFLite, using the TensorFlow Lite Converter [68]. Several options can be chosen such as:
  - Dynamic range quantization: is the simplest form, providing static quantization only for the model weights, from floating point to integer. It does not require a representative dataset for calibration.
  - Full integer quantization: in this case all the model math is fully integer quantized. This provides further improvements, with respect to dynamic range quantization, in terms of latency reduction and memory usage peaks. In addition, this technique allows compatibility with integer only hardware devices and accelerators. To obtain a full integer quantized model, it is necessary to perform a calibration phase, including a calibration dataset, to estimate the range of values for all the variable floating point tensors contained in the model.
  - Float16 quantization: allows to decrease the size of a floating point model applying a quantization to the weights to obtain 16-bit floating point numbers, following the float16 IEEE standard. This strategy reduces the weights by a half, generating a minimal loss in accuracy, but the decrease in latency is not as relevant as in quantization techniques to fixed point math.
- **Quantization-aware training**: is a training-time technique to improve the accuracy of quantized models by introducing some inference-time quantization error as noise, which is accumulated in the total loss, so that the optimizer learns parameters, more robust to quantization, around that loss.

The choice fell on the post-training integer quantization technique, which is applied during the model exporting phase. The calibration dataset is chosen as a subset of 100 samples from the validation set defined in Table 5.2. Table 6.5 shows the comparison between the results obtained on the Raspberry Pi 4B, with and without

quantization. Results are computed in terms of mean inference time per image, considering as data input the validation set from the 3D Printing Defects dataset, which is made up of 684 image crops. Looking at the values obtained, the final

Format	Inference time (ms)
Tensorflow Lite	35.2
Tensorflow Lite (Full integer quantization)	22.7

Table 6.5: ABN Classification - Benchmarks results for CPU inference on Raspberry Pi 4B.

model chosen for the classification task is the post-training integer quantized version, which allows about 1.5x CPU inference speedup, with respect to the unquantized version, without loosing too much its classification abilities.

Overall, the Anomaly Detection model takes about 42 ms to execute the task on a single image, where 19.2 ms is the time needed for the nozzle detection and 22.7 ms is the time required for the classification of the area below the nozzle (ABN). When a 30 FPS stream is given in input to the model, the algorithm is able to manage the video stream, by outputting the "anomaly" or "no anomaly" label in near real time. It should be noted that for this type of application it would be sufficient to detect defects with a much lower frequency than that provided by a 30 FPS stream. For example, it has been tested that for not missing important frames, it is sufficient to analyze about 10 FPS depending on the speed of the nozzle. For the purposes of the thesis, however, it has been tried to create a model that is as performing as possible in terms of both inference time and ability to detect anomalies, being able to detect a possible defect with low latency, looking at every frame received from a 30 FPS stream.

Chapter 7

Conclusions and future improvements

The thesis work presented a Machine Learning solution, optimized to run on low-powered devices such as the Raspberry-Pi4, being capable of detecting in near real time defects that can occur during the 3D printing process, and therefore helping to save print time and material. Thanks to this study, each 3D printer can be equipped with an automated "intelligent" system for continuous monitoring of the printing process, considerably reducing the manual work. This is particularly useful in large-scale contexts where manual monitoring is extremely difficult as an operator is assigned to multiple machines, as well as being an important aspect from the point of view of worker safety to avoid prolonged contact with particles harmful to health.

As described in Chapters 4 and 5, from a technical point of view, the thesis work proposed a multi-step Deep Learning pipeline to help in alleviating the critical issues highlighted above: the first step consists in the detection of the 3D Printer Nozzle and the Area Below the Nozzle (ABN), implemented using YOLOv5 (5.1); the second step, takes as input the ABN obtained before and performs a classification with EfficientNet-Lite as backbone network, giving as high level result the target label "anomaly" or "no anomaly" (5.2). This is a first Proof of Concept (PoC) for the problem under analysis, which has been specifically trained and tested considering the data received from a multinational Dutch lighting corporation.

Overall, the solution presented fulfilled the goal posed by the thesis, both in terms of the ability of the model to *detect defects* during a 3D printing process (6.1 - 6.2) and in terms of ability to perform single detection with *limited latency*, when a 30 FPS stream input is provided (6.3). More specifically, the *3D Printing Anomaly Detection Model* was validated on a test dataset kept aside with respect to the training process and containing real data coming from the lighting company,

which were previously annotated with the support of a domain expert, in particular as regards the classification of anomalies. Since the algorithm is composed of two models used in cascade at inference time, two separate validation phases were carried out, one relating to *Nozzle Detection* (6.1) and one relating to *ABN Classification* (6.2).

Concerning the detection phase, as explained in the Data Acquisition and Preparation phase (5.1.1), two types of nozzles were identified by inspecting the material provided and both of them were correctly detected by the model in a high percentage of frames. Specifically, the results for this first step were satisfactory in terms of metrics for the object detection task, since the model evaluated on the test set has obtained a mAP@0.5 equal to 0.976, which is one of the main metric used to assess the robustness of an object detection model.

With regard to the classification phase, its focus is on the *Visual Anomaly Detection* task, whose practical use case concerns *Defects Detection* in the manufacturing sector, particularly in the industrial process of a 3D printer, which corresponds to the main challenge of the thesis work. Looking at the literature and analyzing the data received by the company, it was possible to define various printing defects that occur repeatedly in 3D printing processes, categorizing them into defined groups (2.2). This consideration, also given the fact that the process takes place in a controlled environment, suggested that in the context under analysis the anomalies are not completely unpredictable, but on the contrary they have their own specificity due to certain triggering factors. That said, the design choice was to treat the problem in a supervised fashion as a binary classification task, also considering that a sufficient amount of data samples was available for the "anomaly" class and that many other real data could be gained in the future from the 3D printing industrial processes (3.2). Taking into account the imbalancing between the two classes, which is a direct consequence of the nature of the problem, the benchmark metric used was the F1 score computed with respect to the minority class ("anomaly"), which takes into consideration both Precision (measure of quality) and Recall (measure of quantity) in its formulation. By evaluating the classification model on the test set, a F1 score equal to 0.80 has been obtained. This is a promising result, bearing in mind that it is difficult for a supervised Machine Learning system to handle imbalanced data cases and in particular, with regards to Deep Learning models, a large amount of data is required to learn how to extract the most relevant features.

Future improvements. Even if the presented solution satisfied the objective posed at the beginning of the thesis, some improvements can be done to increase the performance of the final ML system, especially as regards the *ABN Classification* phase which is the most critical one due to the intrinsic difficulty of the task. One restraint concerns the limited amount of data samples received at the moment of model design, both in terms of quantity and in terms of variety. More

data containing different lamps designs, colours and materials would be useful, together with a more representative set of anomalies for all of them. This is mainly due to the fact that, in order to have higher performance, this type of algorithm requires a large and diversified amount of data and that without this some complex lamp designs, with a particular finish, could be misdetected if not well represented in the set of training. A limitation, specific to the supervised setting, is that a good set of representative data is necessary not only for the normal behaviour but also for the defective cases, which is difficult and sometimes unfeasible to obtain. That said, given that the defects in this application can be grouped in main "clusters" and considering the industrial context which allows to setup a dedicated camera system to obtain a continuous flow of data, an interesting future development could be to have a retraining process for the ML algorithm that is triggered every time a new nozzle or anomaly type is available. This will create a system which semi-automatically enhances its performance. In addition, with more data available

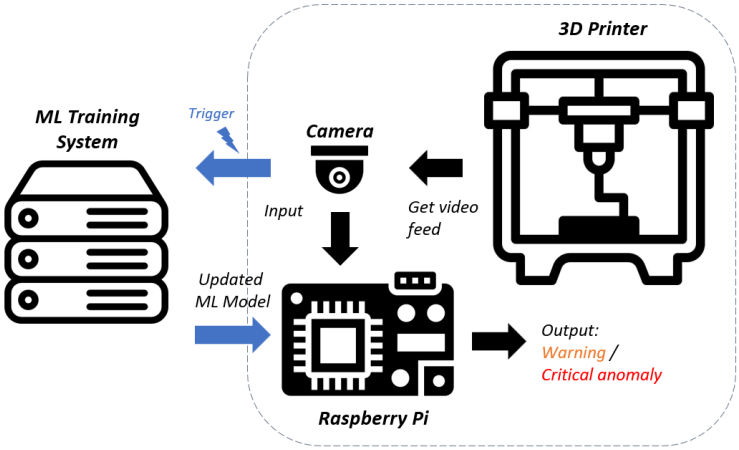

Figure 7.1: Improved *3D Printing Anomaly Detection* system.

concerning the anomalies, it could be interesting to create a multi-class *ABN classification* model. This will allow the complete *3D Printing Anomaly Detection* system to have a near real-time feedback about which anomaly has happened and therefore implement an automatic reaction for certain type of anomalies. For example, if a severe anomaly is detected (like "Layer shifting"), which leads to an object that will be hardly recoverable after production, the 3D printing process could be automatically stopped. In contrast, if a less severe anomaly is detected (like "Stringing"), only a warning is sent to the operator, which can perform a manual check and decide if the 3D Printing process needs to be stopped. Figure 7.1 shows the main actors in the process of *3D Printing Anomaly Detection*, consisting in the *3D Printer*, the *Camera* and the *Raspberry Pi* on which the ML model is

deployed. The *ML Training System* is a new actor, in this improved version, which receives as input new data to perform a new training and produces as output the updated model to be deployed on the Raspberry Pi. Concerning the anomalies, the output of the model is differentiated between "warning" and "critical anomaly". This enables to treat differently the two outcomes, implementing an automatic reaction to stop the process only for those anomalies that are for sure unrecoverable.

Overall it was a very interesting learning experience that allowed me to make new experiences, such as visiting a 3D printer industrial setting, acquiring new knowledge and applying what I learned in my course of study, bringing my personal contribution on a practical use case in an industrial context. In conclusion, I think that my thesis work could be useful, with the aim of reducing waste of resources, in other contexts besides that of 3D printing of lamps, given that 3D printers are used nowadays for many different types of applications and their use is increasing more and more.

Bibliography

[1] T.T. Wohlers, Wohlers Associates, and T. Caffrey. *Wohlers Report 2014: 3D Printing and Additive Manufacturing State of the Industry Annual Worldwide Progress Report*. Wohlers Associates, 2014. ISBN: 9780991333202. URL: https://books.google.it/books?id=iCamoAEACAAJ (cit. on p. 1).

[2] Özgür Keleş, Caleb Wayne Blevins, and Keith J. Bowman. «Effect of build orientation on the mechanical reliability of 3D printed ABS». In: *Rapid Prototyping Journal* 23 (2017), pp. 320–328. URL: https://doi.org/10.1108/RPJ-09-2015-0122 (cit. on p. 1).

[3] *ISO/ASTM 52900:2021, Additive manufacturing — General principles — Fundamentals and vocabulary*. https://www.iso.org/standard/74514.html (cit. on p. 1).

[4] *Machine Learning Reply*. https://www.reply.com/machine-learning-reply/it/ (cit. on p. 2).

[5] *EIT Digital*. https://www.eitdigital.eu/ (cit. on p. 2).

[6] *Print Quality Troubleshooting*. https://help.prusa3d.com/category/print-quality-troubleshooting_225 (cit. on pp. 8, 10).

[7] *Classification: How Many Ways Can a Print Fail?* https://medium.com/analytics-vidhya/classification-how-many-ways-can-a-print-fail-80e5715edff5 (cit. on p. 10).

[8] Felix Baumann and Dieter Roller. «Vision based error detection for 3D printing processes». In: *MATEC Web of Conferences* 59 (May 2016), p. 06003. DOI: 10.1051/matecconf/20165906003 (cit. on p. 10).

[9] Chenang Liu, Andrew Chung Chee Law, David Roberson, and Zhenyu Kong. «Image Analysis-based Closed Loop Quality Control for Additive Manufacturing with Fused Filament Fabrication». In: *Journal of Manufacturing Systems* 51 (Apr. 2019), p. 86. DOI: 10.1016/j.jmsy.2019.04.002 (cit. on p. 10).

[10] Ugandhar Delli and Shing Chang. «Automated Process Monitoring in 3D Printing Using Supervised Machine Learning». In: *Procedia Manufacturing* 26 (2018). 46th SME North American Manufacturing Research Conference, NAMRC 46, Texas, USA, pp. 865–870. ISSN: 2351-9789. DOI: https://doi.org/10.1016/j.promfg.2018.07.111. URL: https://www.sciencedirect.com/science/article/pii/S2351978918307820 (cit. on p. 11).

[11] Zeqing Jin, Zhizhou Zhang, and Grace X. Gu. «Autonomous in-situ correction of fused deposition modeling printers using computer vision and deep learning». In: *Manufacturing Letters* 22 (2019), pp. 11–15. ISSN: 2213-8463. DOI: https://doi.org/10.1016/j.mfglet.2019.09.005. URL: https://www.sciencedirect.com/science/article/pii/S2213846319300847 (cit. on p. 11).

[12] Yuanbin Wang, Jiakang Huang, Yuan Wang, Sihang Feng, Tao Peng, Huayong Yang, and Jun Zou. «A CNN-Based Adaptive Surface Monitoring System for Fused Deposition Modeling». In: *IEEE/ASME Transactions on Mechatronics* 25.5 (2020), pp. 2287–2296. DOI: 10.1109/TMECH.2020.2996223 (cit. on p. 11).

[13] Aditya Saluja, Jiarui Xie, and Kazem Fayazbakhsh. «A closed-loop in-process warping detection system for fused filament fabrication using convolutional neural networks». In: *Journal of Manufacturing Processes* 58 (2020), pp. 407–415. ISSN: 1526-6125. DOI: https://doi.org/10.1016/j.jmapro.2020.08.036. URL: https://www.sciencedirect.com/science/article/pii/S1526612520305405 (cit. on p. 11).

[14] Konstantinos Paraskevoudis, Panagiotis Karayannis, and Elias P. Koumoulos. «Real-Time 3D Printing Remote Defect Detection (Stringing) with Computer Vision and Artificial Intelligence». In: *Processes* 8.11 (2020). ISSN: 2227-9717. DOI: 10.3390/pr8111464. URL: https://www.mdpi.com/2227-9717/8/11/1464 (cit. on p. 11).

[15] Pattinson Sebastian W. Brion Douglas A. J. «Generalisable 3D printing error detection and correction via multi-head neural networks». In: *Nature Communications* 13.1 (2022). ISSN: 2041-1723. DOI: 10.1038/s41467-022-31985-y. URL: https://doi.org/10.1038/s41467-022-31985-y (cit. on p. 11).

[16] S.J.D. Prince. *Computer Vision: Models Learning and Inference*. Cambridge University Press, 2012. ISBN: 1107011795 (cit. on p. 12).

[17] Richard Szeliski. *Computer Vision: Algorithms and Applications, 2nd ed.* Springer, 2022. ISBN: 3030343715 (cit. on p. 13).

[18] *Dive into deep learning*. https://d2l.ai/chapter_introduction/index.html#supervised-learning (cit. on p. 14).

[19] Jie Yang, Ruijie Xu, Zhiquan Qi, and Yong Shi. «Visual Anomaly Detection for Images: A Systematic Survey». In: *Procedia Computer Science* 199 (2022). The 8th International Conference on Information Technology and Quantitative Management (ITQM 2020 & 2021): Developing Global Digital Economy after COVID-19, pp. 471–478. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2022.01.057. URL: https://www.sciencedirect.com/science/article/pii/S1877050922000576 (cit. on p. 15).

[20] Zesheng Lin, Hongxia Ye, Bin Zhan, and Xiaofeng Huang. «An Efficient Network for Surface Defect Detection». In: *Applied Sciences* 10.17 (2020). ISSN: 2076-3417. DOI: 10.3390/app10176085. URL: https://www.mdpi.com/2076-3417/10/17/6085 (cit. on p. 15).

[21] Yibin Huang, Congying Qiu, Xiaonan Wang, Shijun Wang, and Kui Yuan. «A Compact Convolutional Neural Network for Surface Defect Inspection». In: *Sensors* 20.7 (2020). ISSN: 1424-8220. DOI: 10.3390/s20071974. URL: https://www.mdpi.com/1424-8220/20/7/1974 (cit. on p. 15).

[22] Jakob Božič, Domen Tabernik, and Danijel Skočaj. *End-to-end training of a two-stage neural network for defect detection*. 2020. DOI: 10.48550/ARXIV.2007.07676. URL: https://arxiv.org/abs/2007.07676 (cit. on p. 15).

[23] Hui Lin, Bin Li, Xinggang Wang, Yufeng Shu, and Shuang Niu. «Automated defect inspection of LED chip using deep convolutional neural network». In: *Journal of Intelligent Manufacturing* (2019), pp. 1–10. URL: https://doi.org/10.1007/s10845-018-1415-x (cit. on p. 15).

[24] Jiabin Zhang, Hu Su, Wei Zou, Xinyi Gong, Zhengtao Zhang, and Fei Shen. «CADN: A Weakly Supervised Learning-Based Category-Aware Object Detection Network for Surface Defect Detection». In: *Pattern Recogn.* 109.C (Jan. 2021). ISSN: 0031-3203. DOI: 10.1016/j.patcog.2020.107571. URL: https://doi.org/10.1016/j.patcog.2020.107571 (cit. on p. 15).

[25] Jakob Božič, Domen Tabernik, and Danijel Skočaj. «Mixed supervision for surface-defect detection: From weakly to fully supervised learning». In: *Computers in Industry* 129 (Aug. 2021), p. 103459. DOI: 10.1016/j.compind.2021.103459. URL: https://doi.org/10.1016%2Fj.compind.2021.103459 (cit. on p. 16).

[26] Yadan Li, Zhenqi Han, Haoyu Xu, Lizhuang Liu, Xiaoqiang Li, and Keke Zhang. «YOLOv3-Lite: A Lightweight Crack Detection Network for Aircraft Structure Based on Depthwise Separable Convolutions». In: *Applied Sciences* 9.18 (2019). ISSN: 2076-3417. DOI: 10.3390/app9183781. URL: https://www.mdpi.com/2076-3417/9/18/3781 (cit. on p. 16).

[27] Junwen Chen, Zhigang Liu, Hongrui Wang, Alfredo Núñez, and Zhiwei Han. «Automatic Defect Detection of Fasteners on the Catenary Support Device Using Deep Convolutional Neural Network». In: *IEEE Transactions on Instrumentation and Measurement* 67.2 (2018), pp. 257–269. DOI: 10.1109/TIM.2017.2775345 (cit. on p. 16).

[28] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*. 2015. DOI: 10.48550/ARXIV.1506.02640. URL: https://arxiv.org/abs/1506.02640 (cit. on pp. 17, 18).

[29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. *Going Deeper with Convolutions*. 2014. DOI: 10.48550/ARXIV.1409.4842. URL: https://arxiv.org/abs/1409.4842 (cit. on p. 19).

[30] Joseph Redmon and Ali Farhadi. *YOLO9000: Better, Faster, Stronger*. 2016. DOI: 10.48550/ARXIV.1612.08242. URL: https://arxiv.org/abs/1612.08242 (cit. on p. 19).

[31] Joseph Redmon and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018. DOI: 10.48550/ARXIV.1804.02767. URL: https://arxiv.org/abs/1804.02767 (cit. on p. 19).

[32] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. DOI: 10.48550/ARXIV.2004.10934. URL: https://arxiv.org/abs/2004.10934 (cit. on p. 19).

[33] Mingxing Tan and Quoc V. Le. «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks». In: (2019). DOI: 10.48550/ARXIV.1905.11946. URL: https://arxiv.org/abs/1905.11946 (cit. on pp. 19, 20).

[34] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. «MobileNetV2: Inverted Residuals and Linear Bottlenecks». In: (2018). DOI: 10.48550/ARXIV.1801.04381. URL: https://arxiv.org/abs/1801.04381 (cit. on p. 20).

[35] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. «MnasNet: Platform-Aware Neural Architecture Search for Mobile». In: (2018). DOI: 10.48550/ARXIV.1807.11626. URL: https://arxiv.org/abs/1807.11626 (cit. on p. 20).

[36] H. Schulzrinne, A. Rao, and R. Lanphier. *RFC2326: Real Time Streaming Protocol (RTSP)*. USA, 1998. URL: https://www.rfc-editor.org/rfc/rfc2326.html (cit. on p. 24).

[37] *Adobe's Real Time Messaging Protocol*. https://rtmp.veriskope.com/pdf/rtmp_specification_1.0.pdf (cit. on p. 24).

[38] Wesley Eddy. *Transmission Control Protocol (TCP)*. RFC 9293. Aug. 2022. DOI: 10.17487/RFC9293. URL: https://www.rfc-editor.org/info/rfc9293 (cit. on p. 24).

[39] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *RFC 2616, Hypertext Transfer Protocol – HTTP/1.1*. 1999. URL: https://www.rfc-editor.org/rfc/rfc2616.html (cit. on p. 24).

[40] Glenn Jocher. *YOLOv5 Documentation*. https://docs.ultralytics.com/ (cit. on p. 24).

[41] Glenn Jocher. *ultralytics/yolov5*. https://github.com/ultralytics/yolov5/releases/tag/v6.1. DOI: 10.5281/zenodo.6222936. URL: https://doi.org/10.5281/zenodo.6222936 (cit. on p. 24).

[42] *Pytorch Framework*. https://pytorch.org/ (cit. on p. 25).

[43] Tsung-Yi Lin et al. «Microsoft COCO: Common Objects in Context». In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312 (cit. on pp. 25, 36).

[44] *CUDA Documentation*. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (cit. on p. 25).

[45] Glenn Jocher. *ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support*. https://github.com/ultralytics/yolov5/releases/tag/v6.0. DOI: 10.5281/zenodo.5563715. URL: https://doi.org/10.5281/zenodo.5563715 (cit. on p. 25).

[46] *EfficientNet-Lite*. https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet/lite (cit. on p. 25).

[47] *TensorFlow Lite*. https://www.tensorflow.org/lite (cit. on pp. 25, 65).

[48] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. *Squeeze-and-Excitation Networks*. 2017. DOI: 10.48550/ARXIV.1709.01507. URL: https://arxiv.org/abs/1709.01507 (cit. on p. 25).

[49] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. DOI: 10.48550/ARXIV.1710.05941. URL: https://arxiv.org/abs/1710.05941 (cit. on p. 26).

[50] Alex Krizhevsky. *Convolutional Deep Belief Networks on CIFAR-10*. http://www.cs.utoronto.ca/~kriz/conv-cifar10-aug2010.pdf. 2010 (cit. on p. 26).

[51] Sinno Jialin Pan and Qiang Yang. «A Survey on Transfer Learning». In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359. DOI: 10.1109/TKDE.2009.191 (cit. on p. 26).

[52] Olga Russakovsky et al. «ImageNet Large Scale Visual Recognition Challenge». In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y (cit. on p. 26).

[53] Min Lin, Qiang Chen, and Shuicheng Yan. *Network In Network*. 2013. DOI: 10.48550/ARXIV.1312.4400. URL: https://arxiv.org/abs/1312.4400 (cit. on p. 26).

[54] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting». In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html (cit. on p. 26).

[55] John S. Bridle. «Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition». In: *Neurocomputing*. Ed. by Françoise Fogelman Soulié and Jeanny Hérault. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 227–236. ISBN: 978-3-642-76153-9 (cit. on p. 26).

[56] *ISO8601 Standard*. https://www.iso.org/obp/ui/#iso:std:iso:8601:-1:ed-1:v1:en (cit. on p. 28).

[57] *Computer Vision Annotation Tool (CVAT)*. https://www.cvat.ai/ (cit. on p. 33).

[58] *YOLOv5 - Transfer Learning with Frozen Layers*. https://github.com/ultralytics/yolov5/issues/1314 (cit. on p. 36).

[59] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*. 2017. DOI: 10.48550/ARXIV.1706.02677. URL: https://arxiv.org/abs/1706.02677 (cit. on p. 36).

[60] *Binary Cross-Entropy with Logits Loss*. https://pytorch.org/docs/master/generated/torch.nn.BCEWithLogitsLoss.html#bcewithlogitsloss (cit. on p. 37).

[61] *PyTorch - SGD*. https://pytorch.org/docs/stable/generated/torch.optim.SGD.html#sgd (cit. on p. 37).

[62] *Tensorflow*. https://www.tensorflow.org/ (cit. on p. 45).

[63] *Keras*. https://keras.io/ (cit. on p. 46).

[64] Rafael Padilla, Wesley L. Passos, Thadeu L. B. Dias, Sergio L. Netto, and Eduardo A. B. da Silva. «A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit». In: *Electronics* 10.3 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10030279. URL: https://www.mdpi.com/2079-9292/10/3/279 (cit. on p. 55).

[65] *Raspberry Pi*. https://www.raspberrypi.org/ (cit. on p. 64).

[66] *Raspberry Pi 4*. https://www.raspberrypi.com/products/raspberry-pi-4-model-b (cit. on p. 65).

[67] *Open Neural Network Exchange*. https://onnx.ai/ (cit. on p. 65).

[68] *TFLiteConverter*. https://www.tensorflow.org/api_docs/python/tf/lite/TFLiteConverter (cit. on p. 66).