School of Industrial and Information Engineering

Master of Science – Mechanical Engineering

Artificial Intelligence for Image Classification and Anomaly Detection in the Food Sorting industry
A Comparative Study

Supervisor:
Prof. Marco Tarabini

Co-supervisors:

Prof. Marco Bocciolone
Ing. Davide Maria Fabris

MSc Thesis of:
Pietro Oppici 971543

Jacob Niccolai 963162

Academic Year 2021 – 2022

Acknowledgements

I would like to express my gratitude to prof. Tarabini and Dr. Fabris, for the opportunity to carry out this thesis and the guidance in pursuing it. I would also like to show my appreciation to my colleague Pietro Oppici for the great teamwork.

Thanks to my parents, Marco and Petra, for guiding me over the years towards the person I am.

Thanks to Daria, for standing by me with your strength, your care and your smile.

Thanks to my friends, for always being there.

--Jacob Niccolai

I would like to express my gratitude to prof. Tarabini and Dr. Fabris, for the opportunity to carry out this thesis and the guidance in pursuing it. I would also like to show my appreciation to my colleague Jacob Niccolai for the great teamwork.

I would like to thank my parents, Carlo and Cristina, and my brother, Filippo, for giving me the opportunity to embark on this journey and for being there for me, especially in difficult times.

Thanks to my partner, Valentina, for always being by my side.

Thanks to all my friends for the fantastic times we spent together.

--Pietro Oppici

Abstract

This thesis aims at the implementation and comparison of machine vision algorithms based on deep learning for the quality control of tomatoes. Currently the partner of this thesis, which operates in the field of automated industrial food sorting, was not able to build a properly functioning machine, since the existing defects pose some specific issues that traditional machines are not capable to overcome. Therefore, the partner is interested in the recent developments of AI based computer vision and wants to implement such a method in a new machine. In this way a system able to substitute cumbersome, human labour-based, quality control for tomatoes is created. The goal of this thesis is to identify some open-source state of the art machine learning models for these needs, adapt them to the specific case, construct a common pipeline to handle and optimize them in a standardized way and finally identify the best one for the given application. Three classification algorithms and three anomaly detection algorithms based on deep learning are applied and optimized to this purpose in a python environment. Those optimized models are then compared. Comparisons are performed with statistically relevant tools to identify the best model for the given application in a robust way. Also, some insights on strengths and weaknesses of the different models are given. These characteristics could be exploited in other applications similar to the one presented. The best model for the case in exam is so identified. This model shows high performances, with a mean accuracy of 98.61%, an AUROC of 99.94% and elaboration times for single image low enough to be integrated with the already existing sorting machines. This optimized model is then additionally tested on in-field acquired images of the same vegetable obtaining an accuracy of 98.56%, proving the robustness of the proposed solution.

Keywords: Food Sorting; Machine Vision; Deep Learning; Python; Keras; Tensorflow; Anomaly Detection; Image Classification.

Sommario

Questa tesi ha come scopo l'implementazione e il confronto tra algoritmi di visione artificiale basati sul deep learning per il controllo qualità di pomodori. Attualmente il partner industriale di questa tesi, che opera nel settore delle macchine selezionatrici automatiche, non riesce a fornire i propri servizi in questo campo perché i particolari difetti presenti nei pomodori freschi pongono delle problematiche che i sistemi tradizionali da loro adottati non sono in grado di superare. Per questo motivo il partner industriale è interessato ai recenti sviluppi dell'intelligenza artificiale nei sistemi di visione automatica e vuole sviluppare una nuova macchina usando questi metodi. In questa maniera si rende possibile un dispositivo in grado di sostituire il lavoro umano in questa gravosa operazione di controllo qualità nella filiera del pomodoro fresco. L'obbiettivo di questa tesi è di identificare dei modelli di machine learning sorgente aperti per questo scopo, adattarli all'applicazione specifica, costruire un architettura comune per gestirli e ottimizzarli in maniera standardizzata e infine identificare il modello migliore per l'applicazione di interesse. Con questo fine vengono identificati, implementati e ottimizzati tre algoritmi di classificazione e tre di rilevamento di anomalie in un ambiente Python. Questi modelli ottimizzati sono poi confrontati tra loro. Le comparazioni vengono fatte con strumenti statisticamente rilevanti per individuare il modello migliore in maniera robusta. Allo stesso tempo in questo processo vengono messi in luce i punti di forza dei singoli algoritmi, caratteristiche che potrebbero essere sfruttate in altre applicazioni simili a quella qui presentata ma con priorità diverse. Viene quindi identificato il modello migliore per l'applicazione. Questo restituisce delle prestazioni elevate, con un'accuratezza media del 98.61%, AUROC di 99.94% e tempi di elaborazione sulla singola immagine abbastanza bassi da poter essere integrati con le macchine selezionatrici già esistenti. Questo modello ottimizzato viene poi ulteriormente collaudato su immagini di pomodori acquisite in loco con un normale smartphone ottenendo un'accuratezza del 98.56%, il che dimostra la robustezza della soluzione qui proposta.

Parole chiave: Visione Artificiale; Controllo Qualità; Deep Learning; Python; Keras; Tensorflow; Classificazione di Immagini, Rilevamento di Anomalie.

Table of Contents

1 Introduction

This introduction section is divided in 5 parts. In the first the motivations and objectives of the work are stated, after that the literature research is given to introduce and briefly describe the working principle of the models used. In part three, the datasets used are described and in part four, the hardware settings for the experiments are illustrated. In the fifth and last paragraph a short outline of the thesis is reported to better understand the workflow of the experiments and the structure of the document.

1.1 Project Objectives

Quality control in the food industry is extremely important. Since it is one of the biggest markets on earth, this process often includes controlling millions of items in the production line while maintaining a high quality standard. In many cases using human labour to perform this task is very costly and represents a strenuous work for the operators. Automated solutions for quality check are therefore strongly requested in the food industry.

However, this sorting procedure is particularly difficult in the case of vegetables since each element has a slightly different shape, weight, colour and, when it comes to quality control, also the defects can be various. Because of that, it is not easy to identify a single simple feature able to distinguish good units from bad ones. This means that in some cases it becomes impossible to model characteristics of the different classes in strict terms. Because of that, classical algorithms based on telling the computer what to do one step after another, splitting the problem in smaller simpler tasks is not achievable. Deep learning algorithms, on the other hand, seem to be able to handle the problem.

Deep learning is a subsection of machine learning in which so-called artificial neurons are used. These neurons are mathematical functions connected one another. By constructing a neural network of these functions, it is possible to extract features from complex data. In particular stacking many layers of them has shown to be the most effective way to use them in practical applications [1]. This layout which resembles the structure and function of biological brain cells is the reason why these new machine learning algorithms are known as a whole with the name “Deep learning algorithms” based on “neural networks”. In fact, these algorithms learn from observational data, figuring out their own solution to the problem at hand. These features make them achieve outstanding performances on many important problems in computer vision, speech recognition, and natural language processing [2]. This is because deep learning has shown tremendous capabilities in learning expressive representations of complex data. In the food industry in particular, Deep Learning applications for computer vision are raising attention for the reason cited above: many quality control procedures in the industry are still not automated since standard algorithms fail to recognize the good product from the bad one.

Automatizing these systems brings strong advantages. Therefore, it is an industrial need to understand which among those deep learning algorithms performs the quality control task the best and whether such a system has market-ready performances. This thesis work reaches in this direction. It is part of the project ***”Studio e sviluppo di TEcnologie avanzate per il SORting automaticO nei processi di produzione alimentari – TESORO”*** issued by the Italian Ministry of Economic Development and developed in collaboration with various industrial partners among which Raytec Vision S.P.A. . Raytec Vision S.P.A. is a big player in the machine vision industry for fruit and vegetables quality control. Their core business consists in developing sorting machines which are able to discard waste product basing on specific information coming from optical sensors. Their machines put them among the world leaders in this field. Still, they struggle to perform effective quality control with those techniques on food categories in which specific information aren’t identifiable in a rigorous way. In those cases, the human eye still outperforms their classical machine vision systems, and the company could not enter the corresponding market sections. For example, and this is the application studied in detail here, when dealing with tomatoes their traditional methods are not able to bring the needed performances. The study reported in the following focuses on tomatoes, but other vegetables are still not handled with automated systems for the same motivations. This is the reason why deep learning represent a big opportunity for Raytec Vision S.P.A. to expand its market. This thesis reaches in that direction. Essentially the goal is to implement, analyse peculiar characteristics, and compare methods based on neural networks able to divide images of tomatoes into good and bad samples.

Among models in the deep learning field suitable for this task, two major families are identified [3]: Images Classification (IC) algorithms based on deep learning and deep learning approaches for Anomaly Detection (AD) (deep anomaly detection for short).

The first ones try to learn the characteristics of the different families in the dataset and are so able to identify the two families of good and bad data. For image classification tasks many very strong performing neural networks are available. Most of them were developed to participate into the ImageNet large scale visual recognition challenge (ILSVRC) competition and show astonishing results [1], [4].

Anomaly detection, on the other hand, is referred to as the process of detecting images that deviate from the majority of data instances [5]. In other words, they operate based on the idea that a family of good images exist and therefore classifying everything else as bad. Due to the afore-mentioned ability of deep neural networks to extract features in complex structured data like images, the advancements of deep learning brought also significant steps forward in anomaly detection [3] and has emerged as a critical direction in this field. With respect to classification, anomaly detection algorithms do not suffer from insufficient representative samples in the anomalous class. In fact, in this case the model does not try to represent the good and bad samples in two classes but simply labels as bad everything which is not coherent with the good class.

It must be noticed that in this master thesis, classification and anomaly detection algorithms were compared with the aim of understanding which of these two types of paradigm is the best one for the partner's industrial needs. This means that the experiments had as objective to answer the question: "Which is the best way to discern vegetables I would like to keep in my production line from vegetables I would like to remove?" In this way the usage of anomaly detection and classification algorithms satisfy the same need. In fact in the following, it should be remembered and noticed that even if here they are used to perform the same task each of these two methods has specific peculiarities that result in evident advantages in other situations. In fact, as the name states, anomaly detection, which is also in some case know as novelty detection, is generally meant to be trained only on good data or with few bad samples. This means that almost only the good part of the dataset is needed to use it. This is a big advantage in some cases. Indeed, if for example a new type of unknown disease appears in the vegetables after the system has been trained and installed, anomaly detection methods should be able to continue operating since any type of anomaly is recognized as such by the algorithm.

In classification methods on the other side also many examples of the bad dataset need to be seen by the system during training to be able to learn its characteristics. In the specific case of this thesis work, all possible anomalies were present in the dataset so that after the training, the systems will not have problems encountering unseen data. Anyway, in some applications, it may be difficult to always have a robust dataset representing all possible combinations of bad samples that could rise during the operative life of the machine. For this reason, the peculiar advantage given anomaly detection in that kind of situation is clear.

On the other hand, IC-based models have some other type of strength with respect to anomaly detection. In fact, as the name suggest they can be used to classify data. In this specific case the industrial need was to divide the data into two categories: good and bad. However, classification can also be performed with more than two classes. In many other industrial applications this property is surely needed. It is easy to understand that anomaly detection would not be applicable in such cases. For this reason, in the final part of this thesis, precisely in the results section, also some additional experiments done with the classification methods are reported to understand how they would perform if they had to divide the incoming tomato images not only in good and bad, but also based on their specific defect. This is also helpful to understand which type of fault is the most difficult to be identified by the net. These considerations about the peculiar advantages of each system will be repeated and better elaborated in the following. In short, it must be remembered that those two families of methods are used for the same task here to understand which is the best one in the specific application, but it is generally not possible to confront anomaly detection and classification models since they often perform different tasks.

1.2 Literature Research

The first step towards the development of this thesis is the research on academic and commercial publications to identify the most suitable algorithms for our application, keeping in mind the needs of the industrial partner, Raytec Vision S.P.A. To start this research some keywords are identified: python, deep learning, image classification, anomaly detection. Python is chosen among those keywords since it is the most used open-source programming language for machine learning [6]. Indeed, the following work is implemented in Python. In particular, the procedure for choosing classification algorithms is the following. Firstly, a cross-reference with the Raytec Vision S.P.A. engineers about what algorithms are implemented in their property software (i.e., MVTec Halcon ®) is made. After that, the equivalent of these algorithm in the form of Python open-source software is found.

Following the afore-mentioned rules, three algorithms were identified for classification:

- Mobilenet [7]
- Resnet50 [1]
- GoogLeNet [8]

Raytec Vision S.P.A. has not yet approached anomaly detection at all but is interested in having a comparison with classification algorithms. As for the choice of the anomaly detection algorithms to test, some state-of-the-art models are identified, which are:

- CFLOW-AD [9]
- PatchCore [10]
- DFKDE [11]

In the next sub-sections, a brief introduction to each of them is given.

1.2.1 Classification

For what concerns classification with deep neural networks the literature research shows that most of the improvements in recent years came from the ImageNet large scale visual recognition challenge competition [12]. This is an annual software contest where software programs compete to correctly classify from the ImageNet database. ImageNet is a large visual collection of more than fourteen million images, hand annotated with labels of the respective category [12]. In other words, many of the state-of-the-art algorithms for IC were built to compete and won this competition. This is interesting because also the selected models are deeply integrated with ImageNet and participated in this contest. In fact, in the following this database will be reused to pretrain the models. Before entering in the details of the algorithms analysed in the following, a brief introduction is given.

A classical IC deep learning algorithm receives images as input and gives the class to which each image belongs as output. In particular, among the possible deep learning algorithms, Convolutional Neural Networks (CNNs) are analysed. Typical CNN architectures stack a few convolutional layers then a pooling layer, then another few convolutional layers then another pooling layer, and so on. The image gets smaller and smaller as it progresses through the network, but it also typically gets deeper and deeper.

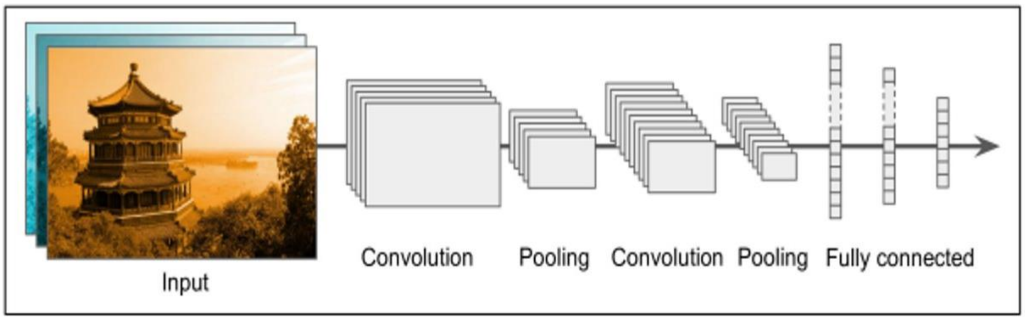


Figure 1 Typical CNN architecture

To be more precise, some definitions should be given. A **convolutional layer** converts all the pixels in its receptive field into a single value. In the case of an image, you will be decreasing the image size as well as bringing all the information in the field together into a single pixel. The result of the convolution is a series of feature map. This discussion can be appreciated looking at Figure 2. The function of a **pooling layer** is to reduce the spatial size of the representation in order to decrease the number of parameters and computation in the network. Moreover, it operates on each feature map obtained from the convolution independently. There are two types of pooling layers, which are max pooling and average pooling. In particular, max pooling is the most widely used. The reason why max pooling layers work so well in convolutional networks is that it helps the networks detect the features more efficiently after down sampling an input representation and it helps over-fitting by providing an abstracted form of the representation.

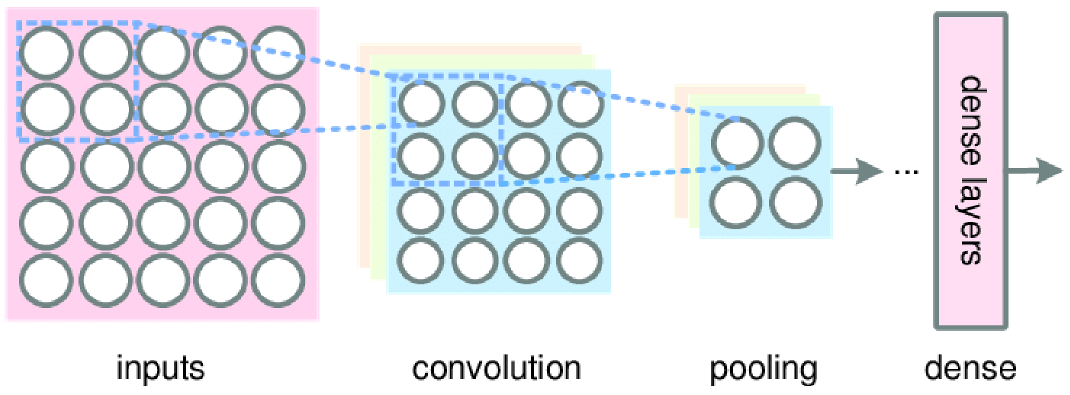


Figure 2 Details of the Convolutional and Pooling layers

1.2.1.1 ResNet50

The name ResNet stands for residual networks. What the name "residual networks" stands for is explained in the following.

As well-known the depth of the neural network is of crucial importance. Anyway, deep neural networks have a degradation problem: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly [1]. As explained in the paper, this degradation problem is not caused by the well-known overfitting, and it leads to a higher training error. The problem has been solved by means of the introduction of a deep residual learning framework [1]. Briefly, a feedforward neural network with shortcut connections was added. The latter skips one or more layers, and they perform identity mapping, as reported in Figure 3.

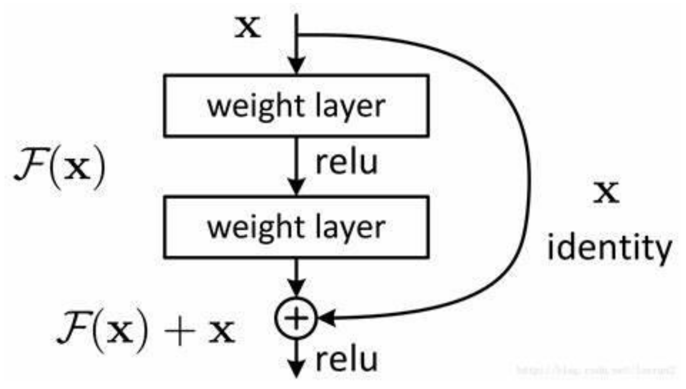

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 3 Residual learning: a building block with a shortcut connection

One important remark is that identity shortcut connections add neither extra parameter nor computational complexity [1]. This is quite important since it is possible to compare residual networks and plain networks (counterpart of residual ones that simply stack layers) that have the same number of parameters, depth, width and computational cost. The results reported are mainly two:

1. Extremely deep residual networks are easier to optimize. Moreover, they show a lower training error as the depth increases.
2. Deep residual networks can enjoy accuracy gains from greatly increased deepth

Thanks to these improvements, the 152-layer ResNet50 is able to reach a single-model top-5 validation error of 4.49%. Moreover, as reported in [1], an ensemble (i.e., a combination) of these residual networks achieves 3.57% error on the ImageNet test set. This result was sufficient to win the competition in 2015.

1.2.1.2 GoogLeNet

As well-known the most straightforward way of improving the performance of deep neural networks is by increasing their size [8]. It must be remembered that increasing the size of deep neural networks means increasing both width and size. This procedure leads to two main drawbacks:

1. Bigger number of parameters and so a network more prone to overfitting.
2. Bigger networks require to use more computational resources.

Here the idea behind the creation of the GoogLeNet's architecture arises, replacing the fully connected layers with sparse ones.

It is assumed that each unit from an earlier layer corresponds to some region of the input image and these units are grouped into filter banks. In the lower layers (the ones close to the input) correlated units would concentrate in local regions [8]. Thanks to this assumption, we would end up with a significant number of clusters concentrated in a single region. This region can be covered by a single layer of 1x1 convolutions in the next one. To avoid patch-alignment issues, filter sizes are restricted to 1x1, 3x3 and 5x5. Additionally, it is well-known that pooling operations are of fundamental importance for the success of convolutional networks. It suggests that adding a pooling path in each module should have an additional beneficial effect, too. One big problem with the above-mentioned modules, is that even a modest number of 5x5 convolutions can be prohibitively expensive on top of a convolutional layer with a large number of filters. This problem becomes even more pronounced once pooling units are added to the mix. To assess this problem, 1x1 convolutions are used to compute reductions before the expensive 3x3 and 5x5 convolutions as can be seen in Figure 4.

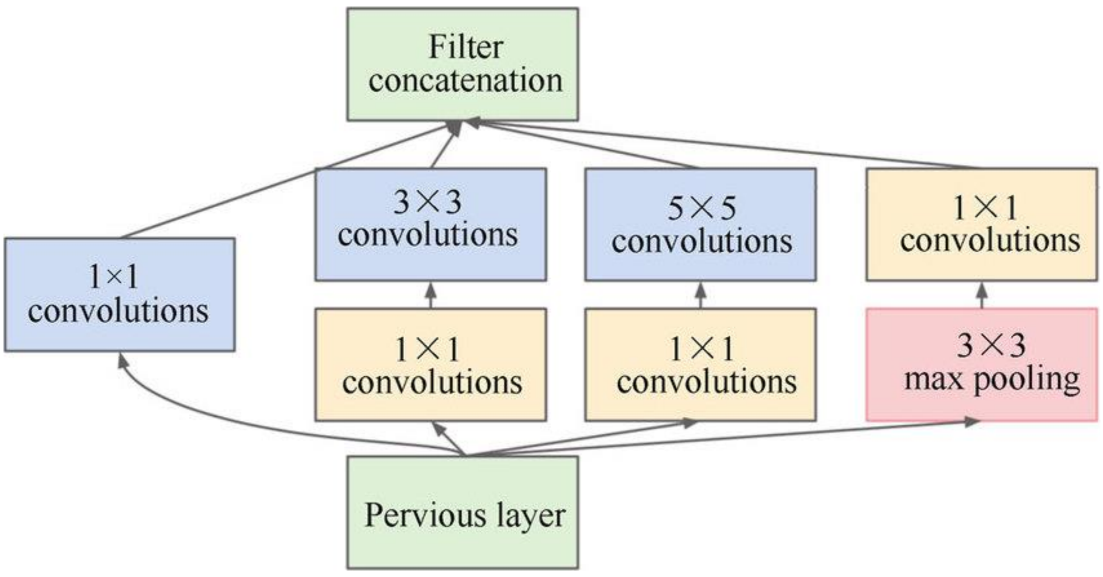


Figure 4 Inception module with dimensionality reduction

In general, an Inception network is a network consisting of modules of the above type stacked upon each other, with occasional max-pooling layers with stride two to halve the resolution of the grid.

By the “GoogLeNet” name we refer to the particular incarnation of the Inception architecture used in the submission for the ILSVRC 2014 competition [8].

1.2.1.3 Mobilenet

The general trend of making deeper and more complicated networks in order to achieve higher accuracy is not always the best solution. Indeed, if you think about many real-word application, such as robotics or self-driving cars, the recognition task needs to be carried out in a computationally limited platform. With this idea in mind, the specifics of the last classification algorithm are presented. MobileNet is based on depth wise separable convolutions which is a form of factorized convolutions which factorize a standard convolution into a depth wise convolution and a 1x1 convolution called a pointwise convolution. This factorization has the effect of drastically reducing computations and model size [7]. The above-mentioned depth wise separable convolution consists of two layers:

1. Depth wise convolutions, used to apply a single filter per each input channel.
2. Pointwise convolutions, simple 1x1 convolution, used to create a linear combination of the output of the previously explained layer.

The MobileNet structure is built on module as explained before except for the first layer which is a full convolution. In Figure 5 it is reported the comparison between a layer with regular convolutions, batch norm (BN) and rectified linear units (ReLU) nonlinearity and the factorized layer with depth wise convolution, 1x1 pointwise convolution as well as batch norm and ReLU after each convolutional layer.

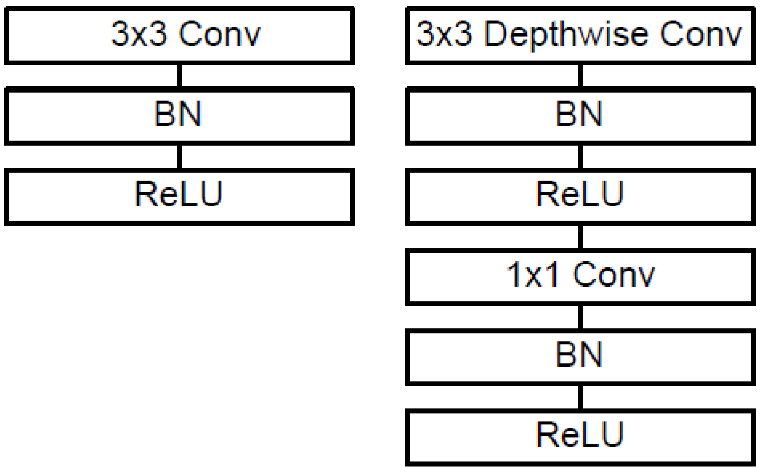


Figure 5 Left: Standard convolutional layer with batch norm and ReLU. Right: Depth wise Separable convolutions with Depth wise and Pointwise layers followed by batch norm and ReLU

Counting depth wise and pointwise convolutions as separate layers, MobileNet has 28 layers [7].

Although the previously explained architecture is already small and offers low latency, many times a specific application requires a smaller and faster model. In order to achieve these models, two parameters are introduced. The first one, α is called width multiplier. The role of the width multiplier α is to thin a network uniformly at each layer. The computational cost of a depth wise separable convolution with width multiplier α is:

$$D_K \cdot D_K \cdot \alpha M \cdot D_F \cdot D_F + \alpha M \cdot \alpha N \cdot D_F \cdot D_F \quad [\,1\,]$$

Where $\alpha \in (0, 1]$. Width multiplier has the effect of reducing computational cost and the number of parameters quadratically by roughly α^2. The other parameters of equation [1] are:

- M, number of input channels.
- N, number of output channels.
- $D_F \cdot D_F$, feature map size.
- $D_K \cdot D_K$, kernel size.

The second hyper-parameter introduced to reduce the computational cost is the resolution multiplier ρ. It is applied to the input image and the internal representation of every layer is subsequentially reduced by the same amount. The computational cost can be finally rewritten as

$$D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F \quad [\,2\,]$$

Where $\rho \in (0, 1]$. Resolution multiplier has the effect of reducing computational cost by ρ^2.

To conclude our discussion, here are a few results of interest obtained in the original paper using ImageNet. MobileNet is nearly as accurate as VGG16 while being 32 times smaller and 27 times less compute intensive. It is more accurate than GoogLeNet while being smaller and more than 2.5 times less computationally heavy [7]. Other impressive results are achieved in tasks such as face attributes, object detection and face embeddings.

1.2.2 Anomaly Detection

To introduce this paragraph about anomaly detection with neural networks, it must first be remembered that, unlike the classification task, here the neural network is only one part of the systems and does not directly give the verdict on if data are good or bad, as in the case with classification. In particular, it is known that neural networks have the ability to extract features from the images they receive. This property which the human eye has naturally can be visualized for example in Figure 6. In this image one of the filters of the first convolutional layer of ResNet50 is visualized. In practice Figure 6 shows which parts of the images the network is more focused on at that level of abstraction.

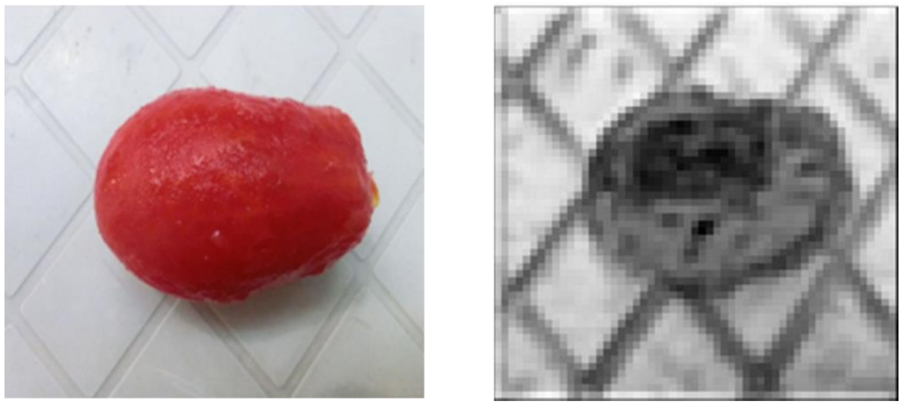


Figure 6 Visualization of a filter of the first convolutional layer of ResNet50 for the reported image. This shows the ability of neural network to extract features from images.

This images clearly reveals that there are some parts of the figure which represent a pattern for the network. In other words, this indicates that the algorithm is extracting features. The more we go down in the network the more this type of features become abstract so that their visualization on the starting image is not graphically satisfactory for the human eye. But it is evident that some kind of information is being squeezed.

A lot of anomaly detection algorithms work using this principle [13]. In fact, they squeeze the information using a network which in this case is called encoder. After that, starting from the squeezed information, another network doing the opposite job reconstructs the image from this vector of information. This is called a decoder. This kind of system is reported Figure 7. This structure is than trained with a loss function and many images so that the starting image and the reconstructed one become as similar as possible. This implicitly means that in the bottleneck of the system, also called latent space, only the valuable information, describing the family of images used in training, is stored [14].

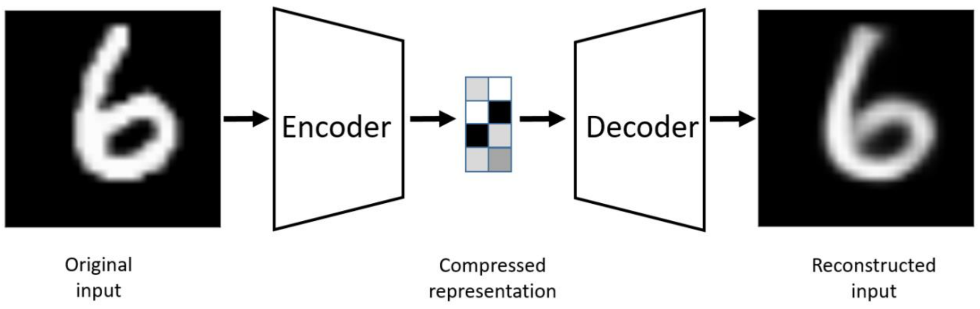


Figure 7 Structure of an autoencoder system. This kind of architecture is often used for anomaly detection

This consequently means that if a so trained system gets a new image in input, and that image is anomalous, the encoder does not have the tools to describe that anomaly in simple terms, since among the features it learned to extract the anomaly never appeared. Consequentially in the bottleneck the information about the anomaly is lost and so the reconstructed image will be similar to the starting one but without the anomaly. At that point it is needed to compare the original and reconstructed images and to score their differences to perform anomaly detection. In practice a threshold is put on this score and so any test image which has a score above that value is labelled as anomalous.

This simple architecture, and specifically the use of encoder and decoder, is recurrent also in many state-of-the-art algorithms of this class. Of course, each algorithm features some additional ideas, but it is important to keep in mind this scheme when describing the models that will be introduced in the following.

1.2.2.1 CFLOW-AD

CFLOW-AD consists of a pretrained encoder, for example ResNet18 pretrained on ImageNet, followed by a multi-scale decoder [9]. The encoder extracts features with multi-scale pooling to capture both global and local semantic information. These features are then processed by a set of decoders to estimate likelihood of the encoded features. The estimated likelihoods are up sampled to input size and added up to produce the anomaly map. The anomaly map is used to decide if an anomaly is present or not. This structure is represented in Figure 8.

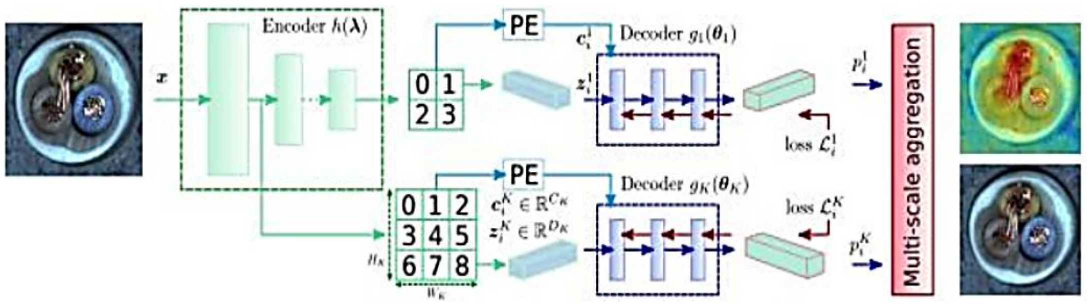


Figure 8 Structure of the CFLOW-AD model. In this case we still have the decoder-encoder logic but they work not only globally but also locally on some pixels regions

It is in fact very similar to the basic one described above. The main developments are that multiple positional encoders are used instead of a single one. These squeeze the information into a vector but work on different parts of the images [9]. Each of these vectors contains also harmonics specifying it spatial location and we have multiple encoders working on overlapping regions. Another difference is given by the way the decoders work. In fact, here conditional normalizing flow networks are used, and this is why the method is called CFLOW-AD. The conditional flow network is a statistical method used to model probability distributions which offers some mathematical advantages reported in [9]. In our application this method was chosen because it is one of the best performing for anomaly detection on the MVTec AD dataset [15] MVTec AD is the main dataset for benchmarking anomaly detection methods with a focus on industrial inspection [16].

1.2.2.2 PatchCore

This algorithm works on the principle that as soon as a single patch of the image is anomalous the entire image can be classified as anomalous. That's the reason it is called PatchCore. In practice, in a similar way to CFLOW-AD, you don't have and encoder working on the entire image at once, but the image is tiled and each of these sections is encoded separately with information about the position of the patch so to obtain locally aware patch features. These features are not squeezed as strongly as in classical encoders.

The authors of the paper state that this is helpful since lower-level features are generally too specific to ImageNet and therefore biased while too high-level features are too generic [10]. So the features are extracted from a mid-level of the backbone network used as encoder. This big quantity of data is down sampled trying to retain the information content. This operation is performed with a technique introduced in [17] called coreset subsampling. The data remaining at this point are to be found in what the authors call a memory bank.

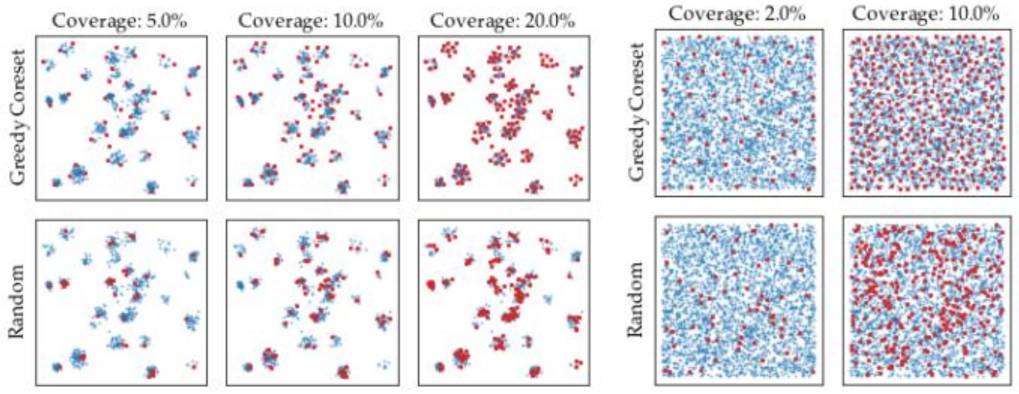


Figure 9 Examples of coreset subsampling (top) vs random subsampling (bottom). It can be seen that coreset better preserves the information

At this point another big difference with respect to the classical autoencoder algorithm appears. In fact here the features are not up-sampled anymore to get an artificial image to compare to the original one. In this algorithm the opposite happens. Indeed, an image to be tested gets treated with this same procedure so to obtain an information vector similar to the ones contained inside the memory bank. After that the anomaly score is simply given by a distance measured to the nearest neighbour in the bank.

In other words, here the image to be tested is squeezed to feature vectors and then the anomaly is obtained on these vectors instead of obtaining it at the pixel level. This procedure makes PatchCore able to achieve state of the art anomaly detection while considerably reducing the computational cost as reported in the original paper [10]. The structure is resumed in Figure 10.

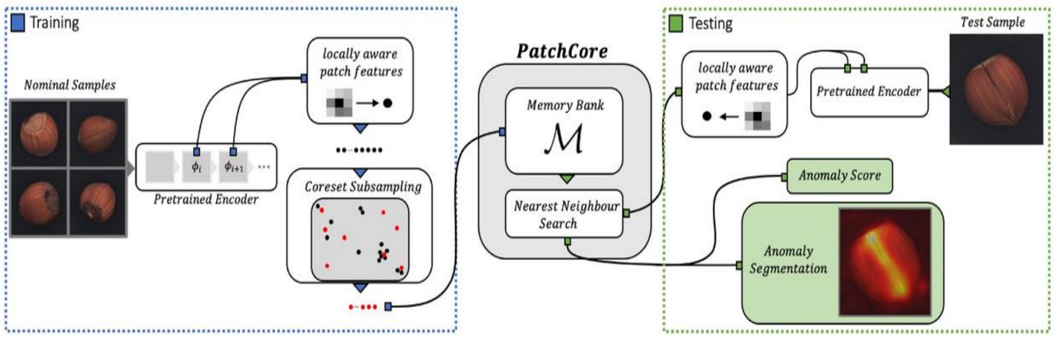


Figure 10 Structure of the PatchCore model. It can be seen that here the anomaly is calculated at vector level in the memory bank and not at image level

In our industrial application this method is chosen because it is the state-of-the-art method for anomaly detection on MVTec AD dataset[1].

1.2.2.3 Deep Feature Kernel Density Estimation (DFKDE)

This algorithm is made of two stages. The first one is very similar to the encoder part of an autoencoder. It consists of a deep feature extraction stage through a backbone like ResNet50 pretrained with ImageNet. In the second stage a probability density of these features is obtained. In fact, by fitting distributions to the deep features obtained during training, a generative model over the feature space is obtained [11]. Test images are then evaluated by simply comparing their extracted feature to the so obtained distribution. The characteristic of this model with respect to others is its use of gaussian kernel density estimation. This is a method to estimate probability density functions based on simple functions as weights. In that case these weight functions are called kernels. In Figure 11 we report a simple example of kernel density estimation with a symmetric kernel function on a four elements sample.

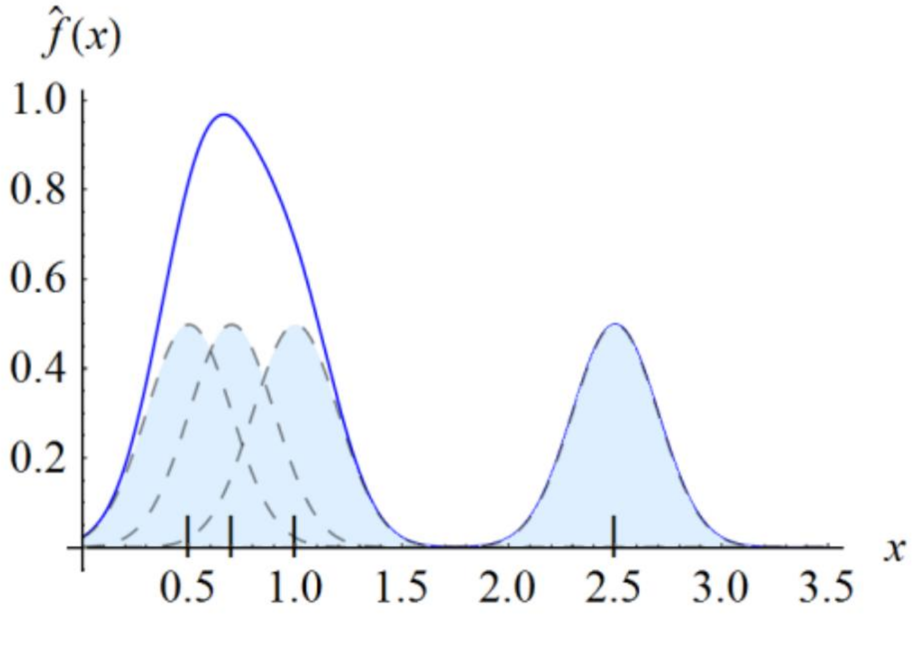


Figure 11 Example of kernel density estimation on a four elements sample. The blue areas are the kernels that are used as weights for the samples, while the blue continuous line is the overall estimation obtained with this kernel density estimation.

[1] https://paperswithcode.com/task/anomaly-detection

In our industrial application this method is chosen because it is one of the state-of-the-art methods encountered in the literature research and showed good results on the additional dataset that will be introduced in the following.

1.3 Dataset

After briefly presenting the models and their functioning, it is clear how a robust dataset in essential for the assessments. In fact, it is known that a sufficient quantity of data is fundamental when training a deep learning system [18]. To be precise, two datasets were provided by Raytec Vision S.P.A. These will be analysed as they differ in terms of number of images, image size and acquisition accuracy. Going deeper in the discussion some information about the main dataset among the two is given. In this first dataset the images of the tomatoes were acquired with a 5MP matrix-scan camera. The setup also consisted of a dome illuminator developed by the industrial partner which can be seen in Figure 12.

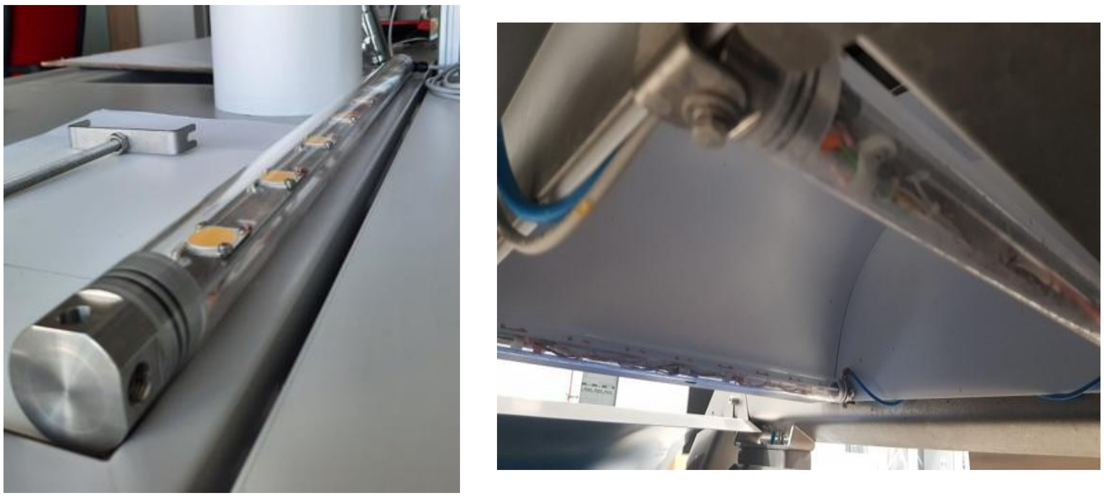


Figure 12 Dome illuminator used to acquire the dataset. It was developed by Raytec Vision S.P.A.

This Dome works emitting strobe light pulsing at 150 us and with 12 A of current. The integration time of the camera is kept at 180 us. The functioning of the acquisition system is the following. A RGB image of the tomatoes running on the tape is captured, see Figure 13. At this point, an algorithm of image detection is able to identify each tomato, as reported in Figure 14.

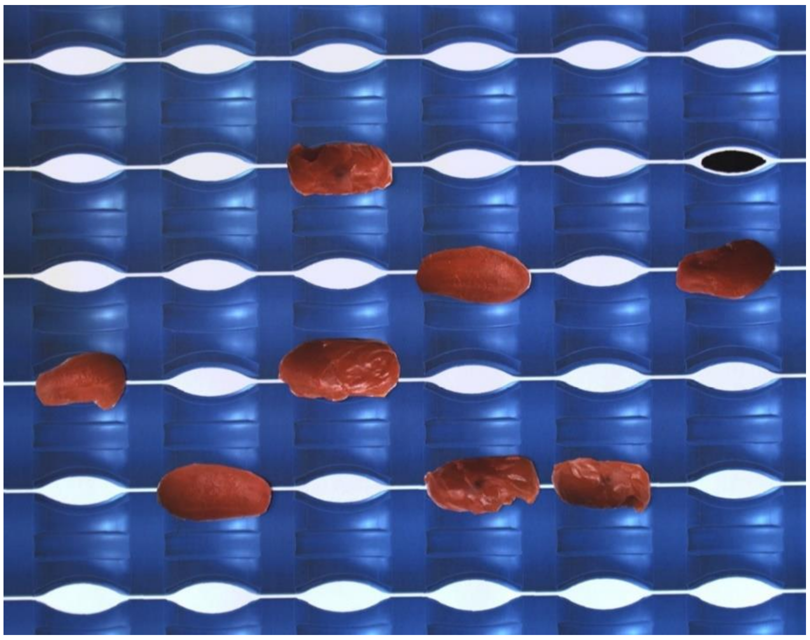


Figure 13 RGB image of tomatoes on the tape

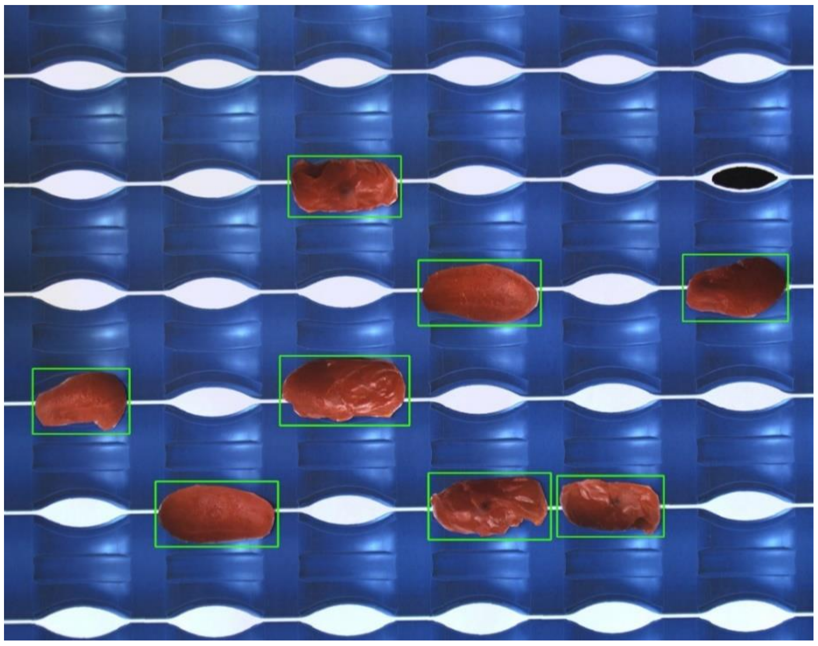


Figure 14 Image detection of the tomatoes

Once the tomato is captured, a mask is applied so that the background is automatically removed. The final result of the acquisition system is reported in Figure 15.

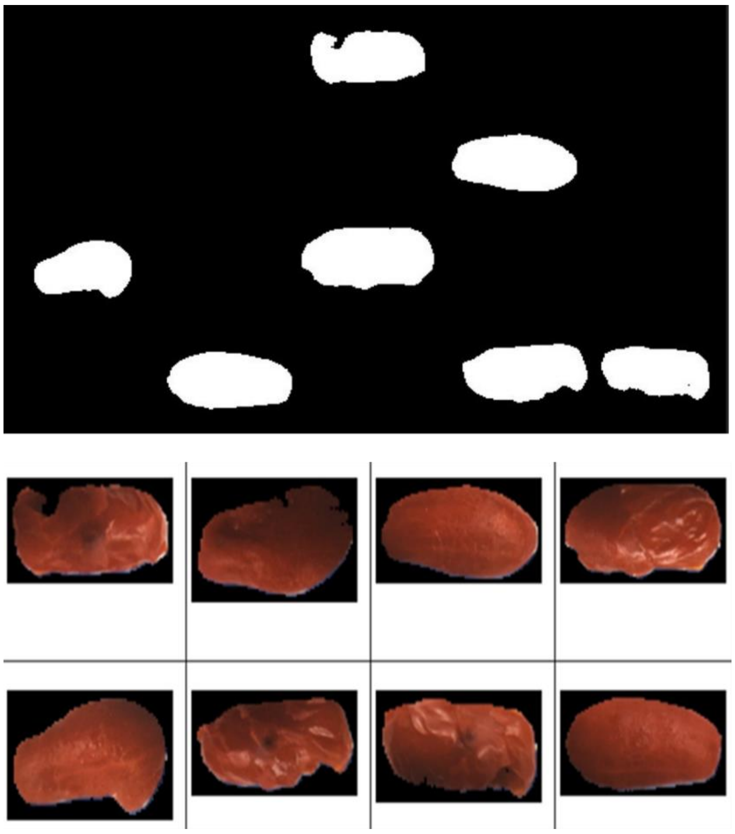


Figure 15 Top: binary mask used to remove the background. Bottom: final result of the process

The dataset is composed of 3370 images with the following distribution:

- 974 good samples of peeled tomatoes
- 1000 yellow samples
- 1000 green samples
- 396 samples with anthracnose or not correctly peeled.

The yellow and green samples also contain some images presenting anthracnose in addition, so that for example there are some tomatoes that should be sorted out both because they are not ripe (they are yellow or green) and because they have anthracnose. In Figure 16 we report an example for each category. As we are going to explain more in the detail in the following, some results are obtained working on the so structured dataset. But the main objective of the thesis is to obtain some software able to distinguish between good and bad data. Therefore, yellow, green and diseased tomatoes are grouped in one category named bad. The bad family contains 2396 images and a good samples folder containing 974 images. In the following, if not specified otherwise, the results reported will refer to experiments performed with this dataset in the specified configuration explained above.

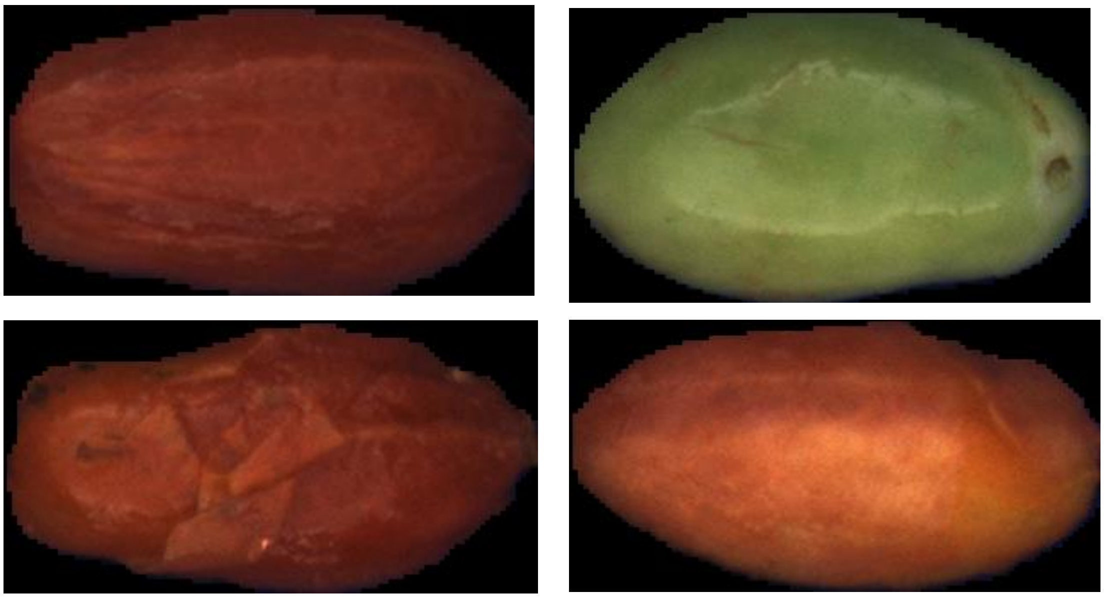


Figure 16 Examples from each category. Upper left: good peeled tomato; Upper right: green tomato; Lower left: diseased and not correctly peeled tomato; Lower right: yellow tomato.

The last thing to be noticed about this dataset, concerns the image size. Indeed, because of the image detection step, not all the images have the same dimensions. In particular, the resolution is in the range between 157*193 and 369*369 pixels. As we will see in the methods chapter, to work with our models, some modifications on the image size are needed.

Some examples of tomato images obtained with this configuration are reported in Figure 17.

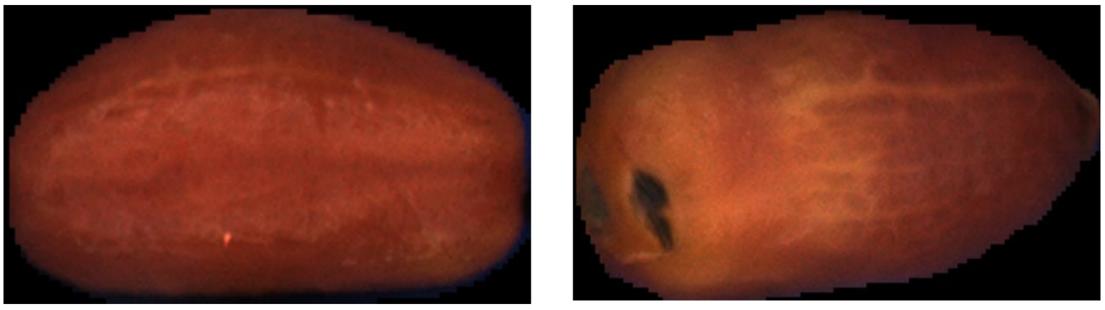


Figure 17 Example of a good and a bad tomato sample acquired with the here described setup

In addition to this dataset Raytec Vision S.P.A. provided also a second dataset. This was acquired in not so strict way. These images have a resolution of 12 MP and they were taken with a smartphone. In this case the background is not standardized and black, but it consists of a white table on which the photos were taken. Moreover, the lighting is not controlled. This dataset is composed by 252 good images and 228 bad ones. In this case the bad ones were just unpeeled samples so that the classification was just possible in the categories peeled vs unpeeled. In Figure 18 an example of these images is given.

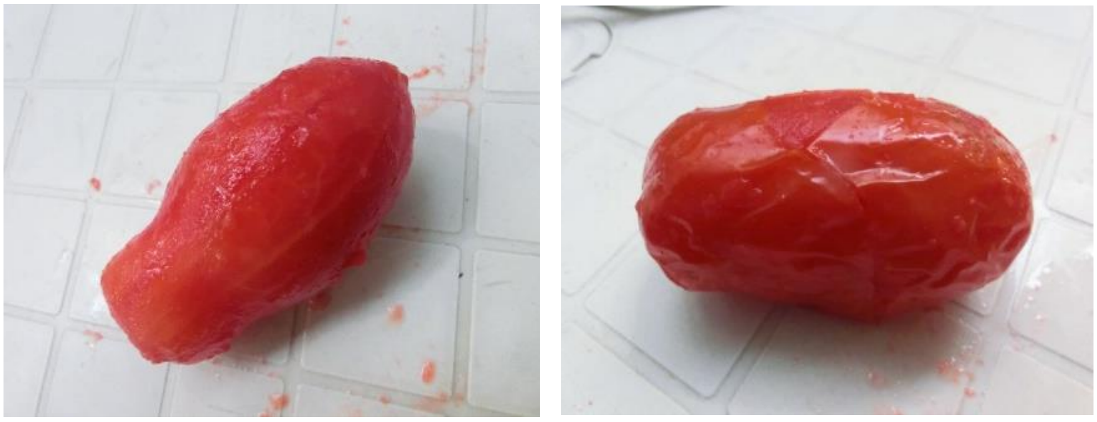


Figure 18 Examples of images from the second dataset: we notice these photos are taken without controlling the lighting and are just standard smartphone images

In the final part of the experiments section the possible effects of the presence or not of the background will be investigated through the application of a software able to simulate random backgrounds. Moreover, in this thesis work, driven by the small number of images in this dataset, also the effect and advantages of data augmentation will be handled.

1.4 Hardware and software settings

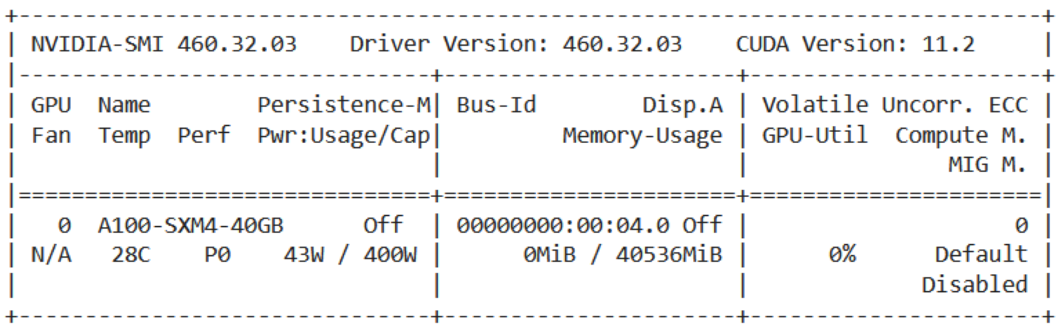
The training and in general the use and optimization of deep neural networks is a very computationally heavy procedure. This practically means that the experiments reported in the following are very time consuming even if Python (i.e., the open-source platform used for writing the software) is compatible with GPU calculations. For example, a single training run with ResNet50 on the main dataset can take up to days on a standard laptop computer, that has:

- 16 GB ram.
- Processor 11th generation Intel Core i7.
- NVIDIA T1200 laptop GPU.

Figure 19 The use of Google Colab and Drive is fundamental to perform some of the experiments which wouldn't run on personal computers because of the high computational request

Moreover, the experiments reported in the following feature hundreds of training runs. For this reason, the use of a powerful hardware is of crucial importance. Said that, the hardware chosen is a cloud computing platform. In particular, Google Colab is chosen for this purpose. This platform permits to access computationally strong resources trough cloud. It must be remembered that also the RAM requirements for the work here presented are quite high meaning that some experiments could not run without the support of the 64 GB of ram given by the cloud. Anyway, the most important support is given by the GPU acceleration. In particular as reported in Figure 20 the use of Google Colab permitted to take advantage of a Nvidia A-100 GPU.

To use TensorFlow® [19], needed for the classification, with this kind of support the CUDA® [20] Deep Neural Network library (cuDNN) [21] is installed. For what concerns anomaly detection, Pytorch® [22] is additionally configured to run the training instances on GPU.



```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  A100-SXM4-40GB      Off  | 00000000:00:04.0 Off |                    0 |
| N/A   28C    P0    43W / 400W |      0MiB / 40536MiB |      0%      Default |
|                               |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
```

Figure 20 Hardware settings of the cloud computation used. Notice the Nvidia A100 GPU.

In the possible future industrial application installing a GPU with performances similar to this one can be imagined. This is useful since in the following also some considerations on the possible applications of the models and their integration with existing machinery will be reported and having such a strong hardware which could resemble the industrial one in the final machine permits to make some considerations in this direction.

1.5 Thesis Outline

In this section the main points analyzed in this thesis document are explained. The work is divided essentially in five main chapters. In the second chapter (i.e., Methods) it is shown how the models adapted for the specific application were obtained starting from the algorithms in general form. The general framework adopted to implement, optimize and compare the models is reported. Even if each of them has peculiar differences in the implementation, which are reported in detail in the methods chapter, a black box scheme that was then followed to standardize their use as much as possible is introduced.

The third chapter reports the results. The numerical outcomes of the optimization and of the experiments performed to compare models are shown. These results are also briefly commented. This is done to help the reader follow the logical scheme they were produced with. In the second part of this section, there are additionally reported some results obtained for multiclass classification and on a different dataset or with different backgrounds to show the robustness of the obtained system and to give some insights on possible different applications of the classification algorithms.

Finally, there is a conclusions chapter. Here a summary of the bullet points of this works is reported. The main insights are summarized, and the results are commented in a broader way, offering some suggestions for future developments.

2 Method

In this section the workflow followed to reach the final software configurations and the experimental procedures used to optimize and compare the different models are explained. This method chapter is composed of six main parts that can be briefly summarized as follows. Firstly, the general framework that was set up to handle the different experiments in a common way is described. Then, in the second subsection, it is described how the classification architectures were implemented in this framework. In the third section the same is done for the anomaly detection models. Then, in the fourth subsection, the metrics considered to evaluate the performances of the algorithms are illustrated. These are mandatory to be able to evaluate and compare the results gathered from the different techniques and models. In the fifth subsection, the models' optimization method is reported. Indeed, once the workflow as well as the metrics are identified, the need to obtain the best possible algorithms must be fulfilled. In the last section of this methods chapter, the procedure used to compare the models is explained. For this reason, K-Fold cross validation, Almost Stochastic Order and bootstrap power analysis are introduced and described.

2.1.1 Black box scheme

As stated above the first step to describe the following experiments is to introduce the logical pattern they were performed with. In particular, the objective was to take the six models described in chapter one, to adapt them so that they could work on the dataset presented, and to implement a software that could run the models in a common framework.

In other words, the objective is to have one black box architecture able to handle all the different models. In that kind of structure, each black box represents the adapted model that will be applied on the dataset. In this framework, there are images coming into the black box and sorting results coming out. This also means that the objective is to write code strongly based on function calls and in which all the variable parts are contained in external configuration YAML files. The different models are in the end implemented by solely changing the configuration files.

With this standardised way of proceeding clearly in mind, it becomes easier to optimize the models with a common method. This initial work of standardization also helps in comparing the models, speeds up the experiments and facilitates the readability of the following sections of this thesis.

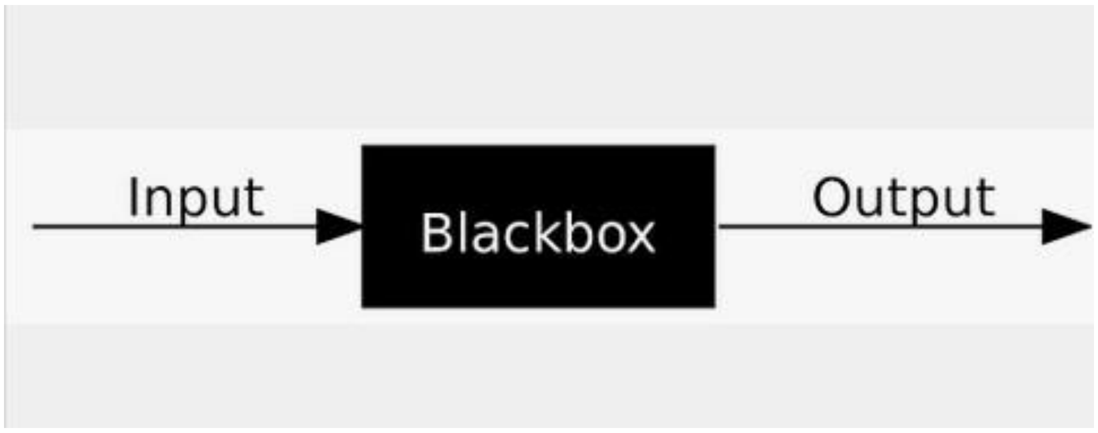


Figure 21 General framework of the adopted method: Images are the input to the specific model used which acts as a black box and gives the sorting result as output

2.2 Models

From now on it is necessary to split the discussion, distinguishing between anomaly detection and classification algorithms. From chapter 2.3 on, methods of both families will be treated as black boxes. Here, in subsections 2.2.1 and 2.2.2, some information about the work done to obtain this framework starting from the different models is given.

2.2.1 Classification

The main library used for the classification task is TensorFlow which, since the release of its second version (i.e., TensorFlow 2), is perfectly integrated with Keras. TensorFlow 2 is an end-to-end, open-source machine learning platform, while Keras is the high-level Application Programming Interface API for TensorFlow 2. Keras is a deep learning API written in Python, running on top of the machine learning platform Tensorflow.

This setup is chosen for two main reasons. The first reason is that in Keras all the algorithms considered are built-in functions. The second reason is that, using Tensorflow, those neural networks can be downloaded in their own ImageNet pretrained version. This is good for two main reasons:

1. There is no need to train the neural networks from scratch. This is time and computational effort saving.
2. Using ImageNet pre-trained CNN features, impressive results have been obtained on several image classification datasets [23] by means of the so-called Transfer Learning (TL) process.

From now on, all the necessary steps to do image classification are explained in detail. The first one is to upload the dataset. The whole dataset is divided into three sub-folders named “training”, “validation” and “test”. This procedure is done by means of simple code able to split the dataset according to the percentage of images to be used for training, validation and test. This percentages are 70, 15 and 15 respectively. The first two folders will be used during the training phase, while the third one, consisting of images never seen by the model, will be used to evaluate the model’s performance in making predictions. This is a common practice in data science. The following step is to upload those folders into the program. In this passage an image size must be set because each model expects a well-defined input image dimension. For all the models, the image size is set to 224*224 pixels. Such an image size is chosen as standard value as this specific value is commonly used and the images given by Ratech Vision S.P.A. have a similar resolution.

After uploading the dataset, the construction of the model should be discussed. Since all the models to be compared are given in their pretrained version, the procedure here performed is commonly known as TL. This consists of taking features learned on one problem and leveraging them on a new, similar problem. Transfer learning in the context of deep learning follows a common workflow[2]:

1. Take layers from a previously trained model.
2. Freeze them, to avoid destroying any of the information they contain during future training rounds.
3. Add some new, trainable layers on top of the frozen layers. They will learn to turn the old features into predictions on a new dataset.
4. Train the new layers on your dataset.

That said, for the specific case analyzed, it is not possible to import the pre-trained models and directly pass to the training phase, but some modifications are needed. As explained in points 1 and 2, it is needed to import most of the layers of the pretrained models, freezing them. In particular, both first and last layer, which are fully connected layers, are not imported.

Following the bullet point list, it is needed to replace some of the layers: the first one is substituted with an input layer where dimensions of the images (i.e., 224*224 pixels) and number of channels (i.e., 3 channels since we work with RGB images) are specified. A so called "preprocessing layer" is added to follow the previously mentioned one. Its function is to apply a series of transformations to make the images' format equal to the one used during the pre-training procedure with ImageNet dataset. For the last layer the discussion is a bit more complicated. In fact, it is not sufficient to replace one layer with another one, but it is needed to add:

- A flatten layer. It takes the pooled feature map and flattens it into a column, as shown in Figure 22. After the flattening step, a long vector of input data is obtained and passed to the artificial neural network for further processing.

[2] Transfer learning & fine-tuning (keras.io)

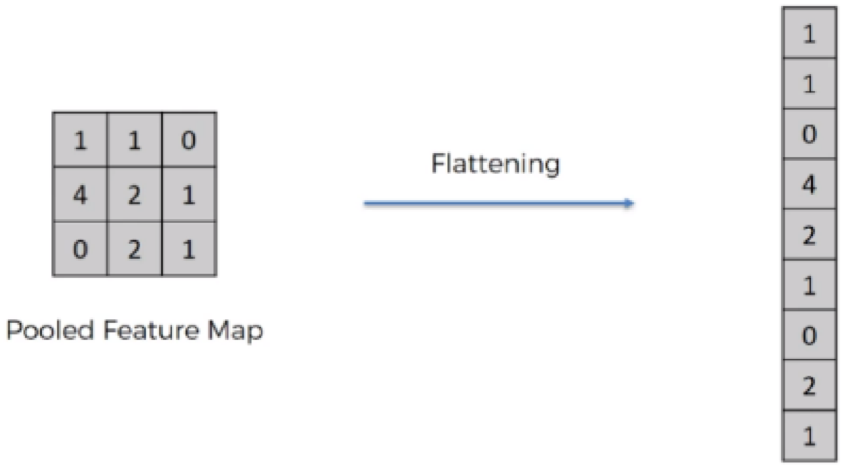


Figure 22 How the flatten layer works

- A dense layer composed by units in the form of Rectified Linear Units (ReLU), as the one reported in Figure 23. The dense layer's neuron in a model receives the output from every neuron of its preceding layer. This layer is the one responsible to learn new weights.

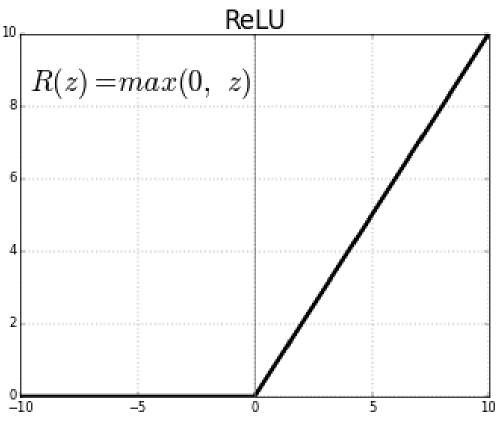


Figure 23 ReLU activation function

- Finally, the last layer, which must have the number of units equal to the number of predictions classes (i.e., two units in the case of binary classification). For this purpose, Sigmoid activation functions are chosen. But how do these functions work? Let's image that we have two classes a and b for example. The two classes are associated with the values 0 and 1 (for illustration purpose, a is associated to 1 and b is associated to 0). The sigmoid, relying on a threshold, converges the probability at one of its extremes. Therefore, if the probability that a specific image belongs to class a is greater than the threshold the value 1 will be assigned.

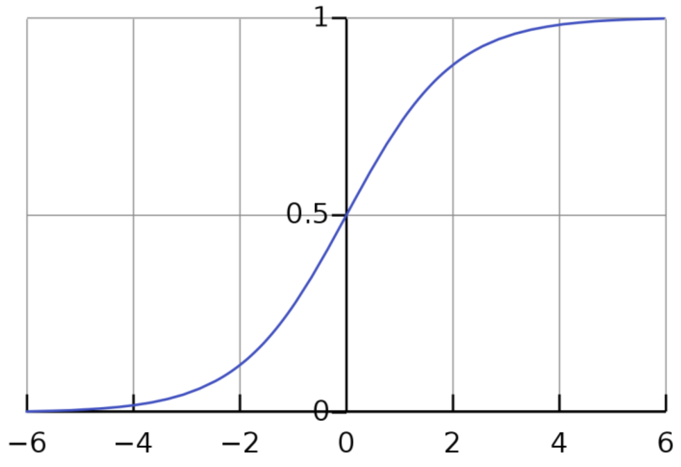


Figure 24 Sigmoid activation function

To make what has just been described regarding the model's construction clearer Figure 25 is reported. By looking at the figure it is possible to appreciate:

- Input layer.
- Preprocessing layer, composed by the union of 'tf.math.truediv' and 'tf.math.subtract'.
- The pre-trained model (MobileNet in the picture).
- The flatten layer.
- The dense layer composed by ReLU activation function.
- The dense layer composed by Sigmoid activation function.

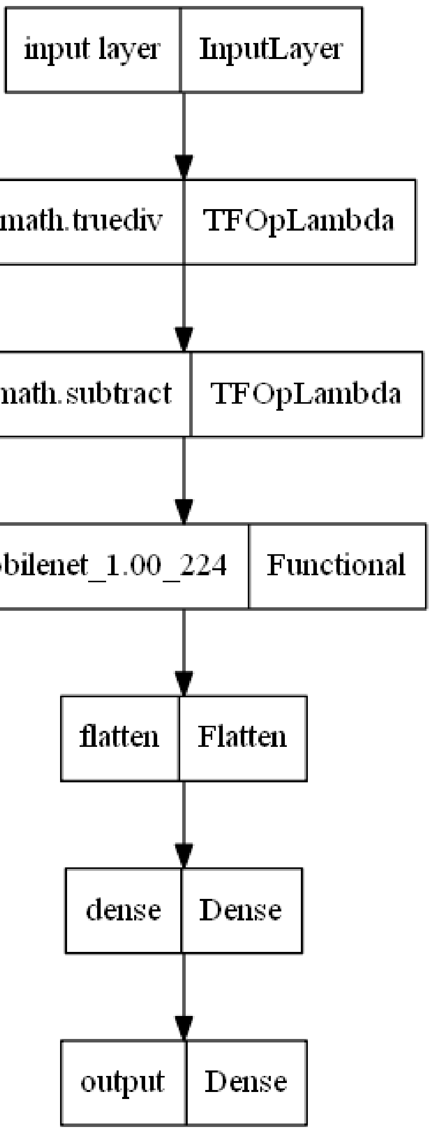


Figure 25 Visualization of the here presented architecture using Mobilenet as pretrained model for the example

Once the model is built, the next step is the 'compilation'. In this step some parameters that characterize the network must be defined. As it is reported in detail in the dedicated chapter, these parameters will be subjected to the so-called hyper-parameters' optimization procedure (see chapter 2.4 on hyper-parameters optimization). In particular, both optimizer and learning rate need to be defined. For this purpose, the Stochastic Gradient Descent SGD algorithm was chosen as the optimizer. The momentum is fixed to be equal to 0.9, while the value of the learning rate will be decided by means of the optimization procedure. After that the categorical cross-entropy is defined as loss function. Without going into too much detail, it is important to give at least a definition of the above-mentioned parameters.

The purpose of training a neural network is finding weights and biases so that the output from the neural network approximates the desired output y(x) for any training input x. The loss (or cost) function quantifies the degree to which the objective is achieved. When the loss function is large, y(x) is not close to the output for a large number of inputs. Therefore, the aim of the optimizer is to minimize the loss function. In other words, we want to find a set of weights and biases which make the cost as small as possible [2]. What does the optimizer do? To explain the concept in a simple way, it is useful to imagine the function as a kind of valley and a ball rolling down the slope of the valley. Now, some sort of law of motion must be found that makes the ball always roll towards the bottom of the valley.

In a nutshell, the gradient descent works by repeatedly calculating the gradient of the cost function and then moving in the opposite direction, "falling down" the slope of the valley, as shown in Figure 26.

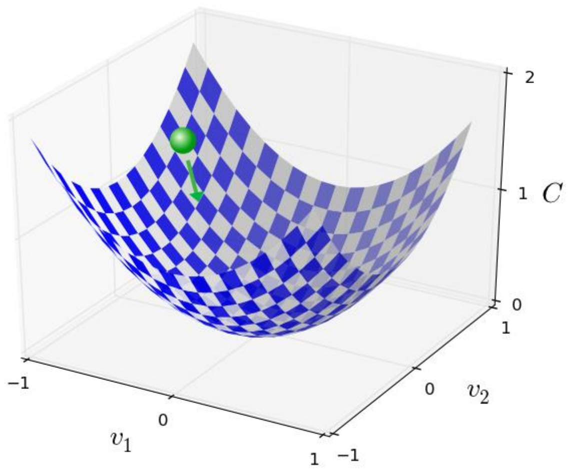


Figure 26 Cost function C as a function of 2 variables v_1 and v_2

In particular, the SGD estimates the gradient of the loss function by computing the gradient for a small sample of randomly chosen training inputs. By averaging over this small sample, it turns out that it is possible to quickly obtain a good estimate of the true gradient. This helps speed up the learning.

The learning rate governs the size of the steps to reach the minimum. There are two main points to consider when choosing the value of the learning rate:

1. If the learning rate is too large, then the steps will be so large that they may overshoot the minimum.
2. Choosing the learning rate too small slows down the SGD algorithm.

As reported by Nielsen et al. [2] the momentum method introduces a kind of inertia term. To understand the way the momentum influences the SGD, it is useful to take a look at Figure 27.

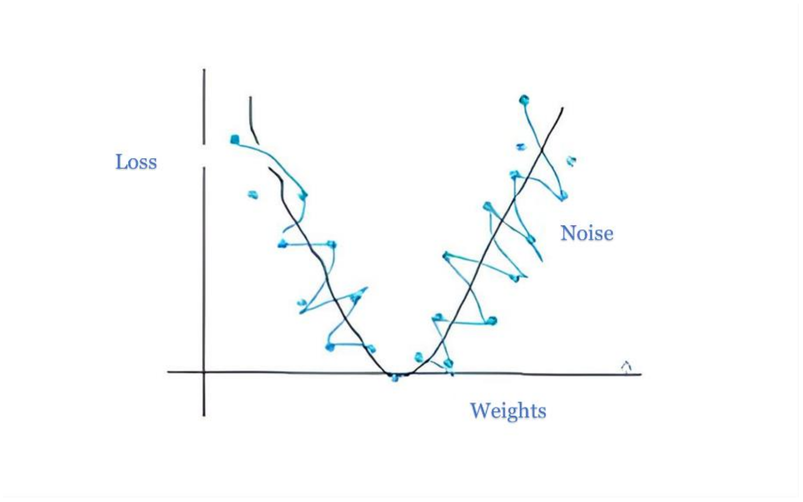


Figure 27 SGD without momentum

The noise and random fluctuations reported in the picture are due to the small batches used by the SGD to compute the derivative of the cost function. The goal of the momentum is to give a more stable direction to the convergence of the SGD. The momentum is used to weight the gradient of the next iteration by introducing the gradients of past iterations into the update function of weights and biases. Thus, improvised variations do not affect the gradient so much. It is possible to appreciate the result in Figure 28.

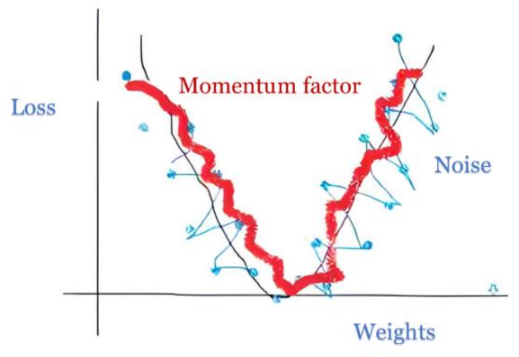


Figure 28 SGD with momentum. The momentum is reducing the fluctuations in the weight updates

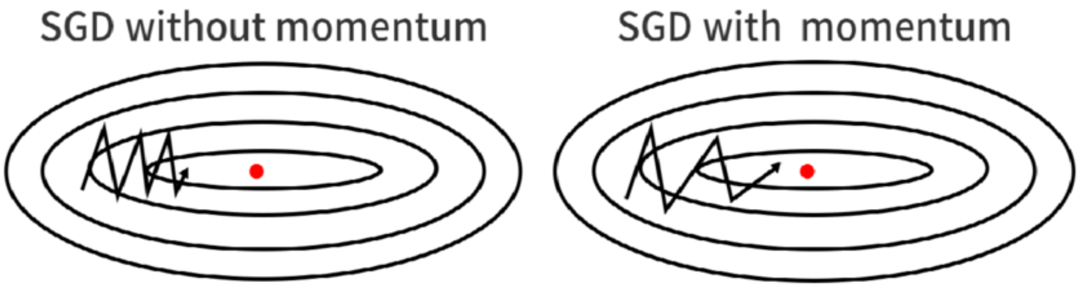


Figure 29 Comparison of SGD algorithms. Left: SGD without momentum. Right: SGD with momentum

In the end, there are two main advantages in using the momentum technique[3]:

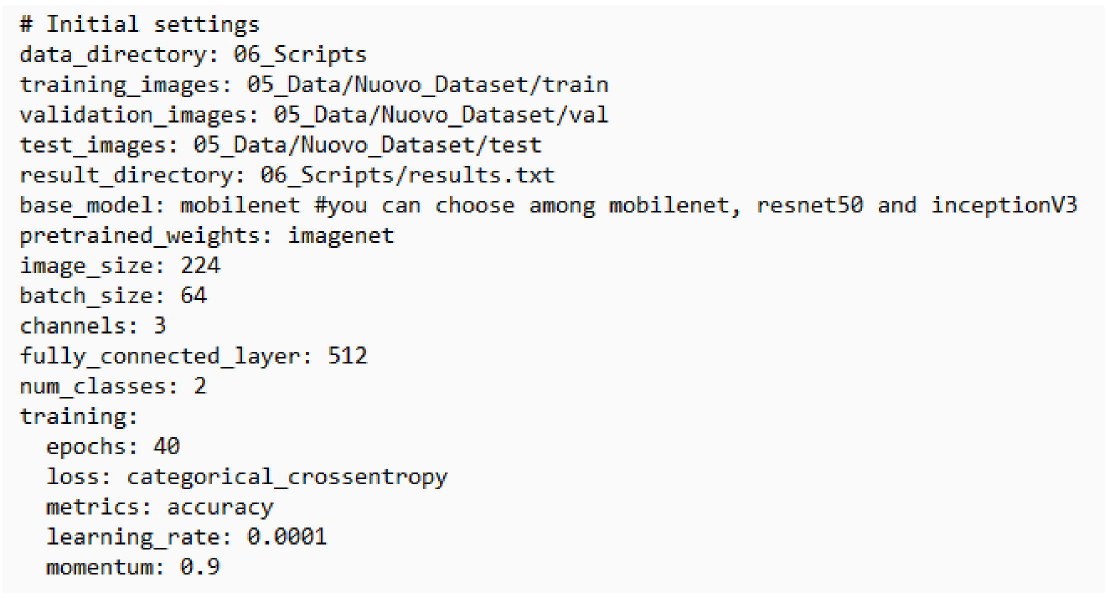
1. SGD with momentum is faster than standard SGD. Therefore, the training will be faster.
2. Global minima can be reached without stopping at local minima due to the momentum involved.

Now that those parameters are clearer, it is possible to continue with the explanation of the model. One important drawback during the training phase of CNNs is the overfitting problem. To avoid that a call-back is defined. Thanks to the this function, it is possible to follow the trend of an arbitrarily chosen parameter during the training of the model. If the monitored parameter does not improve for a number of epochs set by the programmer, the training is stopped and the model with the best parameter value is saved (the same can be done whether the parameter remains constant or decreases). In the end, this procedure allows to avoid overfitting problems. For the case analyzed here, it is decided to follow the trend of the validation accuracy and stop training if it does not increase by 0.001 for a number of epochs equal to 10% of the total number of training epochs. Moreover, thanks to this function, the best model (i.e., the model with the best classification accuracy in this case) of the training procedure is automatically saved. In this way the best model among the ones achieved in training is certainly used, during the test procedure.

To summarize what has been said above, the so developed software is able to upload the dataset, pick up the starting model to be used, modify it as it is needed, compile it and train it. When the training procedure is complete, it saves the best model so that it is possible to test the latter on a new folder of images. The entire procedure can be implemented simply by acting on the

[3] SGD with Momentum Explained | Papers With Code

configuration file previously mentioned. One example of configuration file is reported in Figure 30.



```
# Initial settings
data_directory: 06_Scripts
training_images: 05_Data/Nuovo_Dataset/train
validation_images: 05_Data/Nuovo_Dataset/val
test_images: 05_Data/Nuovo_Dataset/test
result_directory: 06_Scripts/results.txt
base_model: mobilenet #you can choose among mobilenet, resnet50 and inceptionV3
pretrained_weights: imagenet
image_size: 224
batch_size: 64
channels: 3
fully_connected_layer: 512
num_classes: 2
training:
  epochs: 40
  loss: categorical_crossentropy
  metrics: accuracy
  learning_rate: 0.0001
  momentum: 0.9
```

Figure 30 Example of configuration file in YAML format. Trough the black box scheme implemented the entire construction and deployment of the model depends only on this file

2.2.1.1 Gradient-weighted Class Activation Mapping (Grad-CAM) visualization

In order to ensure every IC-based model works correctly, the Grad-CAM method was applied to each of them at the end of the training, as proposed in the original paper [24]. The outcomes of this method are reported in the results section. In this paragraph, only the method with which Grad-CAM was implemented is discussed.

Grad-CAM is a technique used to visualize which regions of the input are “relevant” to perform the prediction. In practice with this method, the different regions of an image are coloured depending on how much the network focuses on them to classify the image. An example of this is reported in Figure 31. It is clearly possible to see that the neural network gives more weight to the tomato when taking the decision than to the background. This indicates the algorithm is focusing on the correct part of the image showing that it is working in the correct way.

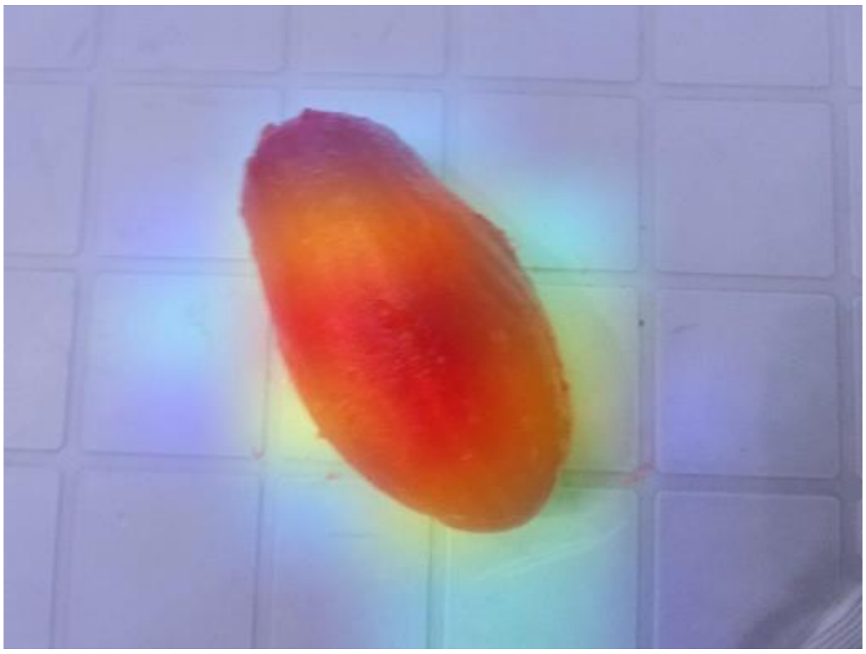


Figure 31 Example of heatmap obtained with ResNet50 applying Grad-CAM

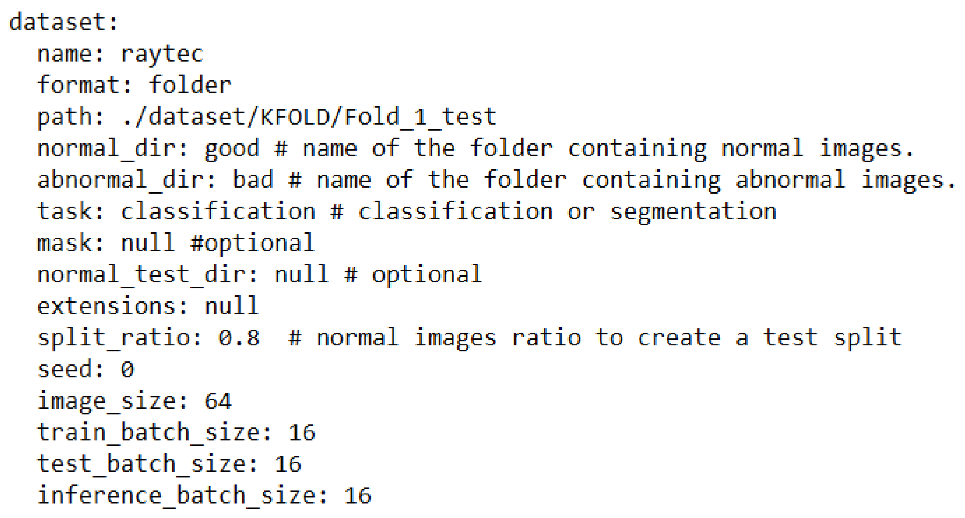
The working principle is illustrated below. The last convolutional layer of the selected model is identified. Then the model is fed with an input image and the prediction is returned. After that, the gradient of this prediction with respect to the output of the selected layer is taken. This is the information to be obtained. As the next step, the information contained in the gradient is simply scaled with respect to the size of the input image, displayed in terms of different colours and superimposed on it. In the following, this tool will be used for each model to check that it is looking in the correct areas of the image and therefore that it functions correctly.

2.2.2 Anomaly detection

For anomaly detection with deep neural networks the main tool used is a GitHub repository called Anomalib [11]. This is an open-source library developed by Intel that provides implementations of many state-of-the-art anomaly detection algorithms. The library is particularly suited for image-based anomaly detection, where the goal of the algorithm is to identify anomalous images. It is the largest open-source library of this type, and the authors suggest that in future it could also contain algorithms for the audio or video domain [11]. In particular, it contains all models mentioned above.

Figure 32 ANOMALIB is the most important library used in this thesis for deep learning anomaly detection

The repository is already based on the use of a single configuration file operating with a huge number of function calls. As a result, the creation of the common pipeline is straightforward in these cases. It is just needed to modify the configuration file so to adapt to the custom dataset. After that the anomaly detection case can be treated in the same black box manner developed for the classification algorithms. An example of such configuration files modified for the task of this thesis work is reported in Figure 33.



```
dataset:
  name: raytec
  format: folder
  path: ./dataset/KFOLD/Fold_1_test
  normal_dir: good # name of the folder containing normal images.
  abnormal_dir: bad # name of the folder containing abnormal images.
  task: classification # classification or segmentation
  mask: null #optional
  normal_test_dir: null # optional
  extensions: null
  split_ratio: 0.8  # normal images ratio to create a test split
  seed: 0
  image_size: 64
  train_batch_size: 16
  test_batch_size: 16
  inference_batch_size: 16
```

Figure 33 Example of custom configuration file for anomaly detection

In the introduced way of operating with black boxes also the use of the Torchmetrics® [25] module was very useful. Through this library it is possible to extract the necessary metrics by slightly modifying the code contained in the repository without the need to hard code their calculations from scratch, as done for the classification.

2.3 Metrics

In the previous section, it was described how all the different models were fitted in a common framework. This is useful since a big part of the thesis objectives' is to compare the various models implemented so to extract some insights and identify the best one for the required industrial application.

In order to find the best model, it is necessary to identify some common metrics to measure the performance of the different models. The objective is to obtain a robust classifier which must be integrated with already existing industrial tools such as conveyors. In that sense not only the quality of the decision about a single item passing under the computer vision system is important. Indeed, other factors play a considerable role as well. This chapter summarises the chosen metrics and the reasoning behind the choices.

2.3.1 Inference Time

In the industrial application here presented, a possible limitation is given by the decision-making speed of the model. As already seen in the dataset description, the images are acquired by the pipeline in a small resolution, not exceeding 369*369 pixels, mainly for computational reasons. Still the speed of elaboration plays a huge role in the industrial reality and customers are often willing to sacrifice some quality in the picking process to gain speed. In the industrial scenario these models are working, GPU usage is given for granted. A hardware setup similar to the one used here is realistic, as confirmed by the industrial partner. It is very important to consider that, choosing a model over another just basing on correct picks could result in drastically reducing the quality control output rates. In fact, time is a limiting resource even using strong and modern hardware in this and similar applications. For this reason, a metric which is used in the following to evaluate and compare the models is the inference time. The inference time was computed with respect to a single image, to understand how long each model takes to classify a given picture. At the end, a comparison of each model according to inference time will be made.

Apart from that there are no big limitations, and the performances of the various models can be evaluated by looking at the quality of the choices.

To evaluate that, two metrics are used:

- Accuracy
- Area Under the ROC curve (AUROC)

Training time, for example, is not a limiting resource for the industrial partner, since even in the worst scenario the training phase of the heaviest among the models considered takes up to 10 hours, which is not a problem when selling the machine, considering the total set-up time of it.

Figure 34 Inference time is chosen as a metric to compare models since it is an industrial application integrated with conveyors

2.3.2Accuracy

Accuracy is the most used metric in classification problems. It is a very intuitive and easy to extrapolate metric. It is the number of correct predictions divided by the total number of predictions. Accuracy is the universally most cited and widespread metric [26]. Anyway, it has some disadvantages. The main one is that it hides the issue of class imbalance. For example, if the data contains only 3% of negative instances, a classifier which always assigns the positive label would reach 97% accuracy.

2.3.3 Area Under the Receiver-Operating Curve (AUROC) and Confusion Matrix

It has been rigorously stated [27] that AUROC should replace accuracy when measuring and comparing classification models. For this reason this more technical metric which is not as intuitive as accuracy should be introduced.

For the binary classification problem this thesis focuses on, it is possible to visualize the results in a so-called Confusion Matrix (CM), like the one reported in Table 1. A CM is a table that allows the visualization of the performance of an algorithm. The matrix is organized in such a way that each row represents the instances in the actual class, while each column represents the instances in a predicted class. For the case in point here, a high value of false positives means that many bad tomatoes were classified as good. Conversely, a high value of false negatives means that many good tomatoes were discarded.

Table 1 Example of confusion matrix to introduce the AUROC

		Predicted class	
		Class 1	Class 0
Actual class	Class 1	10 True Positive (TP)	2 False Negative (FN)
	Class 0	3 False Positive (FP)	35 True Negative (TN)

In this kind of matrixes, the True Positive Rate (TPR) is defined as:

$$TPR = \frac{TP}{TP+FN} \quad [3]$$

While the False Positive Rate (FPR) is defined as:

$$FPR = \frac{FP}{FP+TN} [4]$$

As this thesis focuses on binary classification, the algorithms predict an image as class 1 or 0 based on a threshold (i.e., if the prediction is higher than the threshold the algorithm assigns class 1 to the image and the contrary if the prediction turns out to be lower than the threshold). With this fixed in mind, changing the numerical value of this threshold causes a change in both TPR and FPR. The next step is to introduce the Receiver Operating Curve ROC. The ROC is built changing the value of the threshold, in the range 0-1 for the specific case analysed. More in detail, TPR and FPR are computed for any threshold's value and each couple of values (FPR, TPR) represent a point of the ROC. Since the models output a probability and since the classification is binary, this probability will be in the range 0-1, as explained above. By drawing these points in a graph that has FPR on the x axis and TPR on the y axis it is possible to obtain the ROC curve.

In the ROC curve the threshold value is implicit and once constructed there is no way to extrapolate the threshold value corresponding to each couple (FPR, TPR) just by looking at the graph. As can be observed in Figure 35, the better the classifier the more ROC tends to a graph with a 90 degrees angle in the point of coordinates (0,1). This qualitative observation of the behaviour of the ROC can be exploited in a more mathematical form by using the so-called AUROC parameter.

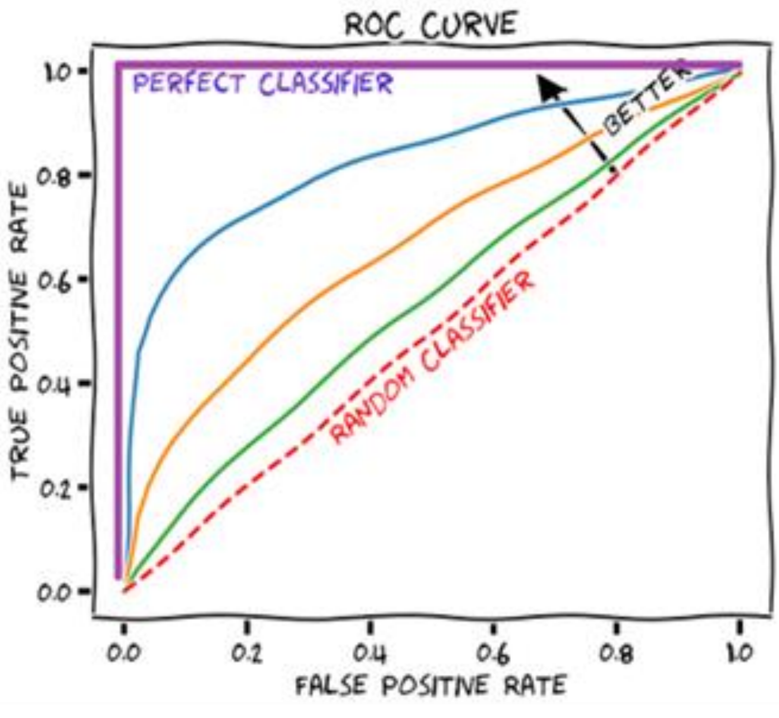


Figure 35 How to qualitatively evaluate ROC curves

In fact, the AUROC is just the integral of the ROC curve (i.e., it is just the area in blue in the example reported in Figure 36). In other words, the higher the value of the AUROC the better the classifier. In particular:

1. A perfect classifier has an AUROC equal to 1
2. A random one has an AUROC equal to 0.5.

3. Any value above 0.9 is considered excellent [28].

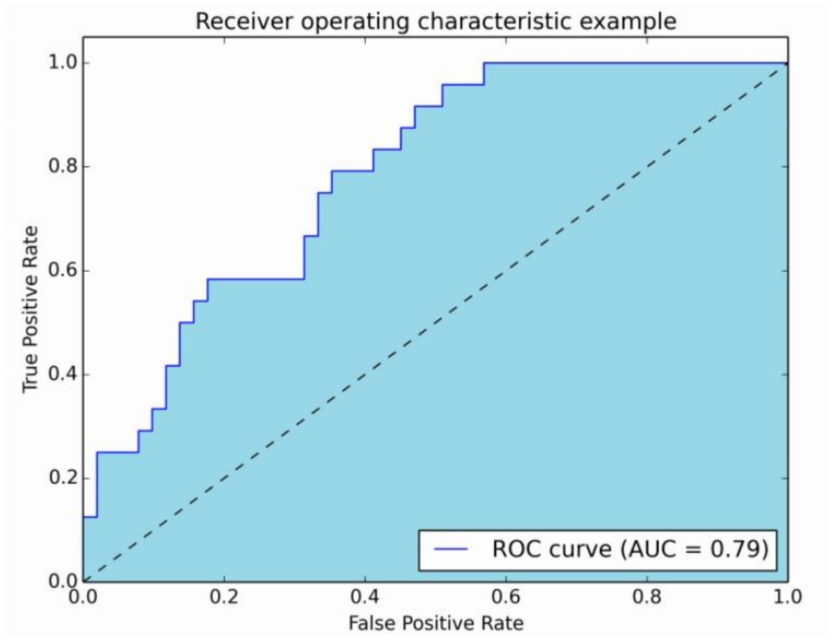


Figure 36 Example of ROC curve with area under the curve in blue

2.4 Hyperparameters Optimization

Hyperparameters are those parameters which are set before the training process of a machine learning algorithm. They cannot be learned by fitting the model to the data. They can be involved in building the structure of the model, such as the number of hidden layers and the activation function, or in determining the modalities in which the training phase happens, such as the learning rate (LR), the stochastic gradient descent (SGD), batch size, and optimizer [29]. Their values clearly have a strong influence on the learning process and thus they have an impact on the overall performance of the model, therefore they should be treated carefully. Choosing them is a key factor for this work. The first approach which is widely used consists in: review of values chosen by others in literature, consultation of dedicated forums and a trial-and-error procedure. This is also the path here followed to identify a first range of possible hyperparameters' values. However, these are not necessarily optimal and therefore a tuning procedure is required. In this section, the method followed to perform the hyperparameters optimization (HPO) is reported. To this end, WandB® website is used. This is an online tool that allows to run training and validation automatically a pre-defined number of times on the chosen model, with the aim of finding the best possible combination of hyperparameters. This is done as follows: after every run the program changes the value of each hyper-parameter in a range priorly defined by the user. This change is performed following a Bayesian scheme in order to optimize a certain metric that has to be chosen before the process starts. In the following the accuracy is used as objective function and the goal is to maximize it. As reported in [30], Bayesian optimization is a sequential model-based method aimed at finding the global optimum with the minimum number of trials. So, compared to other search methods such as Grid or Random, Bayesian optimization can find the best hyper-parameters set with fewer attempts.

Weights & Biases

Figure 37 Software used for hyperparameters optimization

Again, a black box architecture is followed in this HPO. In this case, the input is a configuration file containing the range of values in which the search for optimal hyperparameters has to be performed, while as output we get the optimized value for each of them.

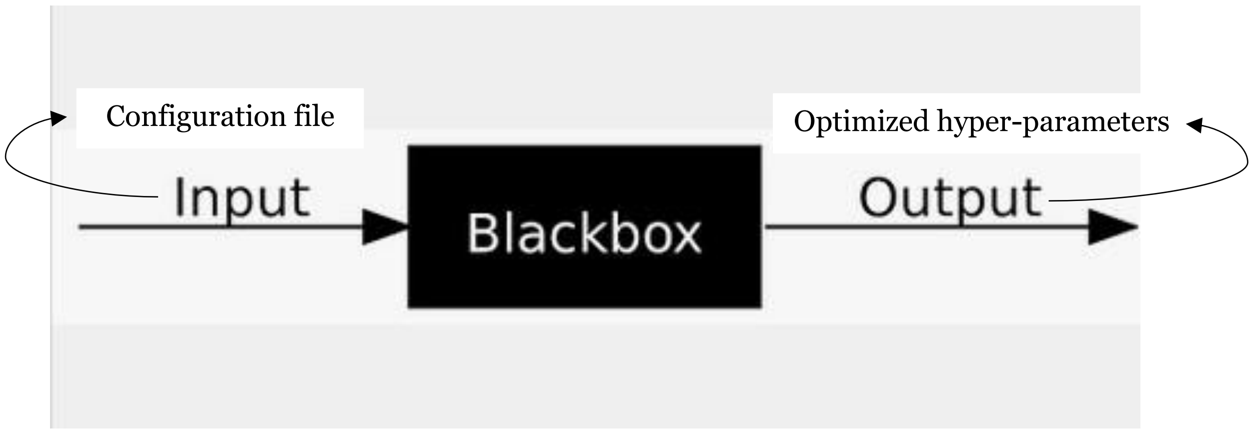


Figure 38 Black box architecture for the HPO procedure

As for the black box scheme introduced for training, validation and testing, also this procedure for HPO is strongly based on function calls. Again, it shows its strongest advantages when working with different models. Indeed, thanks to the fact that all the variables are contained in a configuration file, it is only needed to change this file to perform HPO with the same procedure on all models. This initial work of standardization speeds up the experiments and helps the readability of the following sections of this thesis.

As done for some previous sections, following this introduction about the hyperparameters tuning method used in general we split the discussion. Firstly, the details for the image classification HPO are reported and after that the ones for the anomaly detection case.

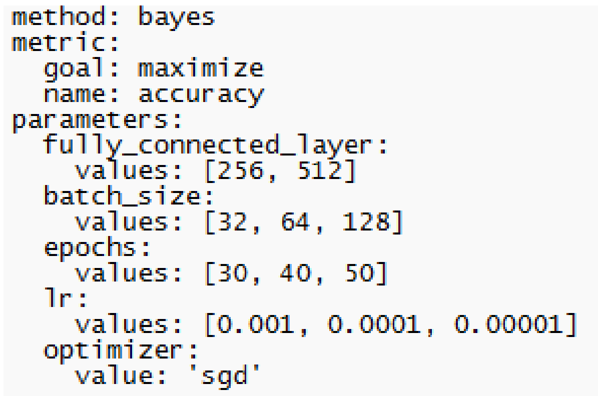
2.4.1 Classification

For the classification task the most interesting hyperparameters are number of training epochs, learning rate, batch size and the number of ReLU in the dense layer. To be more consistent with the results, all the classification models are optimized starting from the same hyperparameters ranges of value. Those ranges are summarized in Table 2.

Table 2 Hyperparameters along with their values used during the optimization

Parameters	**Values**
Training epochs	30, 40, 50
Learning rate	0.001, 0.0001, 0.00001
Batch size	32, 64, 128
ReLU in the dense layer	256, 512

The black box framework is followed further in this proceeding. As mentioned above, the input of the process is a configuration file containing the information resumed in Table 2 and reported in Figure 39. To keep the procedure as standard as possible, the model to optimize is an input that the program gets from the configuration file used for the model creation. As output, the results of the runs for each hyper-parameter combination in the form of parallel coordinate graph is obtained. An example is reported in Figure 39.



```
method: bayes
metric:
  goal: maximize
  name: accuracy
parameters:
  fully_connected_layer:
    values: [256, 512]
  batch_size:
    values: [32, 64, 128]
  epochs:
    values: [30, 40, 50]
  lr:
    values: [0.001, 0.0001, 0.00001]
  optimizer:
    value: 'sgd'
```

Figure 39 Configuration file used for the HPO of classification models

The actual black box is the code responsible for the optimization. This works as follows. First, the function called “sweep identity” must be defined. This function is responsible for loading the configuration file reported in Figure 39. In this way the code can read the hyper-parameters along with their values. In the second step, the main function must be properly coded. Inside the latter two built-in methods are called. These methods are:

1. Configurator. The configurator is the function responsible for pairing the hyperparameters of the model with their values written in the configuration file.
2. Logger. The logger, instead, is the one responsible for uploading the metrics to be monitored in the online dashboard at the end of each training session.

Between these two built-in methods, the piece of code in charge of uploading the dataset, building the model, training it and computing the metrics needed must be defined. This part is similar to the one explained in chapter 2. Therefore, there is no need to illustrate the concept again.

To start the actual procedure, the built-in function named “agent” must be called by the user. Inside this agent function it is needed to specify the sweep identity and the main function previously explained. In addition, the number of times this procedure is going to be repeated is required.

The sequence is automatically repeated by the software. Each time a new run starts the value of the different parameters is changed according to the Bayesian optimization, trying to maximize the accuracy. At the end of the procedure, as said, is it possible to appreciate the results in form of parallel coordinates graph like the one reported in Figure 40. This type of chart is very useful in order to visually understand which are the best hyper-parameters’ value to maximize the accuracy. From these graphs we obtain the optimized hyperparameters’ values.

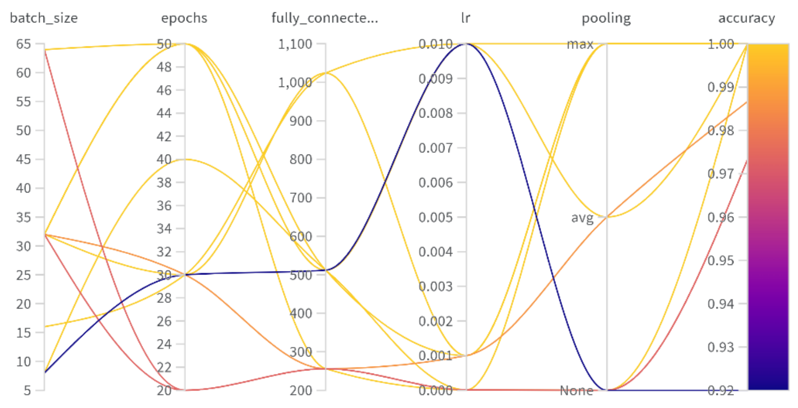


Figure 40 Example of parallel line visualization of the results of the hyperparameters optimization

2.4.2 Anomaly Detection

Switching to the anomaly detection models, WandB® is used again for the HPO procedure. The black-box framework is also used for the anomaly detection case. This architecture is very useful to standardize the different procedures through the usage of a configuration file. In Figure 41 an example of one of these YAML file is reported.

```
method: bayes
metric:
  name: image_Accuracy
  goal: maximize
parameters:
  dataset:
    category: raytec
    image_size:
      values: [64, 128, 224]
  model:
    backbone:
      values: [resnet18, wide_resnet50_2]
    lr:
      min: 0.001
      max: 0.1
    coupling_blocks:
      values: [4, 6, 8]
```

Figure 41 Example of a configuration file for hyperparameters optimization, specifically for CFLOW-AD

Now it is necessary to spend some time talking about the hyperparameters to be optimized. Differently from the HPO for the classification models, each of the anomaly detection algorithms has its own hyperparameters. In the remaining part of this paragraph, the details about model and hyperparameters chosen are given. Starting from CFLOW-AD the hyperparameters are:

- Backbone.
- Learning rate.
- Number of coupling blocks.

The backbone is the network used to encode the starting images. In this case, as in the original paper [31], two types of ResNets (ResNet18 and Wide ResNet-50-2) were proposed. Both are pretrained on ImageNet and this is good because, as already said in the classification's chapter, impressive results have been obtained in literature by pre-training an image-based model on this dataset. The second hyperparameter is the well-known learning rate. For both classification and anomaly detection models the learning rate has the same role. The range in which its optimal value has to be searched is determined by literature research of similar cases. Finally, the last hyperparameter used for CFLOW-AD, is the number of "coupling layers" each of the decoders has. A coupling layer is a particular type of layer that for its physical structure implements a normalizing flow on the information running through it. As stated in the beginning, a normalizing flow is a method used to model probability distributions which offers some mathematical advantages. For these characteristics, to be found in [32], changing the number of these layers in each decoder, slightly changes the modelling performed by the decoder or in other words changes the decoding.

For what concerns PatchCore the hyperparameters used are [10].

- Backbone.
- Number of neighbours.

Starting from the backbone, the option available for PatchCore are the same two types of ResNets previously proposed for CFLOW-AD. The number of neighbours, on the other hand, is a characteristic hyperparameter of this model. Since PatchCore tiles the starting images, it operates on the so obtained patches and the information contained in a patch is often not operating on a large enough receptive field size. This means that it sometimes does not account for meaningful anomalous context, robust to local spatial variations. This problem, that rises from the way the model operates, is tackled by the authors by aggregating the information coming from a number of neighbouring patches, after the features in each of them has been extracted. The number of elements to aggregate in this operation is the hyperparameter to be optimized.

Finally, the hyperparameters used for DFKDE are:

- Backbone.
- Confidence threshold.
- Maximum number of training points.

Again, the backbone has the same role already explained for CFLOW-AD and PatchCore. The confidence threshold is used for the density estimation step of DFKDE and so changing it changes how the probability function is modelled. Finally, the maximum number of training points is a limit to how many points are used for kernel density estimation [11].

In Table 3, Table 4 and

Table 5 the hyperparameters used for HPO on each anomaly detection model are resumed.

Table 3 Hyperparameters used to optimize CFLOW-AD

Parameters	**Values**
Backbone	ResNet18, Wide-ResNet-50-2
Number of coupling blocks	4,6,8
Learning rate	[0.001, 0.1]

Table 4 Hyperparameters used to optimize PatchCore

Parameters	**Values**
Backbone	ResNet18, Wide-ResNet-50-2
Number of neighbors	7,8,9,10

Table 5 Hyperparameters used to optimize DFKDE

Parameters	**Values**
Backbone	ResNet18, Wide-ResNet-50-2
Confidence threshold	0.3, 0.6
Maximum number of training points	15000, 40000, 60000

For each model the optimized values of the chosen parameters are reported in the results section. In particular, also the image size is put among the parameters to be tested in these cases, trying to understand if a greater resolution gives better results.

2.5 Comparing models: K-fold, bootstrap, deep dominance

Once all models have been optimised for specific task and dataset of this thesis, they must be applied (i.e. tested) and compared. The goal is to identify the best model for the industrial application. For this reason, a first step is to use techniques able to generalize the result. This means that it is necessary to understand how the system will behave on independent data such as those with which it will then work in the real application.

One of the most used techniques to analyse the system's performances in this sense is K-Fold cross validation [33]. This technique, that is graphically resumed in Figure 42, consists in:

- Dividing the entire dataset into K parts
- For each part performing the following tasks once:
    - Keep that part as test set
    - Keep the other K-1 parts as training set
    - Train the model on the so obtained training set
    - Test the model on the corresponding test set

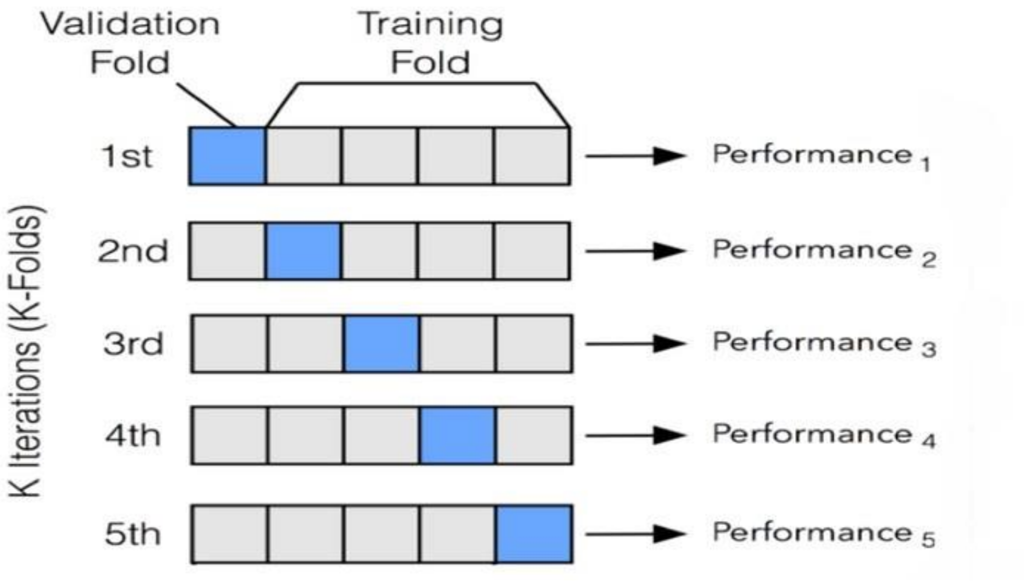


Figure 42 How K-Fold validation works

Generally, these results are then summarized by taking mean and standard deviation of the K performances. On the other hand, recent literature also concluded that since neural networks have non-convex loss surfaces this procedure is not rigorous. As stated by [34]: *"These models have a large number of hyper-parameters and being non-convex, their convergence point depends on the random values chosen at initialization and during training"*.

Therefore, to compare different models a further analysis on the K performances values obtained is needed. In particular, the GitHub repository called Deep-Significance [35] developed by the university of Copenhagen is used to this end. This repository contains a series of tools to perform statistical significance tests on neural networks. In this thesis work, two of them were chosen to compare the models:

- *Almost Stochastic Order*
- *Bootstrap power analysis*

Almost stochastic order (ASO), as stated in the [36] is a statistical tool that is:

- SIGNIFICANT: this assures that future runs of the superior model are likely to get higher scores than future runs of the inferior model
- POWERFUL: this means it is able to make decisions in most possible decision tasks
- GIVES A CONFIDENCE SCORE

The operating principle is reported in the following. A Probability Density Function (PDF) is constructed from samples. In this specific case these samples are the metric scores extracted with the K-Fold method. After that, from the PDF, the Cumulative Density Function (CFD) is taken as reported in the example in Figure 43.

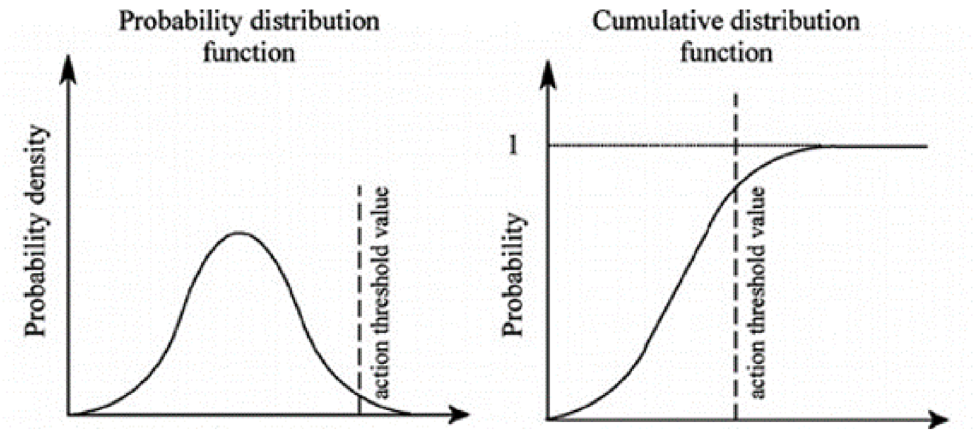


Figure 43 Left:example of PDF. Reft: example of CDF

From the definition of CFD, the stochastic dominance of one distribution over the other can be defined if:

$$F(a) \leq G(a)\ for\ all\ a \quad [5]$$

As an example, is it possible to refer to Figure 44. In the figure, the green curve can be associated to G(a) while the red curve represents F(a). Just by looking at the picture, it is clear that the green curve is dominant with respect to the red one.

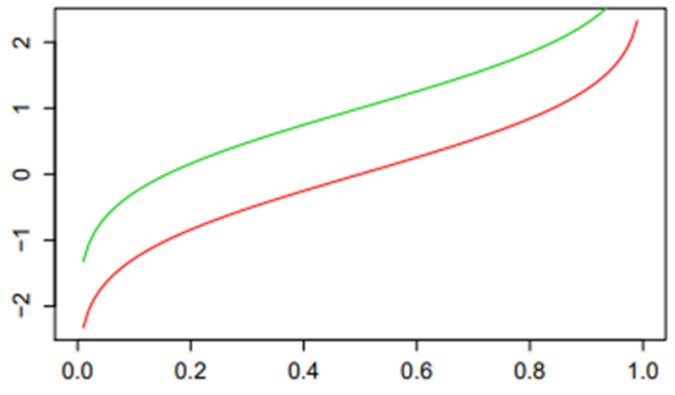


Figure 44 Example of stochastic dominance of a CFD over another

Unfortunately, as noted by [36]: *"It often happens in Statistics, that the concept of stochastic dominance is excessively rigid". "Stochastic order is a 0-1 relation. It is either true or false".*

Indeed, in many applications, situations like the ones shown in Figure 45 could rise. For example, let F(a) be the red curve and let G(a) be the green curve. In that case it is not possible to say which one is dominant, because the two curves intersect.

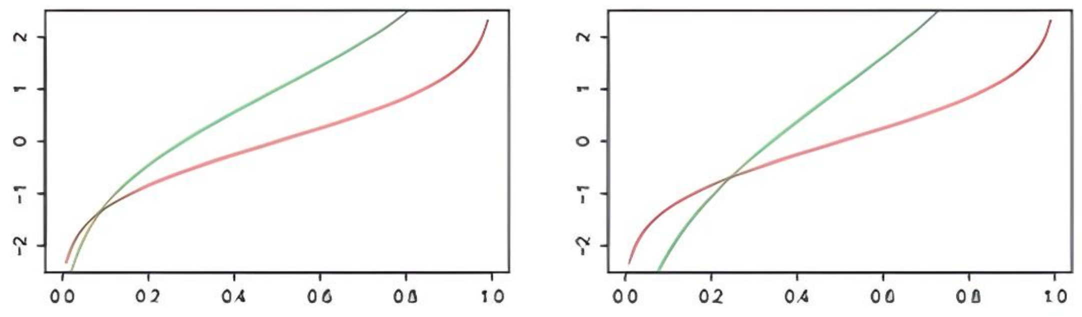


Figure 45 Examples of situations in which a predominant distribution is present but stochastic order is not satisfied

ASO, as proposed in the original paper by Dror et al. [34] Dror et al., is able to identify the better model in such situations with statistical meaning. Starting from two random variables X and Y with CDF's F and G, an index ε_{W2} is calculated (the exact formula can be in found [34]). This index satisfies the formula

$$0 \leq \varepsilon_{W2}(F,G) \leq 1 \qquad [\,6\,]$$

Where:

- 0 corresponds to perfect stochastic dominance of X over Y.
- 1 corresponds to perfect stochastic dominance of Y over X.

It also holds that: $\varepsilon_{W2}(F,G) = 1 - \varepsilon_{W2}(G,F)$.

In other words, this index could be visualized as the quantification of the extent to which stochastic order is being violated like in Figure 46.

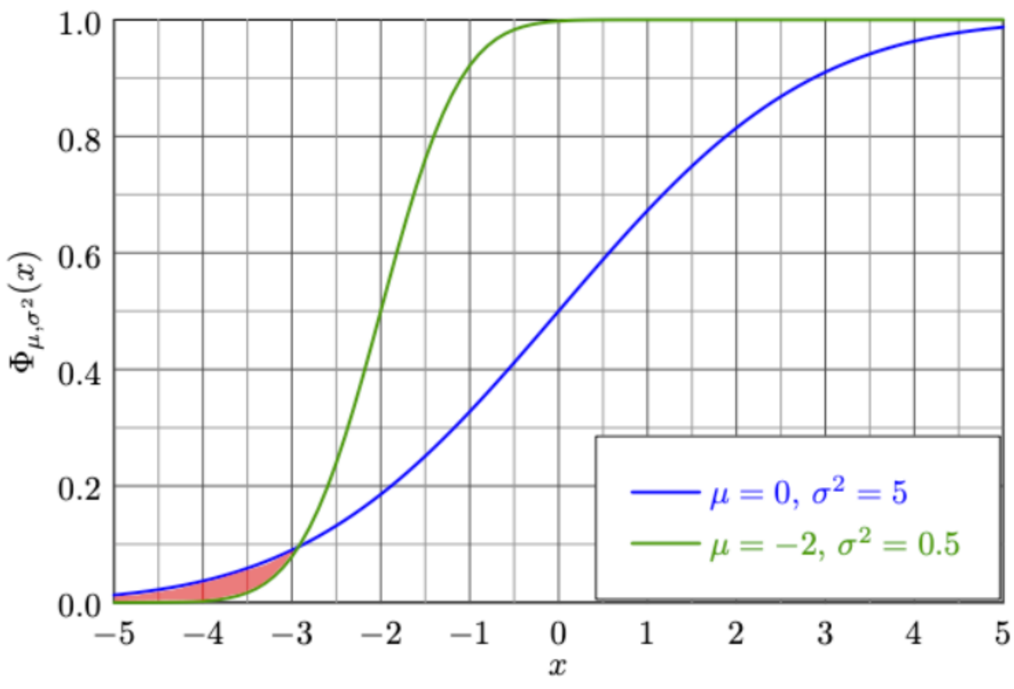


Figure 46 Example of how the index introduced in ASO could be visualized

Based on this quantity a test is formulated with the following hypotheses:

$$H_0 : \varepsilon_{W2}(F, G) \geq \tau \quad [7]$$

$$H_1 : \varepsilon_{W2}(F, G) < \tau \quad [8]$$

Where τ is a pre-defined quantity (generally it is $\tau = 0.5$). In this test refusing the H₀ hypothesis means accepting that $\varepsilon_{W2} < \tau$ and so that F is almost stochastic dominant over G. Starting from this the authors further find a minimal index for which we can reject the null hypothesis with a given confidence [35]. This is the value we are interested and that the given tool in the repository calculates.

In other words, given the observed scores and a confidence value, the ASO algorithm returns a value which expresses an upper bound to the amount of violation of stochastic order. For this thesis work, a confidence level of 95% is chosen. Such a value is suggested in [37] and it is commonly used in other tests of this type. In the following paragraph it is reported how this tool is used in the specific case analysed.

When comparing two vectors A and B of samples of a specific metric (i.e., accuracy) and obtaining $ASO(A, B) = 0.2$, the following information is given: B is almost stochastic dominant over A for less than 0.2. This is useful to conclude that model A is dominant over B for what regards accuracy. On the other hand, a result like $ASO(A,B) = 0.86$ is not returning any useful information since it tells that B is almost stochastic dominant over A for less than 0.86, which could be any value between 0 and 0.86. In simple terms, to state that a specific model is better than another with regards to a certain metric, it is useful to look for values of ASO below 0.5.

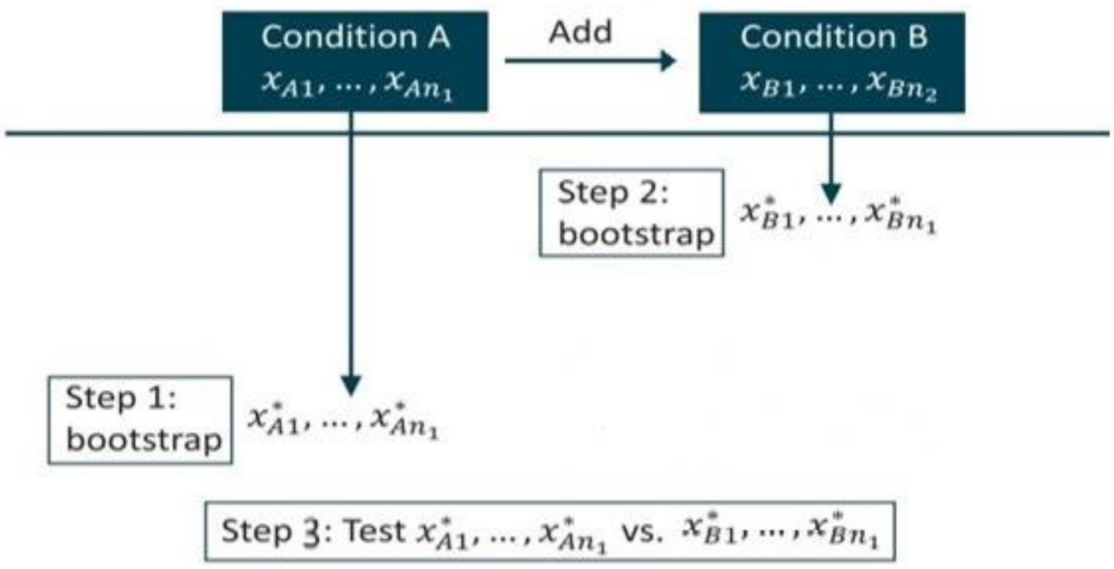
It is interesting to notice that no assumption about the distribution of the scores is made in this procedure. On the other hand, it is evident that the number of scores allowing for reliable significance testing is arbitrary in this method. Due to this, also the bootstrap power analysis is used.

Bootstrap power analysis [38] is used to understand if the number of samples used is sufficient or not to model a PDF and run significance tests on it. In the specific case analysed, this helps to understand if the number of folds with which K-Fold is performed is high enough.

This power analysis works as follows. It takes the sample vector and normalizes it. Then, it gives to all the elements present in the vector a uniform lift to create an second artificial sample. Multiple versions of both samples are then obtained through bootstrapping. This means similar samples are produced with random sampling and replacement from the two initial samples.

The ones coming from the original samples and the ones obtained from the incremented one are then compared with a significance test to see if they belong to the same distribution or not. In practice, the percentage of comparisons that yield significant results is returned. The authors state that a uniform lift of 1.25 should result in statistically significant differences. If this is not the case this means that the starting distribution has such a big variance that the system is not able to identify the difference between the two distributions. In other words, the variance is too high, and more samples should be collected.

The authors conclude that values of the significance test above 0.8 can be considered acceptable. A schematic representation of this proceedings is reported in Figure 47.



Condition A
$x_{A1}, \dots, x_{An_1}$
Add
Condition B
$x_{B1}, \dots, x_{Bn_2}$
Step 2: bootstrap
$x^*_{B1}, \dots, x^*_{Bn_1}$
Step 1: bootstrap
$x^*_{A1}, \dots, x^*_{An_1}$
Step 3: Test $x^*_{A1}, \dots, x^*_{An_1}$ vs. $x^*_{B1}, \dots, x^*_{Bn_1}$

Figure 47 Schematic representation of how power analysis works. Condition B is just condition A incremented by a constant. The here presented procedure is repeated many times and the percentage of significant results is returned.

It must now be noticed that when choosing the K factor for K-Fold, two different needs must be balanced out:

1. High K for good power analysis values.
2. Low K to not have unreliable test sets with low number of samples.

K equal to five, which is a commonly adopted value, was found to be a good choice as will be justified in the results section.

3 Results

In this chapter, the main results of each model will be presented sequentially, starting from the classification algorithms and ending with anomaly detection ones. These results are the direct consequences of the methods described in chapter 2.

In the first part of this section each model's experimental outcome is described independently, mainly relying on tables and figures. In this way, results can be listed in a standardized way, making the following comparisons more readable.

In the second part of this chapter, the results of the comparison among all the models are reported. This comparison was done with the tools introduced in the previous chapter.

3.1 Classification

In the following sub sections the main results for each classification model are reported, using a standardized structure. This chapter is organized in the following way for each model: first, the parallel coordinates plot obtained as result of the hyper-parameters optimization is reported and commented. From this chart, the optimized value of each hyper-parameter is taken and summarized in a table. Then, the main code is executed with the optimized hyper-parameters, and the training procedure can begin. From this, two curves are extrapolated: training accuracy vs validation accuracy and training loss vs validation loss. These curves are of fundamental importance. Indeed, by analysing them, it is possible to understand whether the algorithm is generalising well (i.e., no divergency is presented) as well as if the call-back procedure is working, avoiding any overfitting. After training has been completed, the test phase can be carried out. In this phase, a new set of images is given to the model and on this, the model's performance can be evaluated. In particular, accuracy, AUROC and inference time are considered.

3.1.1 Resnet50

As explained, the first step to get the final model is the identification of the best value for each hyper-parameter. This is achieved by using WandB®

as explained above. The result of the HPO procedure is shown in Figure 48. The optimal value for each parameter is extracted from this graph. For the sake of clarity these values are summarized in Table 6.

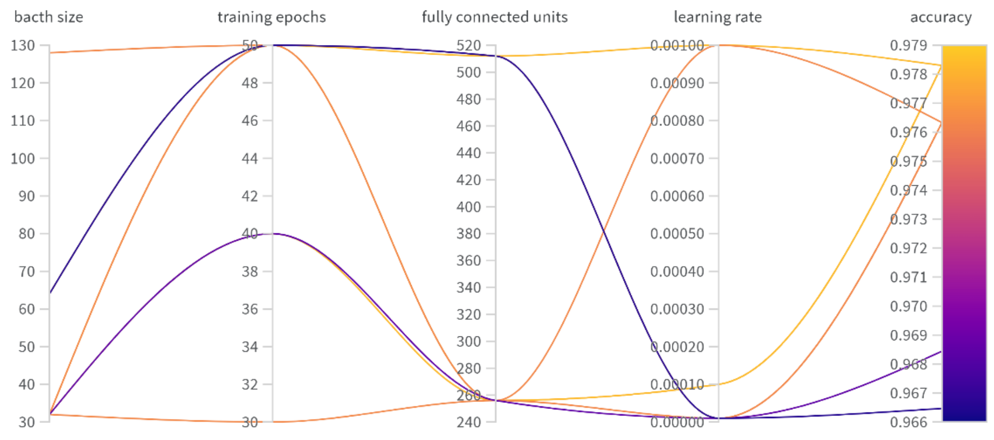


Figure 48 HPO results for ResNet50

Table 6 Optimal values of the hyper-parameters for ResNet50

Parameters	**Values**
Batch size	32
Training epochs	50
Learning rate	0.001
Fully connected units	512

Now, it is possible to compile the model with the so found values (by editing only the configuration files) and to proceed with the training. The results of this procedure in terms of training versus validation accuracy and training versus validation loss are reported in Figure 49.

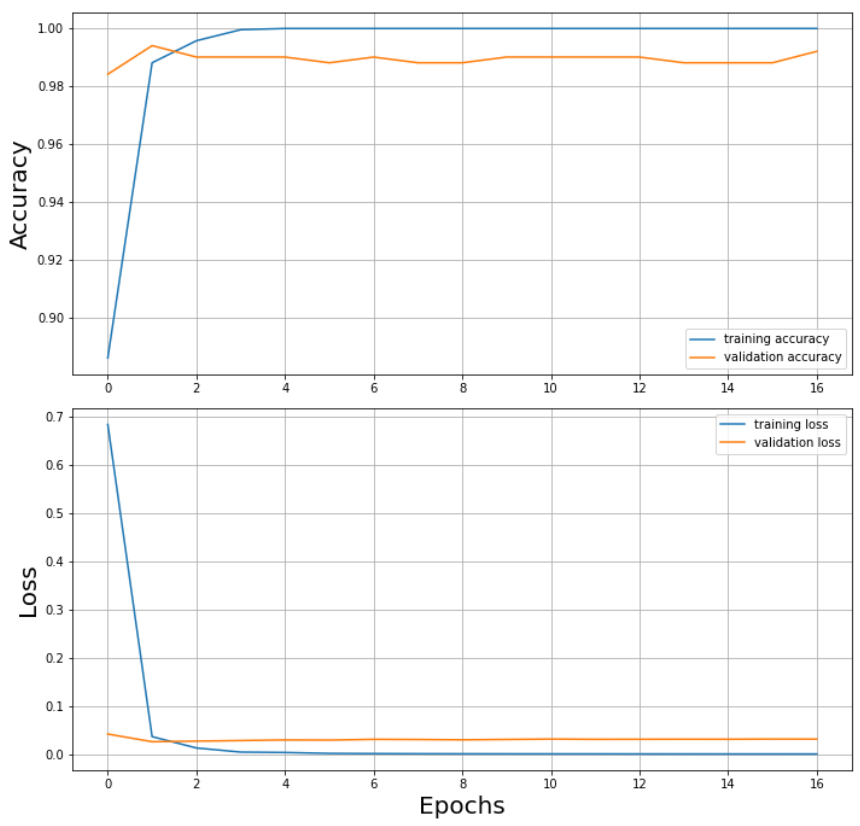


Figure 49 Training and validation curves for accuracy and loss obtained with ResNet50

Just by looking at the graph it is possible to say that the training phase was successfully completed. Indeed, the call-back stops the training at 16 epochs instead of the 40 proposed by the HPO, avoiding the overfitting problem. Furthermore, as can be seen, there is no divergency between training and validation accuracy curves and the same can be also noticed for training and validation loss curves. The last passage consists in testing the model. The value of each metric obtained is summarised in Table 7. Confusion matrix and AUROC are reported in Figure 50 and Figure 51, respectively.

Table 7 Accuracy, AUROC and inference time values for ResNet50 on the test dataset

	Accuracy [%]	Inference time [ms]	AUROC
Values	98.62	3	99.03

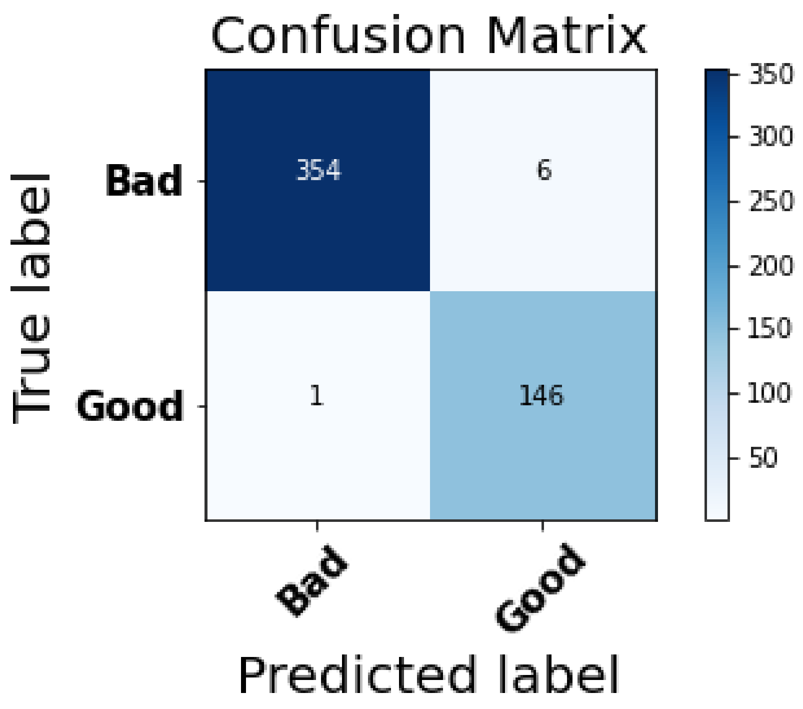


Figure 50 Confusion matrix for ResNet50

According to Raytech Vision S.P.A., among the wrongly classified tomatoes, the worst are those that are classified as good but are actually bad. Raytech Vision S.P.A. has expressly stated that consumers prefer to have a higher product waste in order to keep the quality standard high. Referring to what has just been said and taking Figure 50 as an example, the ideal situation would be the opposite of the confusion matrix. In any case, the number of wrongly cassified tomatoes is very low (7 on a total of 507 images) and and the end result in terms of accuracy is very satisfactory. This consideration is confirmed by the ROC reported in Figure 51.

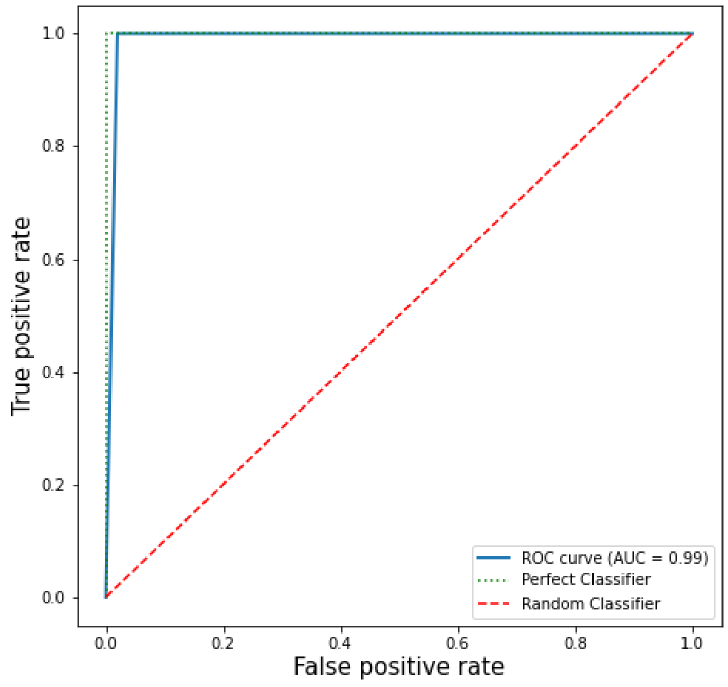


Figure 51 ROC curve for ResNet50

So far, the results of the classical training, validation and testing approach have been reported and briefly discussed. In Table 8, instead, the results of the K-Fold procedure done on the entire dataset are reported. It is useful to remember that K=5 is chosen to apply the K-Fold.

Table 8 Results of cross validation for ResNet50

	Fold n°1	Fold n°2	Fold n°3	Fold n°4	Fold n°5	Mean	Std. deviation
Accuracy [%]	99.55	98.81	99.41	98.37	99.55	99.14	$4.7*10^{-1}$
AUROC [%]	100	100	100	100	99.99	99.99	$1.69*10^{-3}$
Inference time per image [ms]	3.5	3.7	3.6	3.6	3.8	3.6	$5.7*10^{-1}$

On these values, the bootstrap power analysis is performed. That gives the results reported in Table 9 which are satisfying. As can be seen, K=5 was sufficient to performing reliable significance testing.

Table 9 Power analysis for the metric values from ResNet50

	Power
Accuracy	1
AUROC	1
Inference time	1

Lastly, applying the Grad-CAM on some of the images (see Figure 52) the correct functioning of the algorithm is confirmed. In fact, it can be seen that the algorithm focuses on the most interesting part of each picture (i.e., the defects present on the tomato's skin) to perform classification.

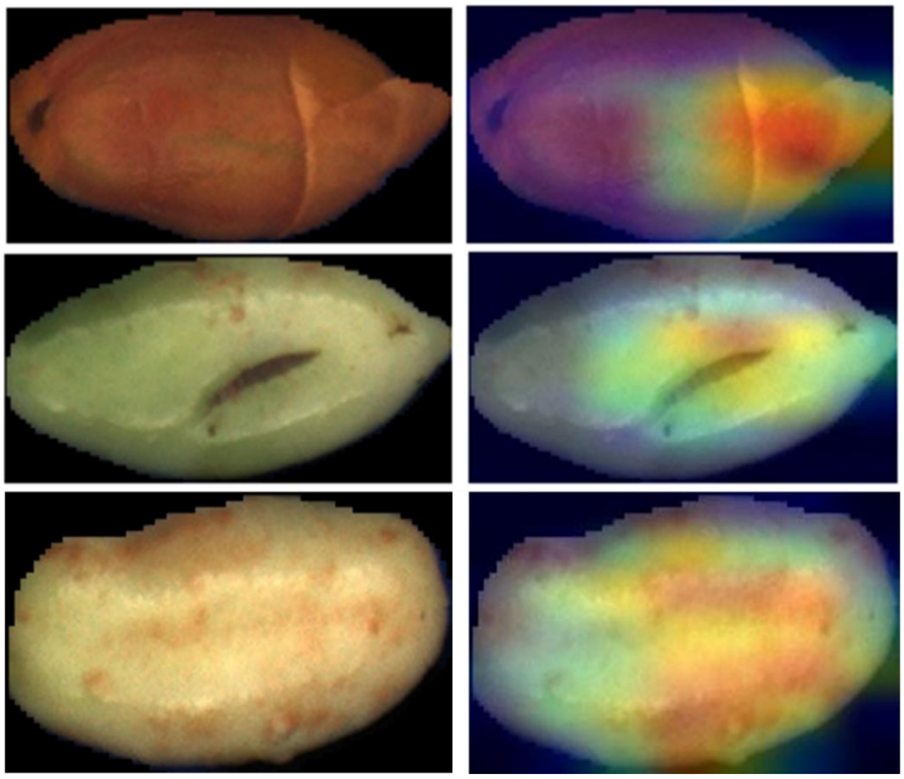


Figure 52 Outcome of the Grad-CAM applied to ResNet50. We see that the model is working correctly since it focuses on the defects. In the bottom, as the defect is the green colour, it focuses on the entire tomato.

3.1.2 Mobilenet

As before, the first step is the HPO procedure. The best way to analyse the result is to look at the parallel coordinate graph, shown in

Figure 53.

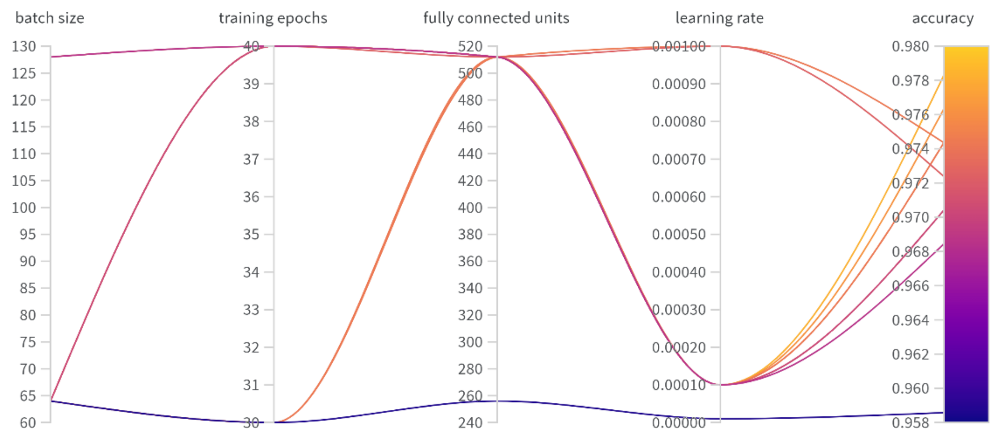


Figure 53 Parallel coordinates plot for the HPO of MobileNet

From the chart, the value of each hyper-parameter is extrapolated to ensure the best possible accuracy. For the sake of clarity, these values are reported in Table 10.

Table 10 Optimal values of the hyper-parameters for MobileNet

Parameters	Values
Batch size	64
Training epochs	40
Learning rate	0.0001
Fully connected units	512

As explained, these values must be entered manually in the software through the modification of the configuration file. After that, the training procedure can be started. The result of the training and validation, in terms of accuracy and loss curves, is reported in Figure 54.

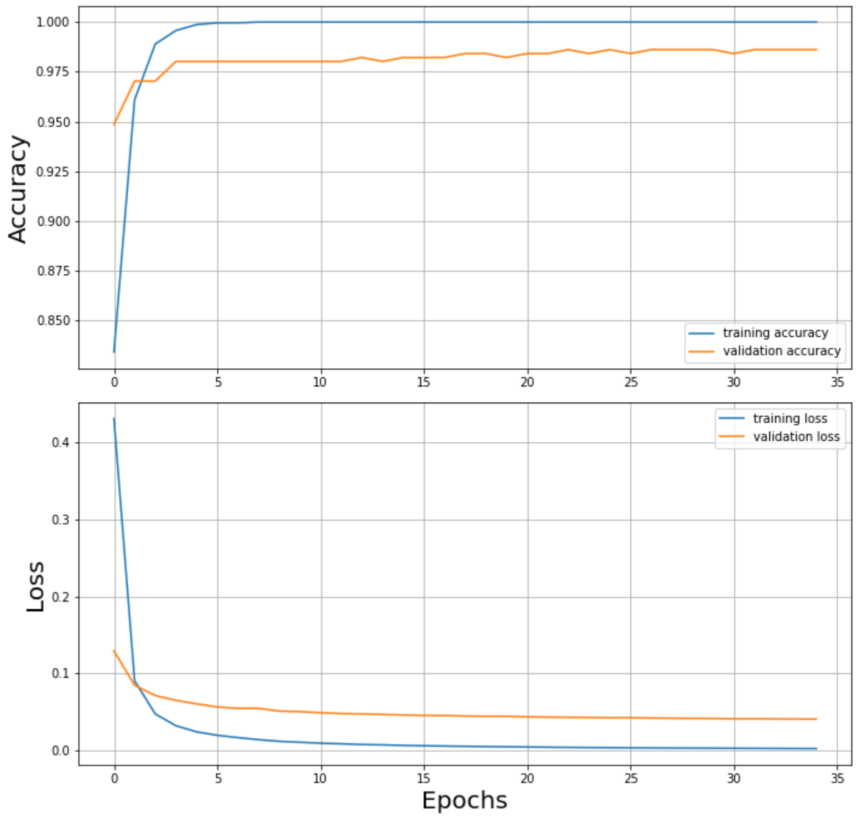


Figure 54 Training and validation curves for accuracy and loss for MobileNet

By analysing the behaviour of these curves two main points can be highlighted:

1. As for ResNet50 case, the call-back stops the training before the 40 epochs suggested by the HPO. Anyway, MobileNet requires more training epochs than ResNet50 that means more training time is needed.
2. Fortunately, the algorithm generalises well, because, comparing training and validation curves for both accuracy and loss, no divergency is presented.

The test phase followed the training one. The classification accuracy on the test dataset of MobileNet along with inference time and AUROC are reported in Table 11. This table is followed by the plots of Confusion matrix and ROC curve. It should be noted that, the inference time achieved by MobileNet is very low. According to Raytec Vision S.p.A., this inference time is good enough to be integrated in already existing sorting hardware.

Table 11 Accuracy, AUROC and inference time values for MobileNet on the test dataset

	Accuracy [%]	Inference time per image [ms]	AUROC [%]
Values	98.82	2	98.16

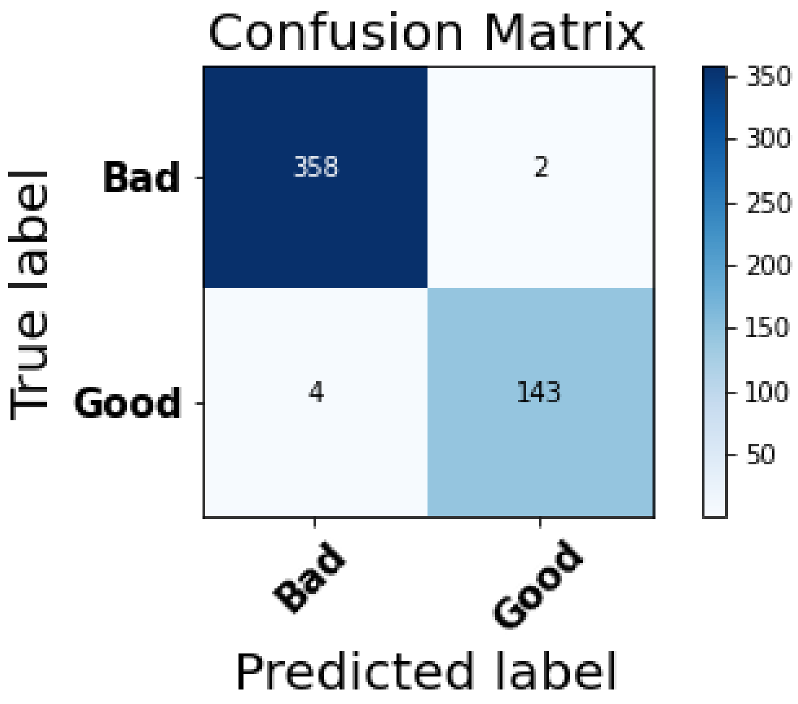


Figure 55 Confusion matrix for Mobile Net

Unlike the previous case (i.e. Figure 51), Figure 55 reflects the ideal case that Raytech Vision S.P.A. would like to achieve. In this case, in fact, the model discards more good tomatoes bad ones which it 'does not see'. The excellent functioning of this model can be confirmed by looking at Figure 56. Indeed, an AUROC above to 0.98 is considered excellent, as explained previously.

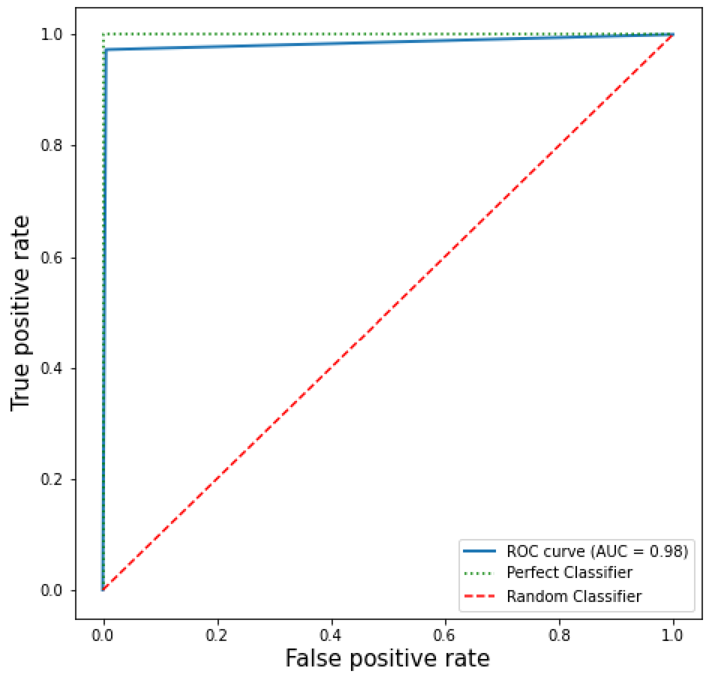


Figure 56 ROC curve for MobileNet

Once training, validation and testing are concluded, K-fold cross validation on the entire dataset can be done. In Table 12 are shown the results of the cross validation using K=5.

Table 12 K-fold results for MobileNet

	Fold n°1	Fold n°2	Fold n°3	Fold n°4	Fold n°5	Mean	Std. deviation
Accuracy [%]	98.22	98.51	98.66	98.52	99.11	98.61	$2.9 * 10^{-1}$
AUROC [%]	99.99	99.98	99.99	99.99	99.73	99.94	$1.15 * 10^{-1}$
Inference time per image [ms]	1.7	1.3	1.3	1.3	1.4	1.4	$1.73 * 10^{-1}$

To briefly comment the reported charts, it should be said that MobileNet is far faster than ResNet50, still preserving a pretty high accuracy and AUROC.

In Table 13 are reported the results of the power analysis done on the values obtained using the K-Fold cross validation.

Table 13 Power analysis for MobileNet

	Power
Accuracy	1
AUROC	1
Inference Time	0.84

These results show that K=5 is a satisfactory value for the following comparison.

Finally, the Grad-CAM was used with the aim of understanding where the model is looking to classify the images. The result is shown in Figure 57. The tomato is green but without punctual defects. In fact, as expected, the model shows an even heatmap on the entire tomato skin and does not colour the background.

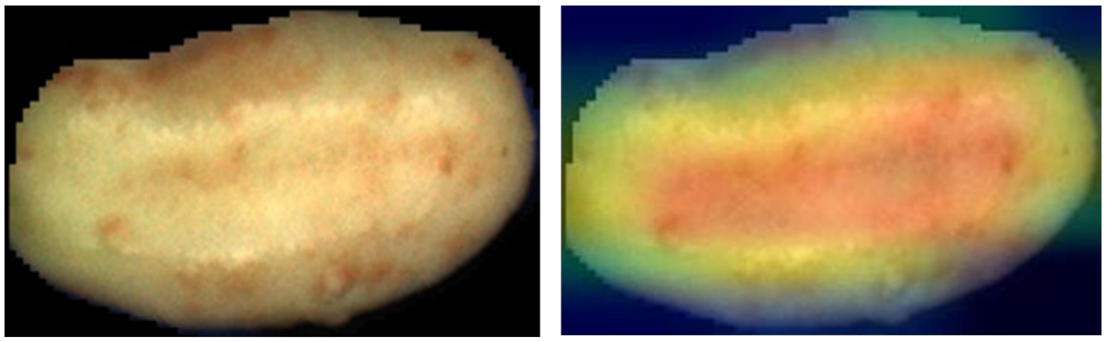


Figure 57 Example of Grad-CAM for MobileNet. The model evenly focuses on the entire tomato since it is green, but no punctual defects are present

3.1.3 GoogLeNet

For the last classification algorithm, the starting point is again the HPO thanks to the WandB® support. The results of the latter are reported in the parallel coordinates graph in Figure 58 and summarized in Table 14.

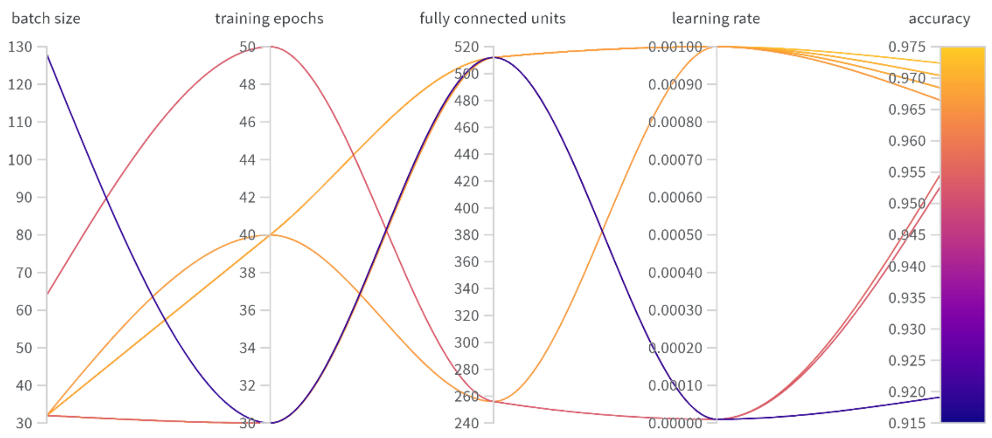


Figure 58 Parallel coordinates plot for the HPO of GoogLeNet

Table 14 Optimal values of the hyper-parameters for GoogLeNet

Parameters	**Values**
Batch size	32
Training epochs	40
Learning rate	0.001
Fully connected units	512

Following the same structure used for the previous models, the results of the training are reported in Figure 59. Clearly, before starting the training, the optimized value for each hyper-parameter must be manually entered in the code through the YAML file.

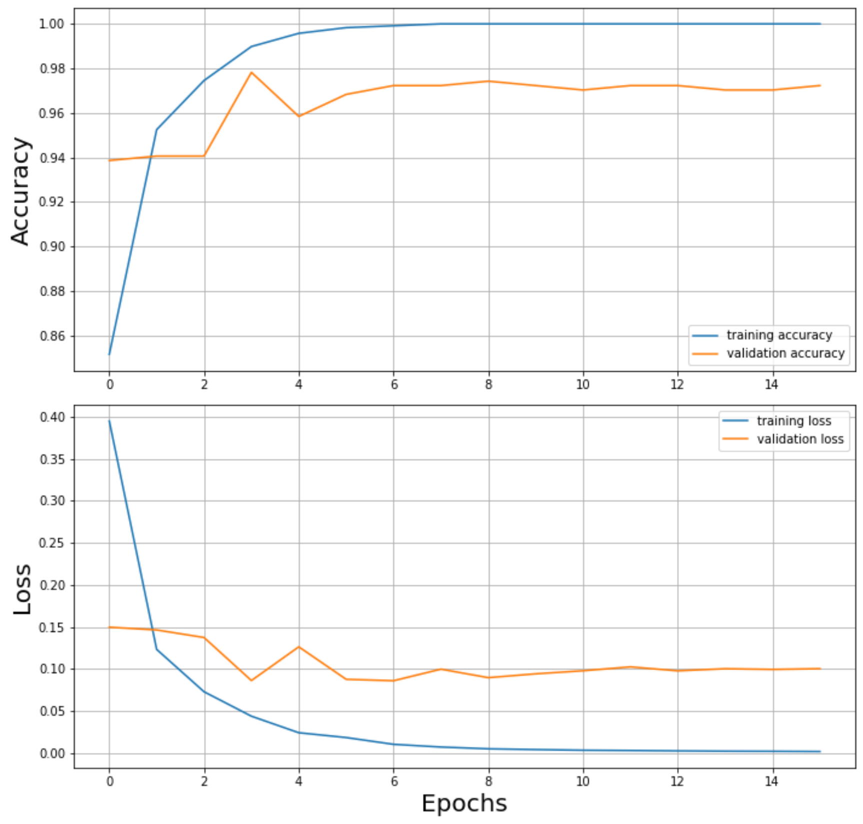


Figure 59 Training and validation curves for accuracy and loss for GoogleNet

From the training curves, two main points can highlighted:

1. The correct functioning of the call-back should be noticed. Indeed, the training is stopped much sooner than what is expected from the HPO.
2. Moreover, there is no divergence, a sign that the algorithm is generalising well.

As last passage, the testing phase can be accomplished. The results of the latter are reported in Table 15. The plots of confusion matrix and ROC curve for GoogLeNet are shown in Figure 60 and Figure 61, respectively.

Table 15 Accuracy, AUROC and inference time values for GoogLeNet on the test dataset

	Accuracy [%]	Inference Time per image [ms]	AUROC [%]
Values	97.63	4	96.00

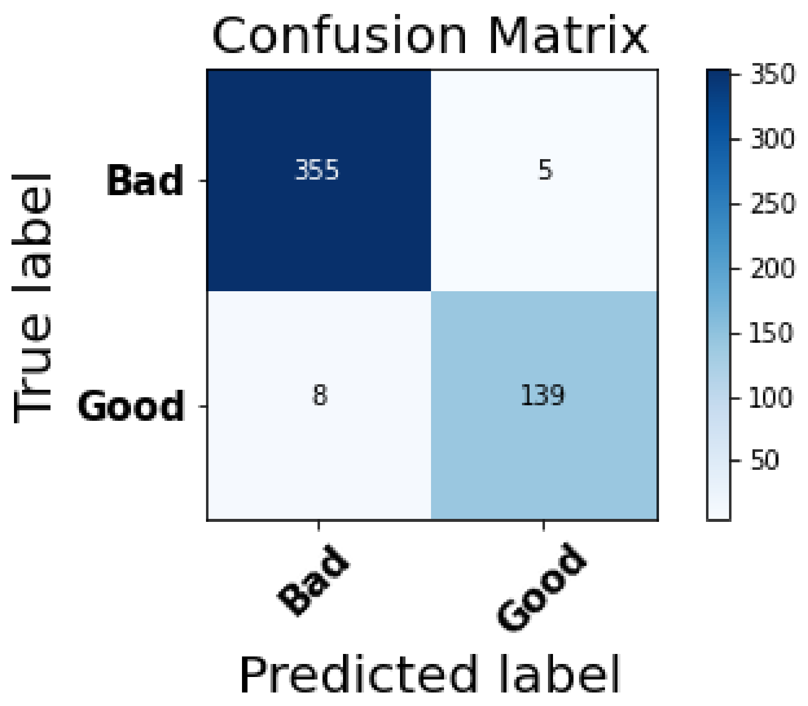


Figure 60 Confusion matrix for GoogLeNet on the test set

For GoogLenet as well as for MobileNet, the Confusion Matrix shown in Figure 60 gives an idea of the number of bad tomatoes classified as good. As already stated previously, this index is really important for the industrial partner Raytech Vision S.P.A. In this case, of the thirteen wrongly classified, less than half are actually bad. The accuracy achieved by the classifier is satisfactory. This conclusion is further confirmed by the ROC shown in Figure 61. Indeed, although the AUROC value is the lowest of those found so far, as specified in the previous chapter, an AUROC equal to 0.96 is considered excellent.

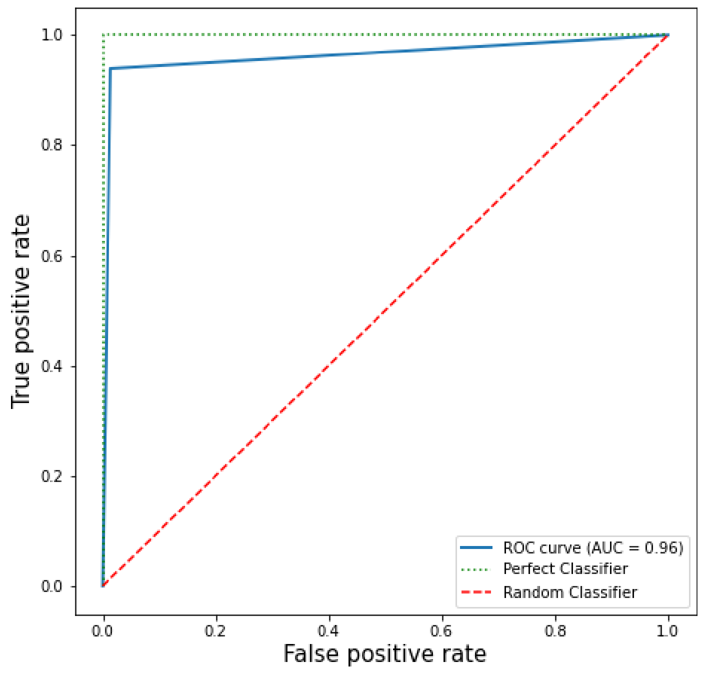


Figure 61 ROC for GoogLeNet on the test set

Finally, it is possible to perform the K-Fold cross validation on the entire dataset. The results of this procedure are summarized in Table 16.

Table 16 K-Fold results for GoogLeNet

	Fold n°1	Fold n°2	Fold n°3	Fold n°4	Fold n°5	Mean	Std. deviation
Accuracy [%]	98.48	96.59	97.92	97.33	97.92	97.64	$7.18*10^{-1}$
AUROC [%]	100	100	100	100	99.64	99.93	$1.42*10^{-3}$
Inference time per image [ms]	2.6	2.0	2.0	2.3	2.3	2.24	$2.51*10^{-1}$

On these values it is possible to perform bootstrap power analysis (Table 17). The results obtained from the latter are, once again, satisfactory. In fact, the different values of the power found confirm that K=5 is a sufficiently high value.

Table 17 Power analysis for GoogLeNet

	Power
Accuracy	1
AUROC	1
Inference time	0.90

As usual, the last operation consists in using the GRAD-CAM method to understand if the model is operating properly. From Figure 62, it is possible to appreciate where GoogLeNet is looking to classify the specific image analysed. In particular, the algorithm focuses on the defect presents on the skin and therefore we conclude that the last classification model also works properly.

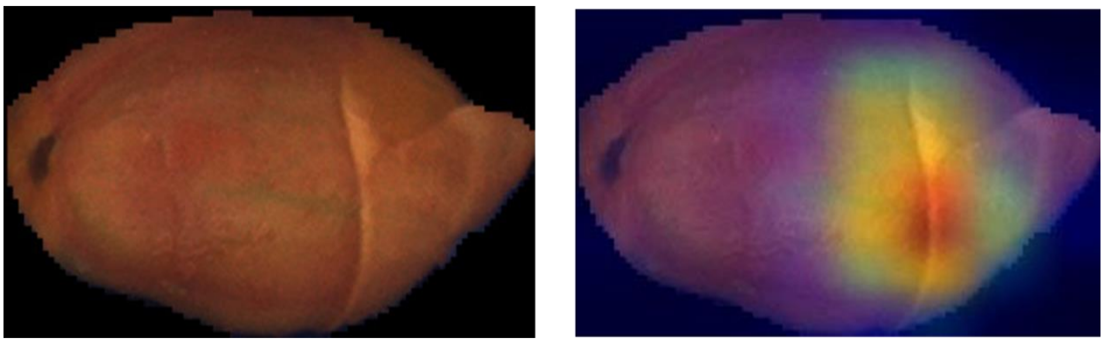


Figure 62 Grad-CAM result using GoogLeNet. The model is correctly focusing on the defect to classify the image. In this case the defect is the remaining unpeeled part of the tomato.

3.2 Anomaly Detection

This section describes the scheme followed to report the experimental results of the three anomaly detection models. First, the HPO results are shown, summarising the values of the chosen hyper-parameters in a table and presenting the increase in model performance after this step. Then the results of the K-Fold procedure are shown through a table. Lastly, the bootstrap power analysis is performed on the k values. Again, bootstrap results are shown in a table.

3.2.1 CFLOW-AD

Applying the standard CFLOW-AD algorithm as contained in the GitHub repository on the main dataset, the following performance for what concerns AUROC on the validation set are obtained. See Figure 63 for the details.

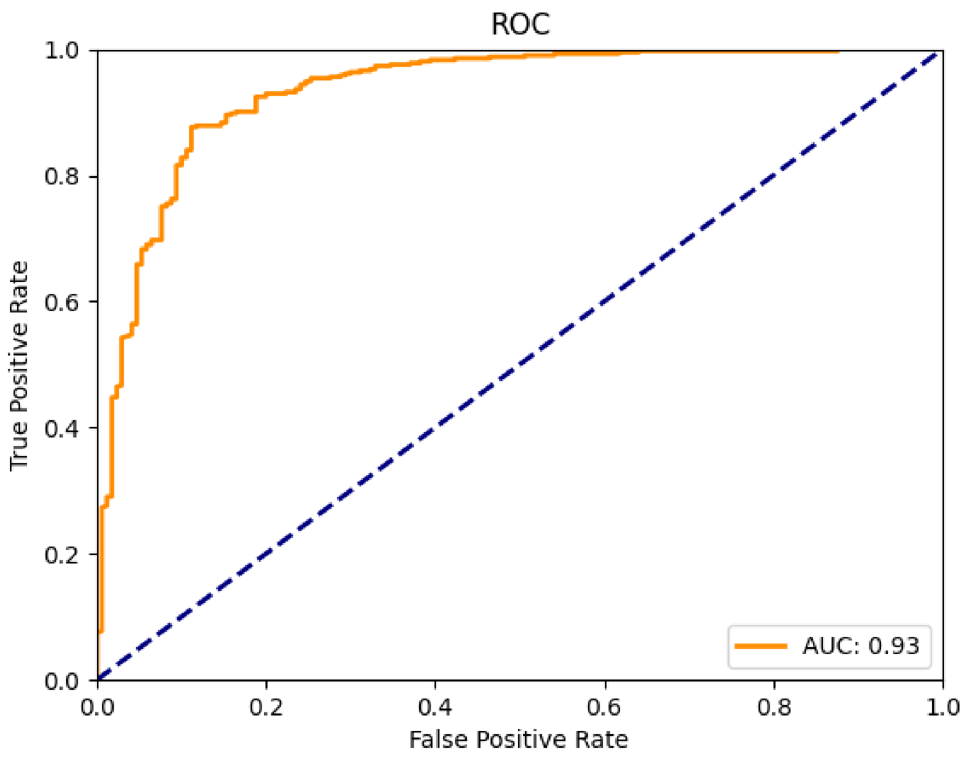


Figure 63 AUROC on validation dataset with default settings of CFLOW-AD

Using the WandB® support it is explored how the accuracy of the model on the specific dataset changes with the different settings introduced in the methods section. The results can be appreciated by looking at Figure 64.

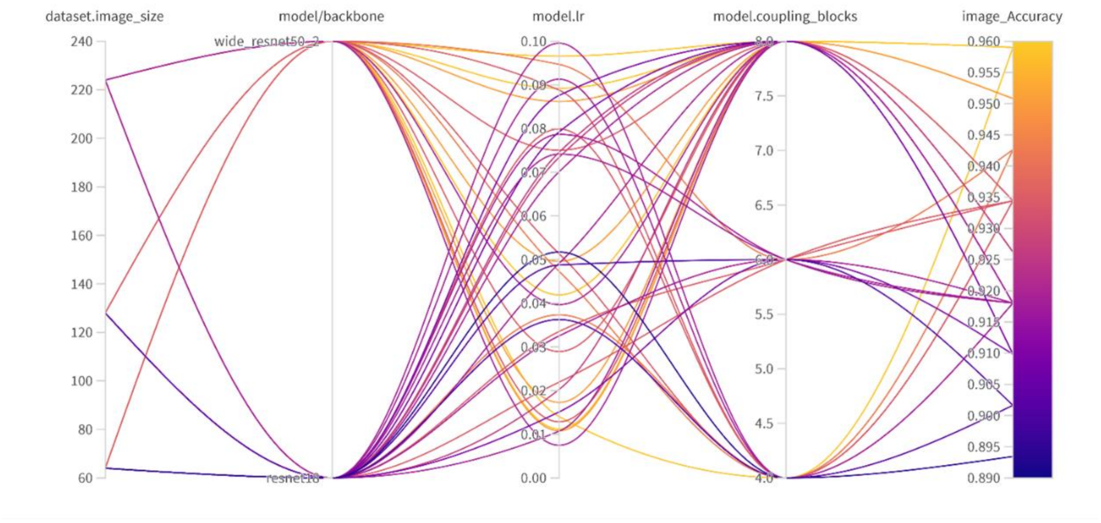


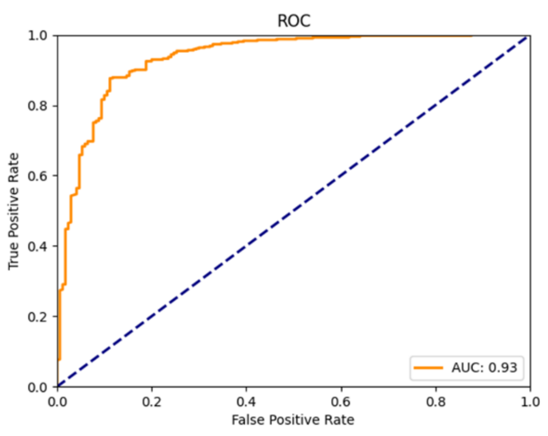
Figure 64 Hyperparameters optimization visualization for CFLOW-AD

With the graphical aid given by the dashboard the best combination of values for the hyper-parameters was identified to obtain the highest accuracy value. The values found for each hyper-parameter are reported in Table 18.

Table 18 CFLOW-AD hyperparameter chosen for the dataset after optimization

Parameters	**Values**
Image size	224
Model backbone	Wide_resnet50_2
Learning rate	0.01122
Coupling blocks	8

By training the model with the values for the hyper-parameters just found, the ROC curve obtained on the validation set shows an increment in the AUROC value, as can be seen graphically seen in Figure 65.



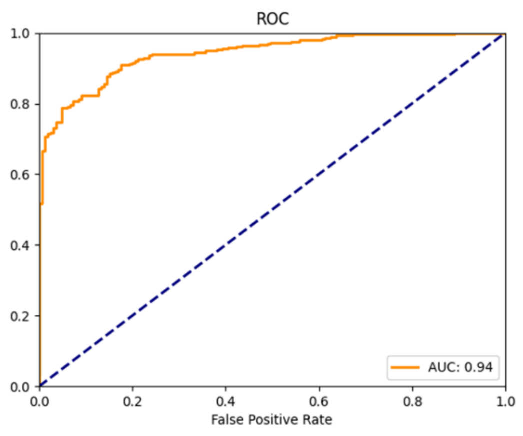


Figure 65 Increase of performance after hyperparameters optimization for the CFLOW-AD model

As said at the beginning, CFLOW-AD defines the anomaly score at image level. In other words, in the case of CFLOW-AD it is possible to visualize the anomaly score in a graphical way. This is done following a logic similar to the one used with Grad-CAM for the classification case. In fact, by visualizing the anomaly score heatmap it is possible to understand where the model "thinks" the anomaly is located. In a nutshell, the algorithm is expected to focus on the specific anomalies present. This trend is confirmed, and some examples are reported in Figure 66. The meaning of this type of visualization is similar to the Grad-CAM. Indeed, it helps in understanding if the model is operating correctly.

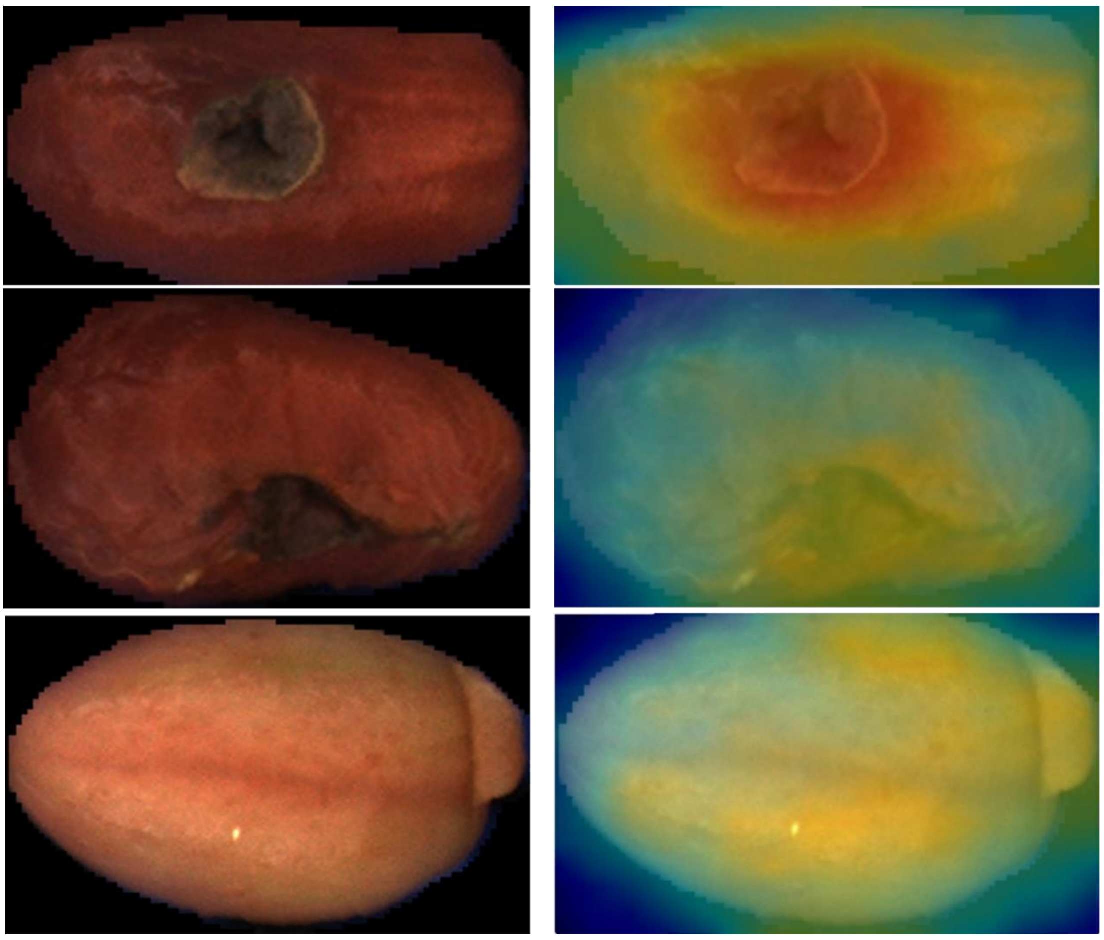


Figure 66 Examples of anomaly maps produced by CFLOW-AD: it can be noticed that the model identifies the anomalies, this indicates that it is working correctly

As can be seen in the two bottom images of Figure 66, the anomaly is given by the overall skin of the tomato that is unripe and in fact the model gives a weak anomaly (i.e., light yellow colour) on the entire tomato. On the other hand, for what concerns the other four images, it should be noticed that the model is detecting the anomaly weakly on the entire tomato (and this is good because they are not correctly peeled), but it focuses mainly on the localized defects. This is the expected behaviour. With the so optimized model, K-Fold cross validation with 5-folds is performed. Table 19 contains a resume of the so obtained performances.

Table 19 K-Fold results for CFLOW-AD

	Fold n°1	Fold n°2	Fold n°3	Fold n°4	Fold n°5	Mean	Std. deviation
Accuracy [%]	95.34	94.94	95.62	95.22	95.45	95.31	$2.56*10^{-1}$
AUROC [%]	95.70	94.22	95.93	94.97	94.81	95.13	$6.93*10^{-1}$
Inference time per image [ms]	28.5	30.3	25.6	32.2	27.7	28.86	2.3

With these values it is now possible to apply the bootstrap power analysis. The results are reported in Table 20 and show that K equal to five is a sufficiently large number of samples.

Table 20 Bootstrap power analysis results on CFLOW-AD output

	Power
Accuracy	1
AUROC	1
Inference time	0.98

3.2.2 PatchCore

Figure 67 shows the AUROC of the standard PatchCore algorithm on the validation dataset.

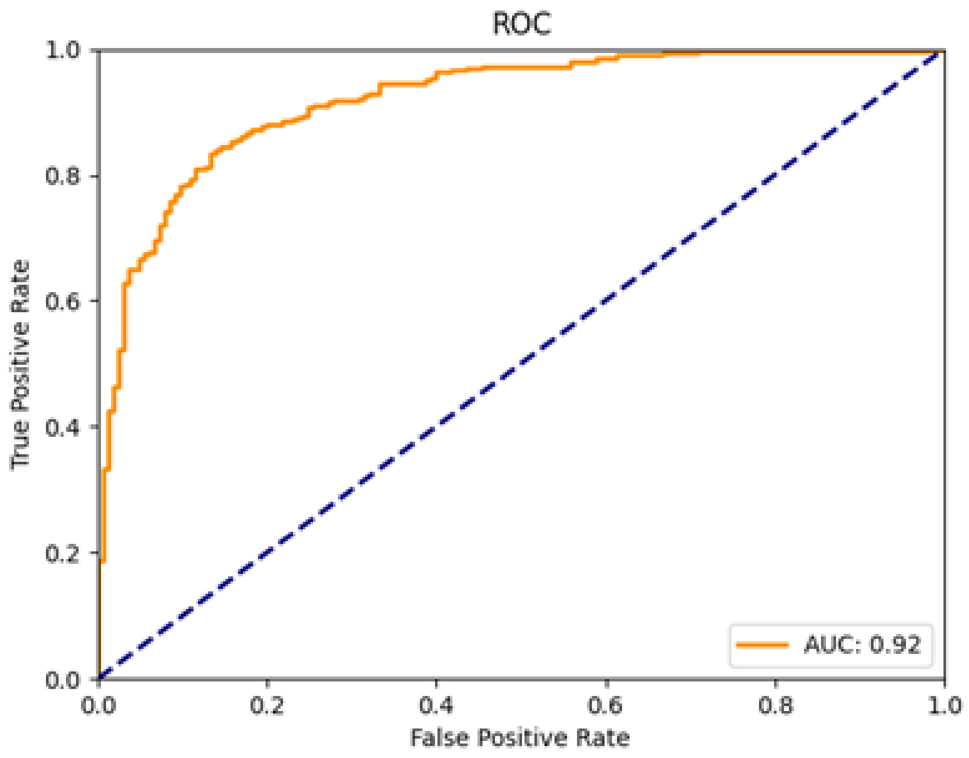


Figure 67 AUROC of the standard PatchCore algorithm on the validation dataset

This result was obtained with the hyperparameters configuration reported in the original paper. In fact PatchCore, as specified in the introduction, is the state of the art method for anomaly detection on the MVtec AD[4] dataset. In other words it is extremely good in industrial applications as this one and in fact it can be noticed that the starting performance is already quite good.

HPO is then applied on this model logging the results on WandB® website. Figure 68 shows the results of this process in terms of the usual parallel coordinates plot.

[4] MVTec AD Benchmark (Anomaly Detection) | Papers With Code

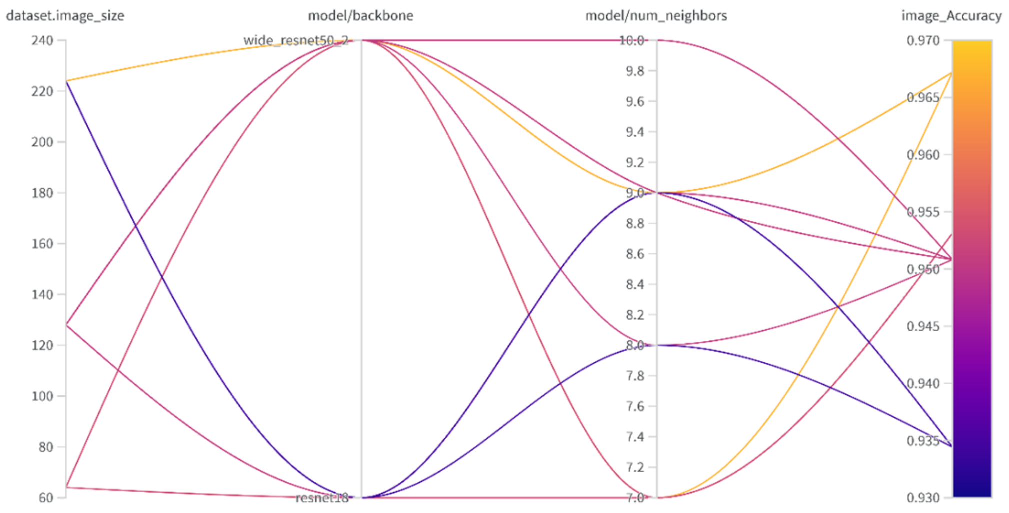


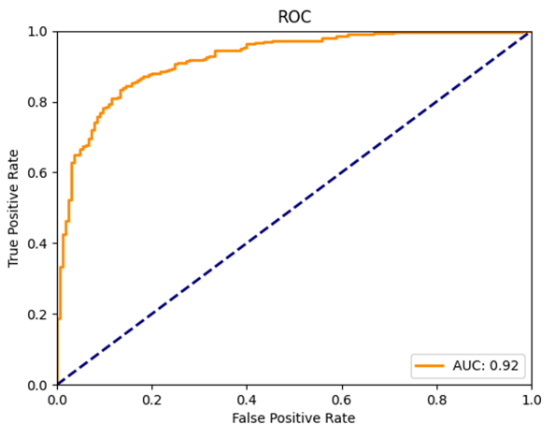
Figure 68 Hyperparameters optimization on PatchCore

Again, the configuration giving the highest accuracy is selected, whose parameters and respective values are reported in Table 21.

Table 21 Optimized valued of the hyperparameters for PatchCore

Parameters	**Values**
Image size	224
Model backbone	Wide_resnet50_2
Number of neighbors	9

Training again PatchCore with these hyperparameters on the specific dataset, gives an increment in the quality of the model that can be visualized in Figure 69 in terms of ROC curve.



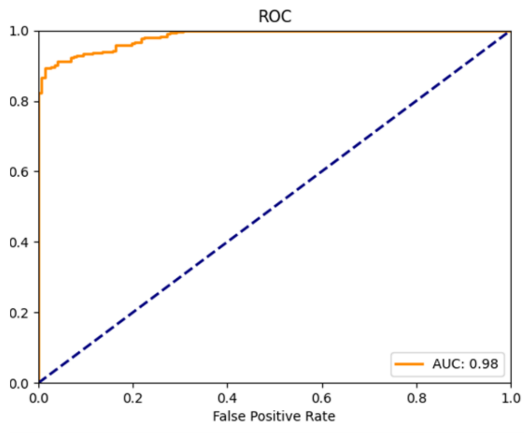


Figure 69 AUROC of PatchCore before and after optimization, the increment is evident

This modified PatchCore model is used to perform K-Fold and in Table 22 the results of the cross validation are listed.

Table 22 Results of cross validation for PatchCore

	Fold n°1	Fold n°2	Fold n°3	Fold n°4	Fold n°5	Mean	Std.deviation
Accuracy [%]	92.12	95.27	95.11	93.53	96.32	94.47	1.65
AUROC [%]	97.40	98.43	98.77	96.62	99.01	98.04	1.02
Inference time per image [ms]	17.5	15.8	15.6	16.6	14.7	16.04	1.06

On these values the bootstrap power analysis is applied. The result of this operation are reported in Table 23. Again we expect these values to be above 0.8 to be able to conclude that we have a sufficiently high number of cross validations. As can be concluded by observing those values we again find that also in this case K=5 is enough.

Table 23 Power analysis on cross validation results for PatchCore

	Power
Accuracy	1
AUROC	1
Inference time	1

3.2.3 DFKDE

The DFKDE algorithm trained and validated on the main dataset, without modification from the version available on GitHub, gives the AUROC reported in Figure 70.

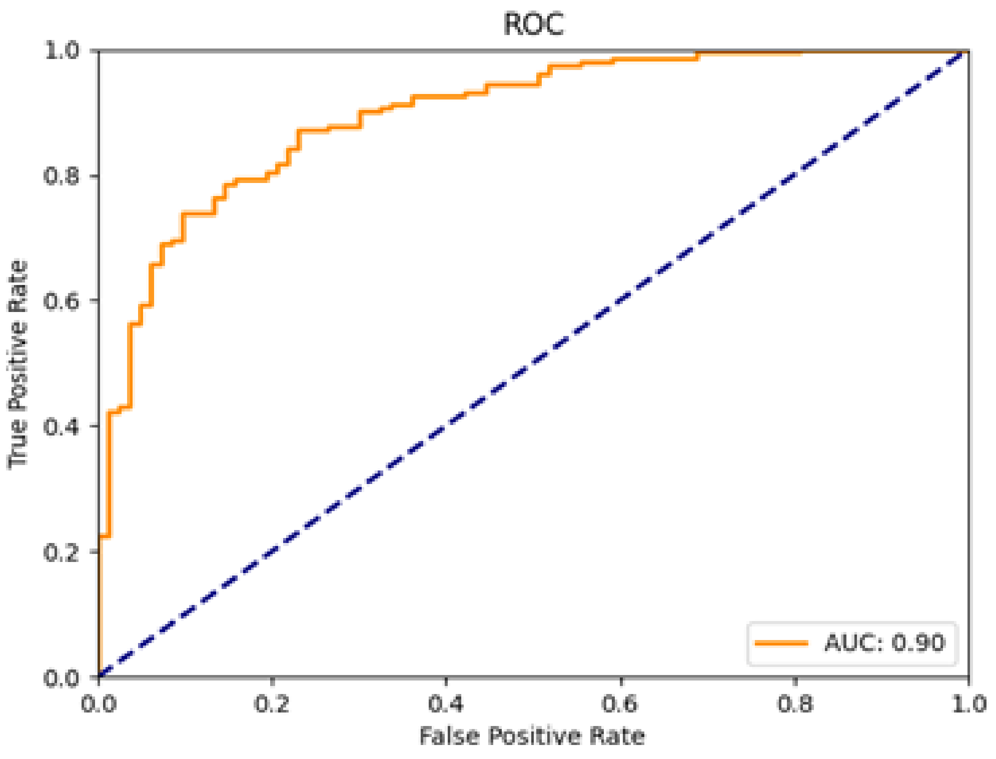


Figure 70 AUROC of DFKDE on validation set with default settings

Again, HPO was applied to obtain the best possible accuracy for the specific case. In Figure 71 the result of the HPO is shown.

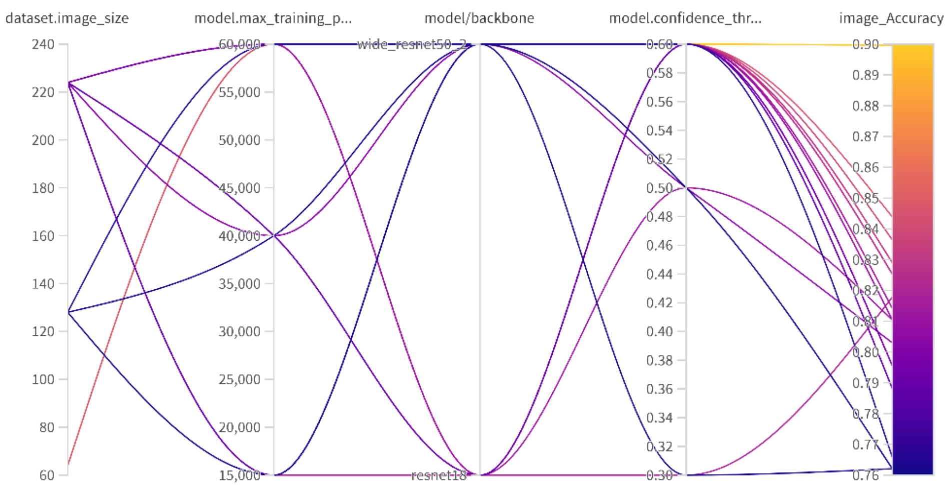


Figure 71 HPO visualization for DFKDE

The best combination of the hyper-parameters values which allow to obtain the highest accuracy is reported in Table 24.

Table 24 Optimized values of hyperparameters for DFKDE

Parameters	**Values**
Image size	224
Model backbone	Wide_resnet50_2
Confidence threshold	0.6
Maximum number of training points	60000

In Figure 72 the comparison between the original AUROC and the one obtained after the HPO is reported. Both were done on the validation dataset.

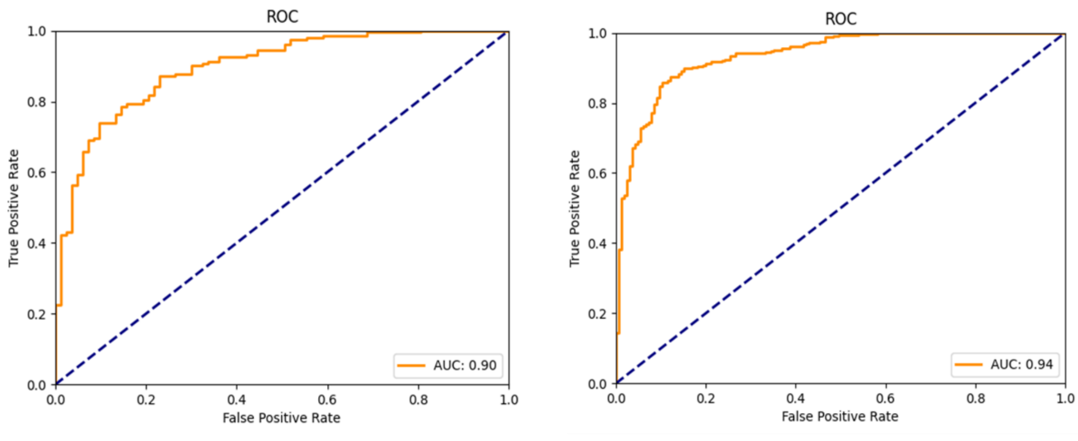


Figure 72 AUROC of DFKDE before and after optimization, the increment is evident

K-Fold cross validation with five folds is performed on the so optimized model. The results are reported in Table 25 and Table 26. Again, the power analysis shows that K equal to five in a sufficient number of splits.

Table 25 Cross validation results

	Fold n°1	Fold n°2	Fold n°3	Fold n°4	Fold n°5	Mean	Std. deviation
Accuracy [%]	84.09	82.51	82.51	83.91	83.39	83.28	$7.5*10^{-1}$
AUROC [%]	93.09	90.25	89.97	91.41	91.08	91.16	1.23
Inference time per image [ms]	11.7	13.1	14.9	13.5	14.08	13.46	1.19

Table 26 Power analysis on cross validation results

	Power
Accuracy	1
AUROC	1
Inference time	0.97

3.3 Comparing the models

Having optimized and cross validated all the models, the results of the comparisons performed among them is reported in this section. In Table 27, Table 28 and Table 29 the performances of all the models are reported in synthetic form.

Table 27 Accuracy results in synthetic form for all models

Accuracy [%]	Fold n°1	Fold n°2	Fold n°3	Fold n°4	Fold n°5	Mean	Std. deviation
ResNet50	99.55	98.81	99.41	98.37	99.55	**99.14**	$4.7*10^{-1}$
MobileNet	98.22	98.51	98.66	98.52	99.11	98.61	$2.9*10^{-1}$
GoogLeNet	98.48	96.59	97.92	97.33	97.92	97.64	$7.18*10^{-1}$
CFLOW-AD	95.34	94.94	95.62	95.22	95.45	95.31	**$2.56*10^{-1}$**
PatchCore	92.12	95.27	95.11	93.53	96.32	94.47	1.65
DFKDE	84.09	82.51	82.51	83.91	83.39	83.28	$7.5*10^{-1}$

Table 28 AUROC results in synthetic form for all models

AUROC [%]	Fold n°1	Fold n°2	Fold n°3	Fold n°4	Fold n°5	Mean	Std. deviation
ResNet50	100.0	100.0	100.0	100.0	99.99	**99.99**	$1.69 * 10^{-3}$
MobileNet	99.99	99.98	99.99	99.99	99.73	99.94	$1.15 * 10^{-1}$
GoogLeNet	100.0	100.0	100.0	100.0	99.64	99.93	$\mathbf{1.42 * 10^{-3}}$
CFLOW-AD	95.70	94.22	95.93	94.97	94.81	95.13	$6.93 * 10^{-1}$
PatchCore	97.40	98.43	98.77	96.62	99.01	98.04	1.02
DFKDE	93.09	90.25	89.97	91,41	91.08	91.16	1.23

Table 29 Inference time results in synthetic form for all models

Inference time per image [ms]	Fold n°1	Fold n°2	Fold n°3	Fold n°4	Fold n°5	Mean	Std. deviation
ResNet50	3.5	3.7	3.6	3.6	3.8	3.6	$5.7 * 10^{-1}$
MobileNet	1.7	1.3	1.3	1.3	1.4	**1.4**	$\mathbf{1.73 * 10^{-1}}$
GoogLeNet	2.6	2.0	2.0	2.3	2.3	2.2	$2.51 * 10^{-1}$
CFLOW-AD	28.5	30.3	25.6	32.2	27.7	28.9	2.3
PatchCore	17.5	15.8	15.6	16.6	14.7	16.0	1.06
DFKDE	11.7	13.1	14.9	13.5	14.0	13.4	1.19

On these results it is now possible to apply the Almost Stochastic Order algorithm to find the best model. As said, since the power analysis' results are good enough for all the models, it is possible to treat every vector of 5 metric values coming from the cross validation as a significant distribution. For what concerns inference times there is no need to report this analysis since there is a clear hierarchy of stochastic order among models. Indeed, MobileNet is for sure the fastest algorithm among the ones analysed.

In Table 30 and Table 31 the results of the ASO applied to accuracy and AUROC respectively are reported. These matrixes are to be read as:

$$ASO(row, column) \quad [\,9\,]$$

On the diagonal the value 1 is set as default. The confidence level is set to 95%.

Table 30 Result of comparison among models' accuracy with ASO. As introduced in the beginning only in values below 0.5 are interesting

Accuracy	Resnet50	MobileNet	GoogLeNet	CFLOW-AD	Patchcore	DFKDE
Resnet50	1	**0.18**	**0.00083**	**0**	**0**	**0**
MobileNet	1	1	**0.028**	**0**	**0**	**0**
GoogLeNet	0.99	1	1	**0**	**0**	**0**
CFLOW-AD	0.99	0.99	0.99	1	**0.43**	**0**
Patchcore	0.99	0.99	0.99	1	1	**0**
DFKDE	1	1	1	1	1	1

Table 31 result of comparison among models' AUROC with ASO. As introduced in the beginning only in values below 0.5 are interesting

AUROC	Resnet50	MobileNet	GoogLeNet	CFLOW-AD	Patchcore	DFKDE
Resnet50	1	0.85	0.83	**0**	**0**	**0**
MobileNet	1	1	0.85	**0**	**0**	**0**
GoogLeNet	1	1	1	**0**	**0**	**0**
CFLOW-AD	0.99	0.99	0.99	1	0.99	**0**
Patchcore	0.99	0.99	0.99	**0**	1	**0**
DFKDE	0.99	0.99	0.99	0.99	0.99	1

As explained when introducing Almost Stochastic Order, among all the values reported in these matrixes, the most interesting ones are the values below 0.5 . In this way it is possible to conclude that algorithm A (row) is better than algorithm B (column). Moreover, when comparing models in terms of sorting quality, AUROC has greater importance than the accuracy for the reason explained in the metrics' paragraph. From these results, some considerations can be extracted. In particular, it is possible to conclude that:

- Classification algorithms have overall better performances than anomaly detection ones both in terms of sorting quality and in terms of inference time.
- The ASO of the AUROC isn't helpful to conclude which among the classification models is the best in terms of sorting quality. Still, looking at the ASO of accuracies ResNet50 is better than both MobileNet and GoogLeNet is worse than MobileNet.
- In terms of inference speed, on the other hand, the fastest model is clearly MobileNet while ResNet50 is the slowest among the classification models.
- GoogLeNet is worse than MobileNet in terms of both sorting quality and inference time.
- Anomaly detection models are overall far slower than classification models.
- Among anomaly detection models DFKDE is the fastest but also the one with the lowest AUROC.
- In terms of AUROC, PatchCore is the best anomaly detection model. Therefore, it is possible to conclude that PatchCore is the best anomaly detection model in terms of sorting quality.
- CFLOW-AD is slower and has lower AUROC than PatchCore.

These observations bring the following conclusions:

- In terms of sorting quality, it is possible to list the models from the best to the worst as follows:
    1. ResNet50
    2. MobileNet
    3. GoogLeNet
    4. PatchCore
    5. CFLOW-AD
    6. DFKDE

- In terms of sorting speed, the same is done as follows:
  1. MobileNet
  2. ResNet50
  3. GoogLeNet
  4. DFKDE
  5. PatchCore
  6. CFLOW-AD

So that it can be observed that:

- Classification models are better than anomaly detection models for this application
- Among anomaly detection models PatchCore is the best choice since it is almost as fast as DFKDE but by far the best in terms of sorting quality.
- Among classification models both MobileNet and ResNet50 have strengths while GoogLeNet is the worst choice. MobileNet is faster while ResNet50 gives higher quality choices. Still, it must be noticed that for both the performances in terms of AUROC and accuracy are very good even for MobileNet. It achieves average AUROC and Accuracy values above 98.5% while being the worse model among the two from this point of view.

Given the need to integrate the here presented algorithms on existing machinery, the Raytec Vision S.P.A. suggested that the doubled classification speed achieved by MobileNet compared to ResNet50 represents a big advantage for which it is permissible to sacrifice a few percentage points in classification accuracy and AUROC. In fact, the inference speed result of MobileNet is the only one low enough to be integrated on the already existing machinery without the need to develop new hardware or reduce the throughput of the existing one. Following this reasoning MobileNet is chosen as the best model. From now on if not specified otherwise we will report some additional experiments performed with MobileNet.

4 Additional results

Chapter 4 is dedicated to some insights. In particular, to have a clear understanding of the work behind this master thesis work, two main topics are analysed and discussed. Firstly, a multi-class classification experiment is performed on the dataset composed by four classes. Then, something about the second dataset is reported. It contains only two classes with about 200 images per class and the acquisition is not good as it is the one of the main dataset. In fact, those are just normal smartphone images. Greater details about it are to be found in the dataset chapter in the introduction. Here the aim is to understand how robust the developed system is with respect to this different dataset.

4.1 MobileNet multi-class classification

In the introduction the fundamental difference in characteristic between IC and AD algorithms is reported. At that stage, it was stated that a fundamental advantage of classification models over anomaly detection ones is that, as the name suggest, IC-based models are able to classify the data in even more than two classes. This property, that is not useful in the industrial application discussed in this thesis, is surely interesting for some other application cases. In fact, it can be used to analyse which type of diseases the classification identifies the hardest. That type of information could be useful in future developments. Because of this, a multi-class classification was performed.

Of course, as written in the introduction, to perform this task a very good dataset with numerous samples from each of the anomalous classes is needed. Luckily the dataset is robust enough and offers three different diseases already divided in folders. Moreover, for each class there is a sufficiently high number of samples to perform this operation.

The framework used for this purpose is identical to what is described above. Just a small number of adjustments need to be done. First, as explained in the introduction, the number of neurons in the final layer needs to be equal to the number of classes we are interested in. To this end, four units were placed in the last layer. Second, as activation function you need to use SoftMax instead of sigmoid. Indeed, the SoftMax function is able to choose which is the higher probability in the case of ranking with more than two. SoftMax applies a standard exponential function to each element of the output layer and then normalises these values by dividing them by the sum of all exponentials. In this way, the sum of all exponential values is equal to 1. To make this passage cleare it is useful to look at the example reported in Figure 73. First, exponentiate each element of the output level and sum the results. It then takes each element of the output layer, exponentiates it and divides it by the sum calculated in the previous step. Clearly, the class given in the output is the one with the highest probability.

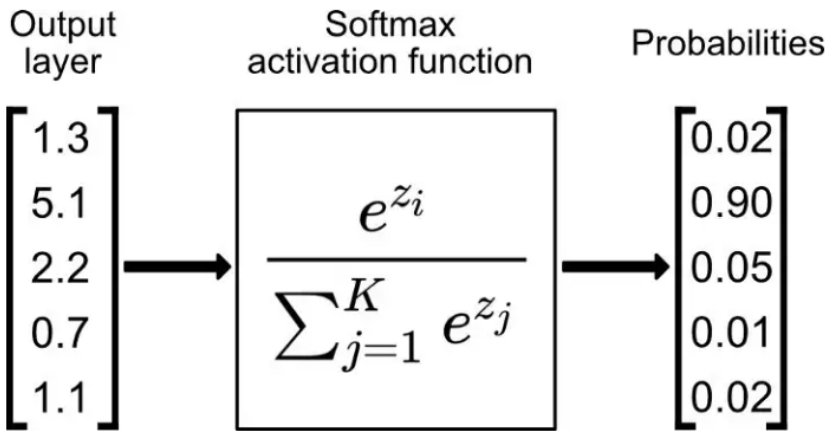

Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

Figure 73 Example of how the softmax activation function works. In this case the number of classes is equal to 5

The change from Sigmoid to Softmax is done autonomously thanks to Python. Moreover, thanks to the black-box scheme and configuration files, implementing these changes becomes quick and easy.

For the multi-class classification, thanks to the low required running time during training associated with good values for both classification accuracy and inference time, MobileNet was chosen as the model to be used. This choice makes it possible not to spend as much time performing this task in the hope of obtaining satisfactory results as in the case of binary classification. HPO was not repeated for this case. Consequently, the values of the hyper-parameters are unchanged from those previously reported. On the so obtained system only training, validation and test procedures are performed. AUROC cannot be used to evaluate the performance in this case. We use accuracy and confusion matrix. In Figure 74 training vs validation accuracy and training vs validation loss can be appreciated. It is possible to state that, even if the hyper-parameters are not the optimal ones, MobileNet generalises pretty well, and the overfitting is not a problem. Indeed, in both accuracy and loss plots there is no sign of divergency. Moreover, the call-back stops the training after just 12 epochs, avoiding a waste of time and ensuring the model does not over-fit.

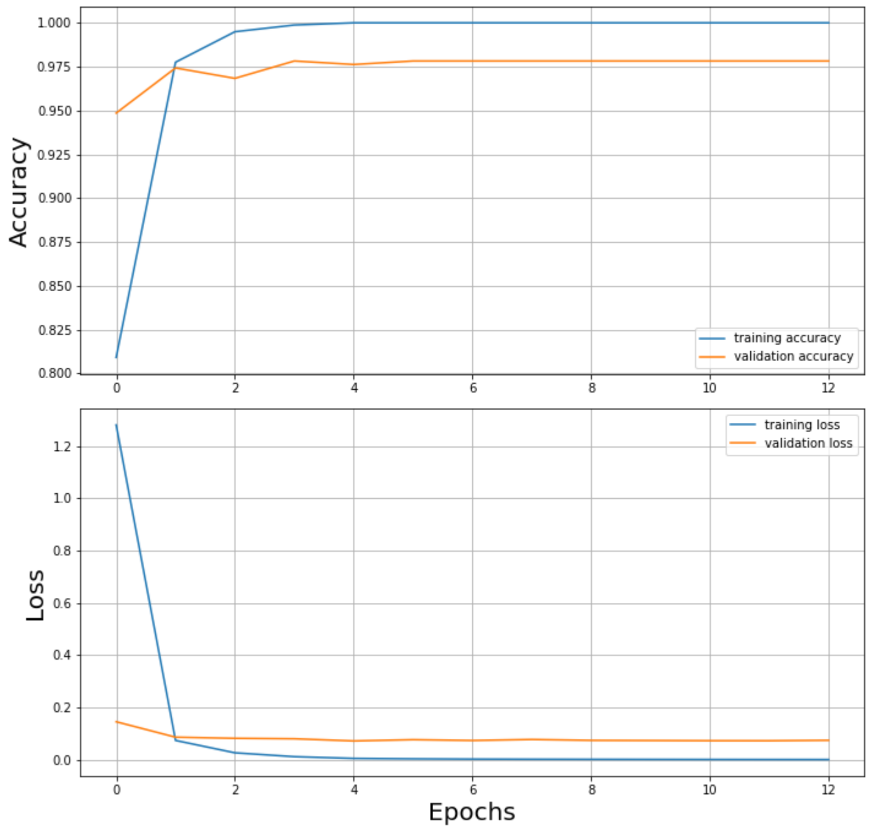


Figure 74 Training vs Loss curves for validation and loss. These curves are obtained in the multi-class class classification task using MobileNet as model.

Identically to what was done for the binary classification problem, the following phase consist in testing the algorithm. The most interesting results are classification accuracy and inference time. The numerical values of these are reported in Table 32.

Table 32 Numerical results in terms of classification accuracy and inference time for MobileNet during multi-class classification

	Accuracy [%]	Inference time per image [ms]
Values	97.04	18.4

Clearly, even for the multi-class classification problem, the best way to understand the result is to plot the confusion matrix. The latter is reported in Figure 75.

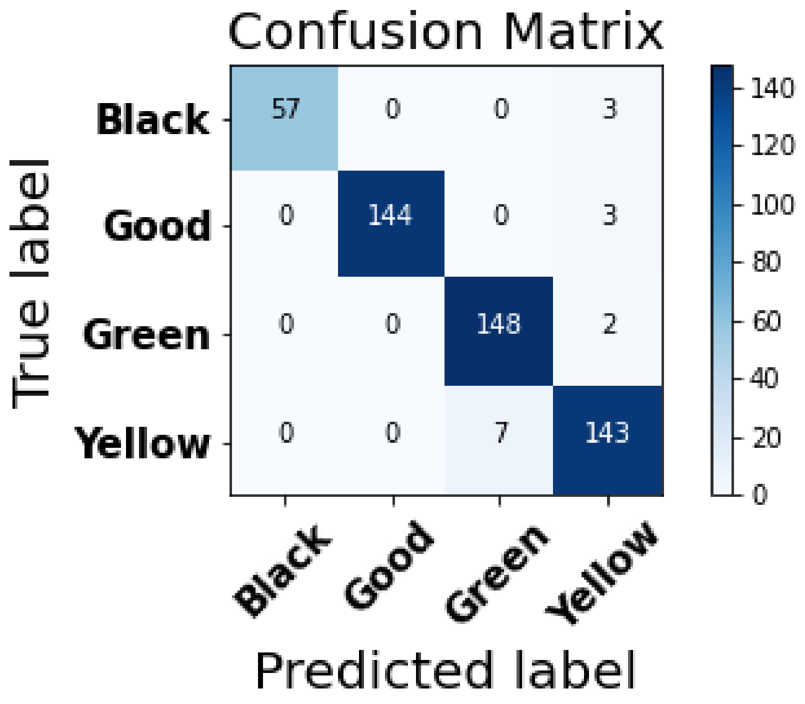


Figure 75 Confusion Matrix for MobileNet during the multi-class classification problem

It can be seen how the algorithm is capable of classifying both good tomatoes and the ones with anthracnose. On the other hand, it still has some trouble in distinguishing green tomatoes as well as yellow ones. Anyway, it is straightforward to conclude that, even without optimizing the algorithm the results are pretty good. In addition, with further improvements in terms of hyper-parameters, a higher accuracy could be achieved.

4.2 Additional Dataset

The additional dataset provided by Raytec Vision S.p.A. was acquired with a normal smartphone and as explained in detail in the dataset section shows the tomatoes on a table so that the background is not standardized and neither the lighting. In addition, in some images, the presence of external elements can be seen (Figure 76). It is composed by only two classes of tomato, peeled and unpeeled. Here this dataset is used to verify how robust the system proposed above is and how much the presence of an uncontrolled background influences the output. It was also useful to conduct some preliminary experiments.

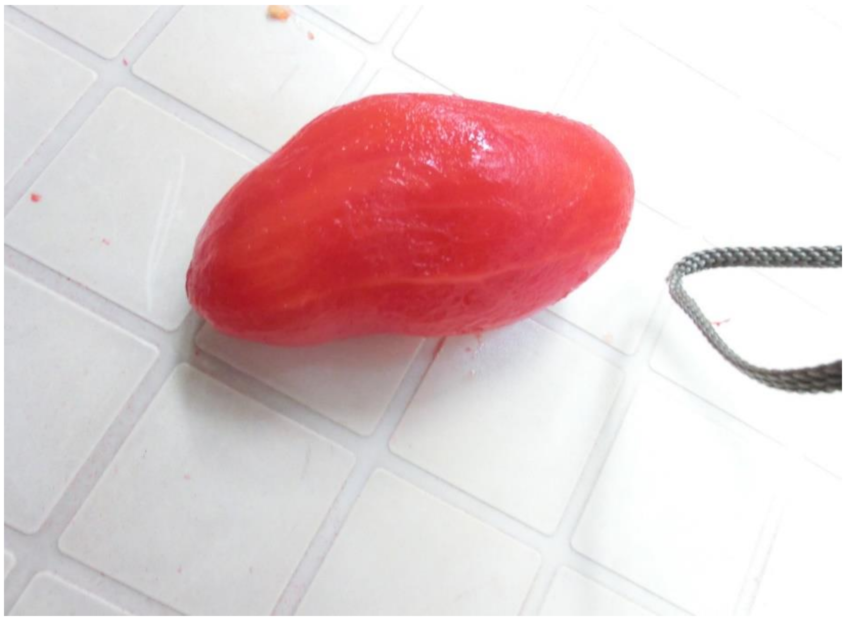


Figure 76 Example of picture taken from the additional dataset

Testing the robustness means to find out if the system, without any optimization and change with respect to what was used above, would still be able to operate in a satisfactory manner on different types of images of the same type of vegetables. This could be useful also when installing the same software with a different acquisition system.

The following models, metrics and result do not need to be explained again because the only change with respect to the binary classification reported above is the dataset. So, in order to not be repetitive, the main results are directly reported. For the same reasons explained in the multi-class classification section (i.e., running time during training and good values for both accuracy and inference time) we chose to use MobileNet for this purpose. This choice is made also since the final machine as said can be imagined working with MobileNet so that the robustness of that model is especially interesting.

About the results, following the same pattern as for the other tasks, we report in Figure 77 training vs validation accuracy and training vs validation loss respectively. Once again, both in accuracy and loss plots there is no sign of divergency. Therefore, the over-fitting does not affect the algorithm.

Once the algorithm is correctly trained, it is possible pass to the test phase, ending with the values for accuracy and inference time reported in Table 33. The confusion matrix is reported in Figure 78.

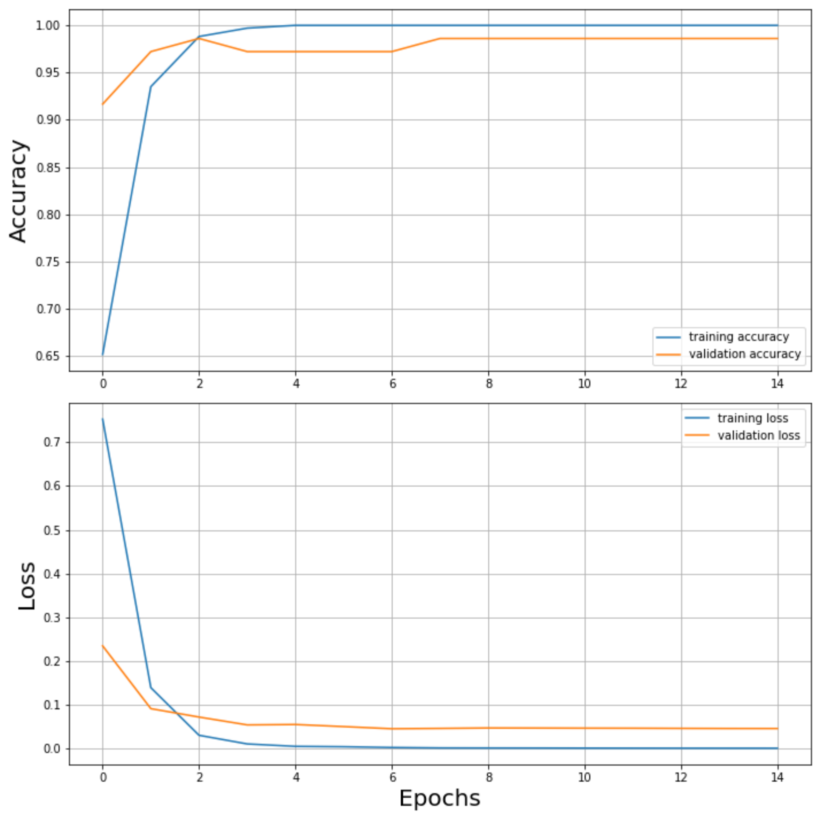


Figure 77 Training vs Validation Accuracy and Training vs Validation Loss for MobileNet on the preliminary dataset

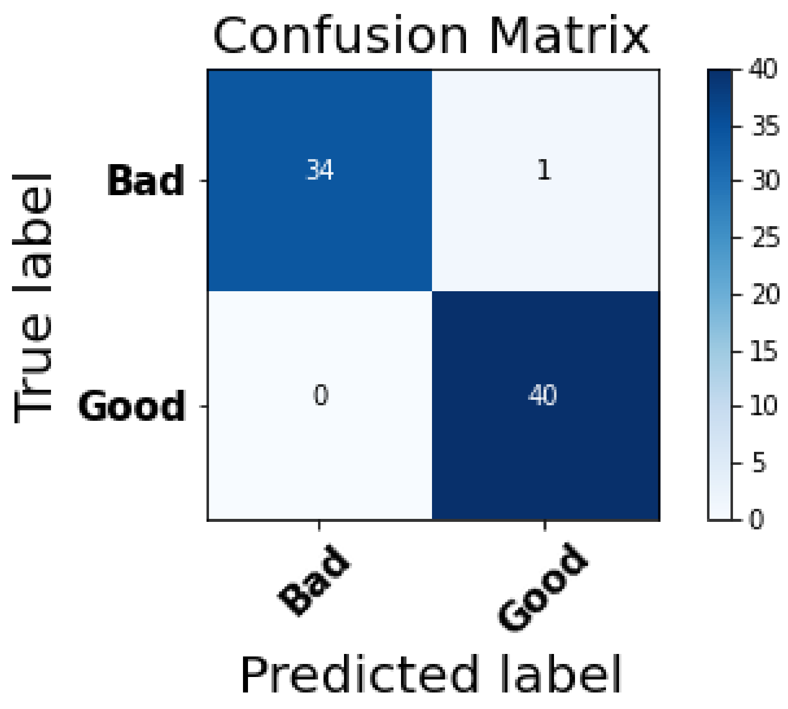


Figure 78 Confusion Matrix for MobileNet on the preliminary dataset

For this testing task, the test dataset is composed by 75 images divided in 35 images of unpeeled tomatoes, labelled as “Bad”, and 40 images of peeled tomatoes, labelled as “Good”. From the confusion matrix, it is possible to see that the algorithm classifies perfectly all the good tomatoes, while it misclassifies one “Bad” tomato. The ROC (Figure 79) follows the confusion matrix plot. As in the previous cases where MobileNet was used, here again the curve identified by the perfect classifier is almost reached.

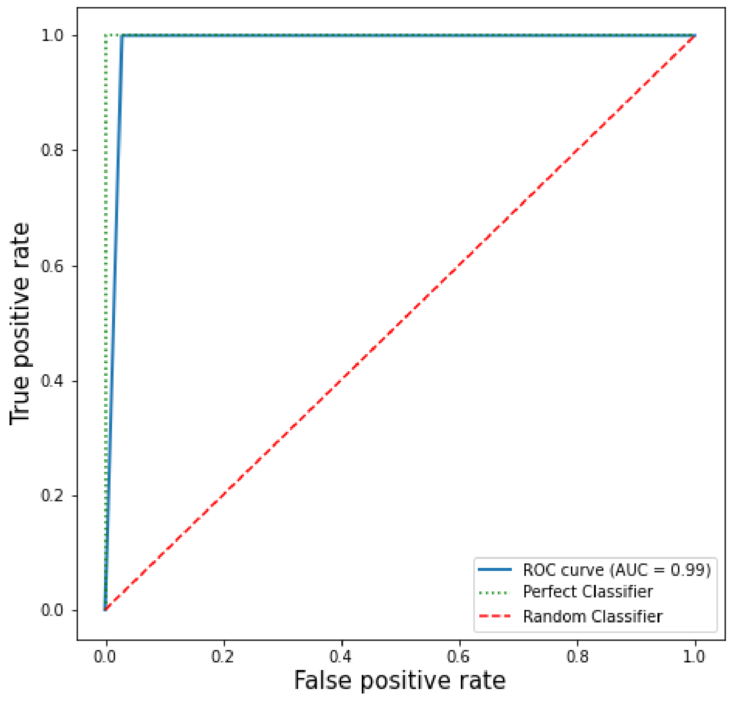


Figure 79 ROC curve for MobileNet on the preliminary dataset

Table 33 Final values of accuracy and inference time for MobileNet on the preliminary dataset

	Accuracy [%]	Inference time per image [ms]	AUROC [%]
Values	98.67	1.9	98.57

As done before, the results obtained in the test phase were validated using K-Fold cross validation, setting five as number of folds. The outcome of the K-Fold procedure is summarized in Table 34.

Table 34 Result of the K-Fold cross validation on the preliminary dataset

	Fold n°1	Fold n°2	Fold n°3	Fold n°4	Fold n°5	Mean	Std. deviation
Accuracy [%]	100	98.97	97.94	100	100	99.38	$8.3*10^{-1}$
AUROC [%]	100	100	100	100	100	100	0
Inference time per image [ms]	6.01	1.38	1.38	1.38	1.45	2.33	1.87

4.2.1 Data Augmentation

Continuing the discussion on the secondary dataset, other procedures need to be reported to make this thesis work complete.

The first procedure that needs to be cited is the well-known data augmentation. Indeed, training a CNN on small datasets makes it prone to over-fitting, inhibiting the CNNs capability to generalize to unseen invariant data. A potential solution is to use Data Augmentation [39]. Data Augmentation consists in creating new data based on modifications of your existing data. Since in our case data are images, data augmentation would include transformations like[5]:

- Flipping the image, either horizontally or vertically.
- Rotating the image.
- Zooming in or out the images.
- Cropping the image.
- Varying the colour of the image.

Following this idea, a series of layers in the model are implemented. Those layers have the task of doing data augmentation directly during training. In this way, the augmented data are not saved, saving space and time. The result of data augmentation starting from a single picture is reported in Figure 80.

[5] Aumento dei dati | TensorFlow Core

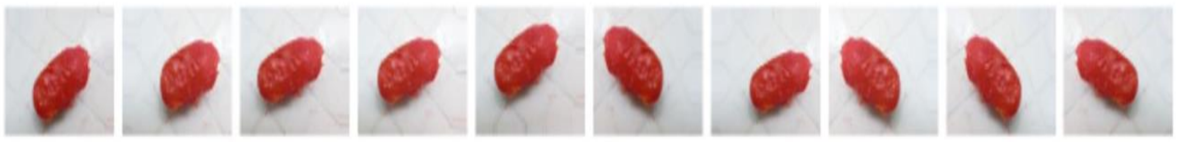


Figure 80 Result of Data Augmentation on the preliminary dataset

Once the piece of code responsible for doing augmentation is robust, it is possible to switch to the usual training phase. The result of the training is reported in Figure 81.

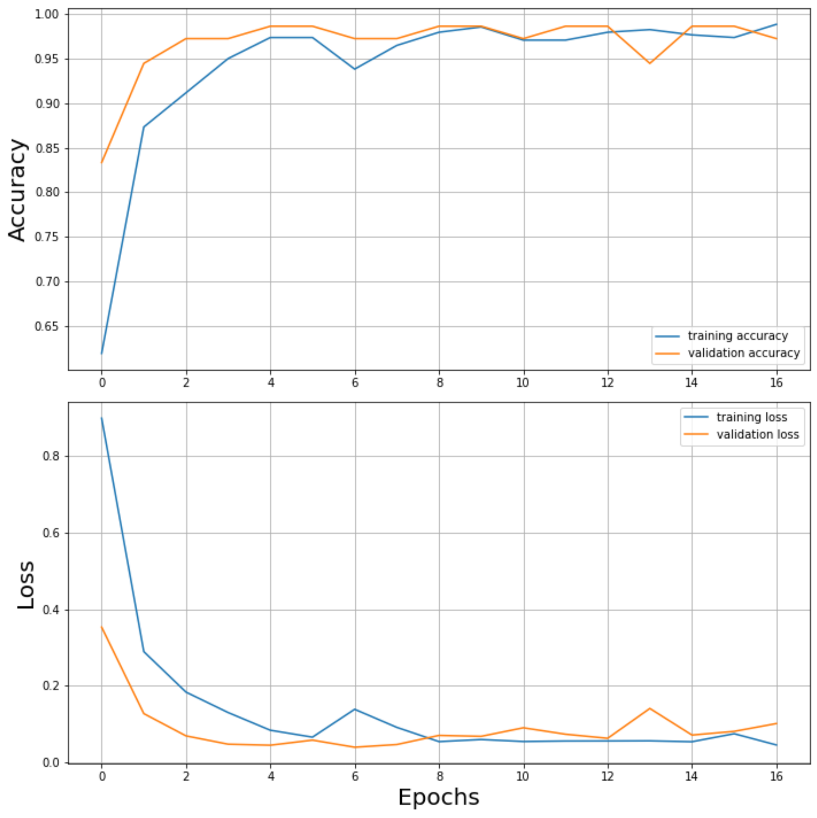


Figure 81 Training vs Validation accuracy and Training vs Validation loss for MobileNet using data augmentation

This time both accuracy and loss plots are not as good as usual. In particular, some oscillations as well as a little bit of divergency can be seen. It is important to remember that hyper-parameters have not been optimised, so expecting perfect curves is not realistic. Once the training is done, the test phase starts. To avoid falsifying test results, you must not apply data augmentation on the test dataset. The results of these procedures are reported in Figure 82, Figure 83 and Table 35.

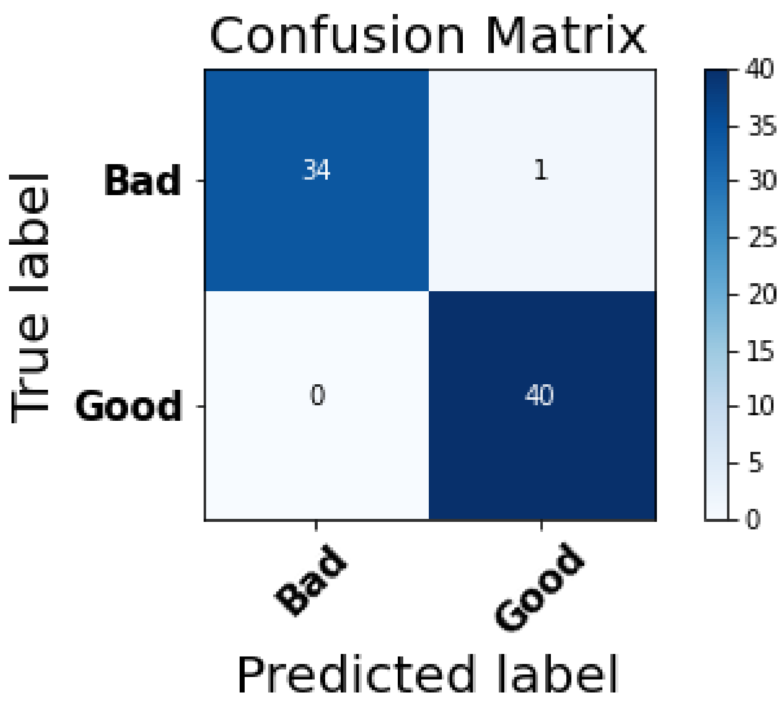


Figure 82 Confusion Matrix for Mobilenet after data augmentation

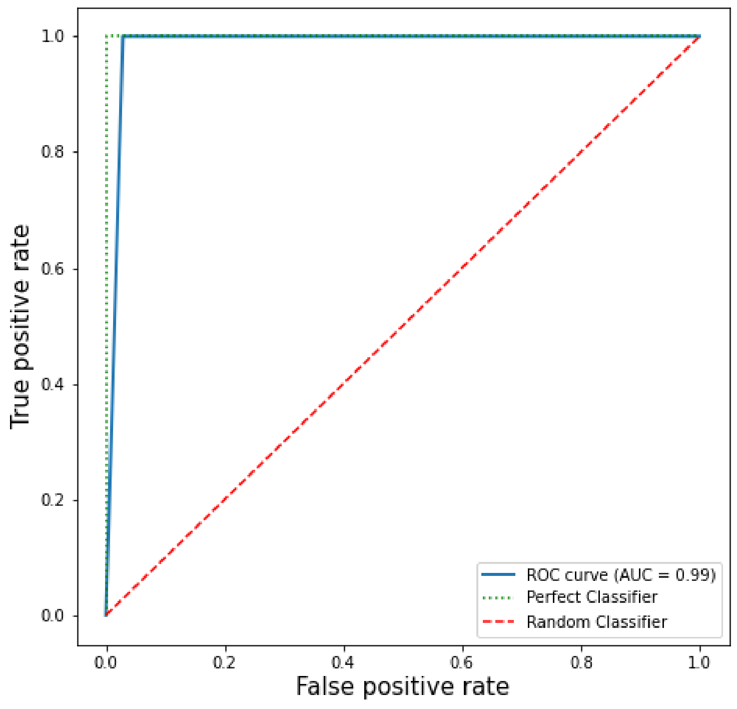


Figure 83 ROC curve for MobileNet after data augmentation

Table 35 Metric values for MobileNet after Data Augmentation

	Accuracy [%]	Inference time [ms]	AUROC [%]
Values	97.33	6.4	98.57

We notice that data augmentation doesn't give a strong increase in performance meaning that the secondary dataset with about 500 images is already large enough.

4.2.2 Random Background Generation

As said the images in the additional dataset did not have controlled lighting and background. For this reason, the influence of the background on the classification results is briefly studied. The background is removed using so-called masks which operate to eliminate everything that is out the range of red and with some morphological transformations. The result of these operations can be seen in the example in Figure 84.

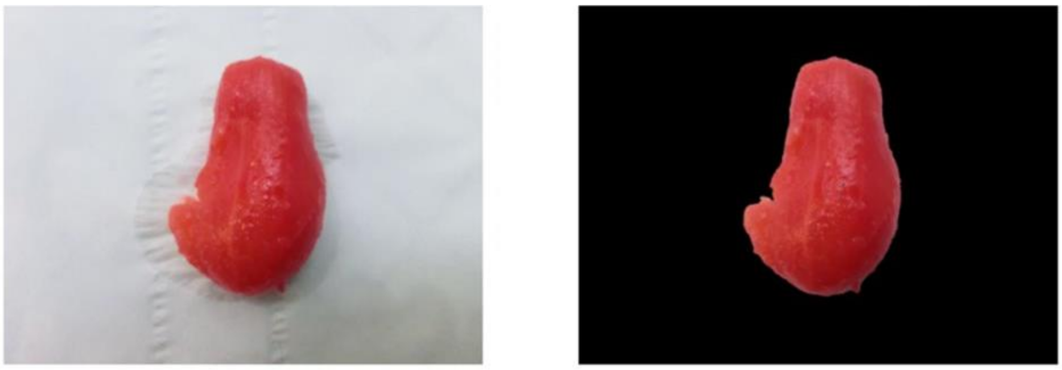


Figure 84 Left: original image. Right: image without background

Another possible step consists in taking the images without background and applying random backgrounds, as in the examples reported in Figure 85 and Figure 86. In this way it is studied if the algorithm is able to work in these conditions and if it is able to generalize the results.

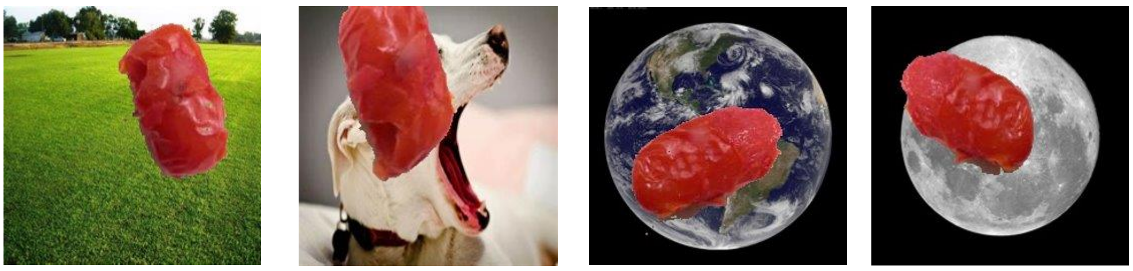


Figure 85 Example of random background application

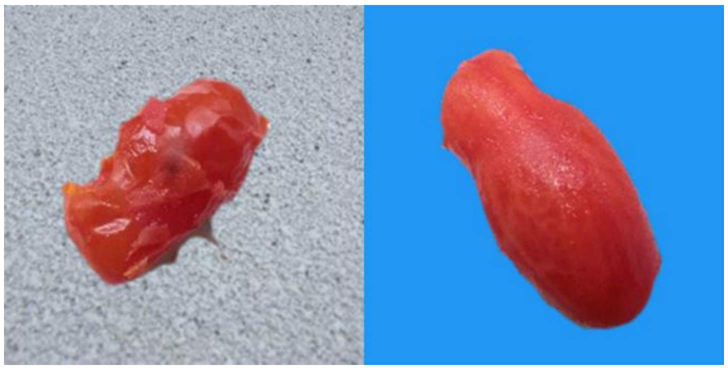


Figure 86 Another example of random background application. In this case is chosen a background that could be similar to the real one.

To avoid being too long in the discussion of this topic, we chose to not report the result in terms of accuracy, Area Under the ROC curve and inference time for this dataset. The reason is that results are similar to ones reported in the base case. In other words, it is possible to conclude that the system is robust with respect to the change of background and that the performances remain high also in these conditions. On the other hand, also no significant increase in performance was observed by simply removing the background and substituting it with a standardized black one. Summarizing, it is concluded that both data augmentation and background removal or substitution did not affect the classification results significantly. This means that about 600 images are enough to train the system in a satisfactory manner and that the system is robust to variable backgrounds.

5 Discussion and Conclusion

This thesis was developed within the context of the project *"Studio e sviluppo di TEcnologie avanzate per il SORting automaticO nei processi di produzione alimentari - TESORO"* funded by the Italian Ministry of Economic Development (MISE). The aim was to show the implementation and comparison of different deep learning-based algorithms for quality control on tomatoes to tackle the specific industrial need issued by Raytec Vision S.P.A.

In particular, both IC and AD-based models were studied to find the best performing solution for operating the quality inspection of the peeling process of the tomatoes. In this manner, an automated solution able to substitute human labour in that procedure was made possible and implemented in the partner's machines.

Starting with the literature search, three anomaly detection and three classification algorithms are identified. State-of-the-art anomaly detection algorithms and image classification algorithms based on neural networks were constructed and compared in a statistically significant manner to identify the most suited one. Each model was adapted to the specific case by means of a black box architecture (i.e., an image is given as input and a classification probability is received as output). By relying on such a scheme, which was easily integrated in the WandB® website, the optimisation of the hyper parameters for maximum accuracy was performed in a standardized manner. The inference time, classification/detection accuracy and AUROC were chosen to measure the performance of the models and optimized. Once the metrics were identified, training and testing of every model were accomplished.

The results of the test phase were analysed by means of the K-fold cross validation scheme. The K-fold results were summarised by reporting the mean and standard deviation. To obtain statistical significance, two statistical methods were identified and applied (i.e., almost stochastic order and bootstrap power analysis). Finally, it was possible to make a comparison of the different models to find the most suitable one for the industrial need. The model chosen, as discussed below, was identified as having the best trade-off among the proposed metrics.

5.1 Milestones

The most important results emerging from this work are:

- For the industrial task analysed both anomaly detection and classification algorithms reach satisfactory results working on simple RGB images. In fact:
    - All models are able to divide faulty images from good ones with an average accuracy on a 5-fold cross validation ranging from 83.2% for the worst model to over 99% of the best one. This suggests two main consequences:
        - The possibility to substitute cumbersome human labour in quality control with automated systems basing on machine vision with one of these models.
        - For the specific task analyses, the use of RGB images is sufficient to perform quality control. There is no need to use hyperspectral or multispectral equipment to obtain satisfactory results on tomatoes. Even the resolution of the RGB images does not need to be high (in this case, a resolution of 224*224 pixels or lower was used) and satisfactory results are still achieved when using the implemented models on simple images taken from a smartphone in uncontrolled light conditions. This demonstrates the robustness of the system.
- The overall performances of classification algorithms are superior to the ones of anomaly detection algorithms for the task analysed. They are both faster and better in terms of AUROC and accuracy. This result was concluded through a significance test with a 95% confidence level. Therefore, the results are statistically significant. For the industrial application studied, the best solution is the implementation of an IC-based model.
- For the needs of Raytec Vision S.P.A., the best model is the MobileNet here implemented. It operates with an accuracy of 98.61% and has an inference time of 0.0015s per image with a realistic GPU setting. With this inference time, it can be integrated with machines like conveyors and existing sorting hardware, which is the core business of Raytec Vision S.P.A.

The here found results could also be interesting for many similar industrial cases that could be automated using one of the presented models. To this end it must be remembered that:

- o Anomaly detection algorithms also operate in cases in which a robust dataset of anomalous samples is difficult or even impossible to achieve. In such situations, the performances of the best anomaly detection model studied can be eye-catching. In fact, PatchCore achieved an average accuracy of 94.47 % on a 5-fold cross validation. The average AUROC is of 98.04% and the average inference time of 0.016s per image.
- o Classification algorithms can also be used in numerous similar industrial applications in which the need is not only to divide the object in good and bad samples, but also multiclass classification is requested. This is demonstrated by trying to identify 4 different tomato classes in the dataset (good, tomatoes with anthracnose, green, yellow) with the chosen MobileNet model and obtaining a 97.04% accuracy.

5.2 Future developments

In future research on this thesis' topics focus could focus on:

- Using the information obtained from the multiclass classification results. These helps in understanding which family of faults creates the biggest problems for the system. Knowing this, it is possible to modify the architecture or the training dataset composition so to be more effective on those specific cases.
- Increasing the anomaly detection's models speed. This could for example be achieved by implementing them in C++. This is useful since with the here achieved inference times it is not possible to integrate those system with already existing sorting hardware.
- Reusing the type of pipeline presented above to implement newer models which may rise in the future. Both IC and AD are active fields of research nowadays in the deep learning field. New architecture and models will be proposed, so that the here proposed models could be overcome by newer ones in a matter of months.
- Using proprietary software which may exhibit some kind of optimized performances with respect to the here presented open-source versions.

List of Figures

List of equations

List of tables

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 2016. [Online]. Available: http://image-net.org/challenges/LSVRC/2015/

[2] M. Nielsen, "Neural Networks and Deep Learning." [Online]. Available: http://neuralnetworksanddeeplearning.com

[3] G. Pang, C. Shen, L. Cao, and A. van den Hengel, "Deep Learning for Anomaly Detection: A Review," *ACM Computing Surveys*, vol. 54, no. 2. Association for Computing Machinery, Apr. 01, 2021. doi: 10.1145/3439950.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks." [Online]. Available: http://code.google.com/p/cuda-convnet/

[5] N. R. Prasad, S. Almanza-Garcia, and T. T. Lu, "Anomaly detection," *Computers, Materials and Continua*, vol. 14, no. 1, pp. 1–22, 2009, doi: 10.1145/1541880.1541882.

[6] V. Lakshmanan, M. Görner, and R. Gillard, "Practical Machine Learning for Computer Vision End-to-End Machine Learning for Images."

[7] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Apr. 2017, [Online]. Available: http://arxiv.org/abs/1704.04861

[8] C. Szegedy *et al.*, "Going Deeper with Convolutions," 2015.

[9] D. Gudovskiy, S. Ishizaka, and K. Kozuka, "CFLOW-AD: Real-Time Unsupervised Anomaly Detection with Localization via Conditional Normalizing Flows," Jul. 2021, [Online]. Available: http://arxiv.org/abs/2107.12571

[10] Karsten Roth et al., "Patchcore: Towards total recall in industrial anomaly detection".

[11] S. Akcay, D. Ameln, A. Vaidya, B. Lakshmanan, N. Ahuja, and U. Genc, “Anomalib: A Deep Learning Library for Anomaly Detection,” Feb. 2022, [Online]. Available: http://arxiv.org/abs/2202.08341

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei, “ImageNet: A large-scale hierarchical image database,” Mar. 2010, pp. 248–255. doi: 10.1109/cvpr.2009.5206848.

[13] C. Zhou and R. C. Paffenroth, “Anomaly detection with robust deep autoencoders,” in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug. 2017, vol. Part F129685, pp. 665–674. doi: 10.1145/3097983.3098052.

[14] D. Bank, N. Koenigstein, and R. Giryes, “Autoencoders,” Mar. 2020, [Online]. Available: http://arxiv.org/abs/2003.05991

[15] S. Jezek, M. Jonak, R. Burget, P. Dvorak, and M. Skotak, “Deep learning-based defect detection of metal parts: Evaluating current methods in complex conditions,” in *International Congress on Ultra Modern Telecommunications and Control Systems and Workshops*, 2021, vol. 2021-October, pp. 66–71. doi: 10.1109/ICUMT54235.2021.9631567.

[16] P. Bergmann, K. Batzner, M. Fauser, D. Sattlegger, and C. Steger, “The MVTec Anomaly Detection Dataset: A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection,” *Int J Comput Vis*, vol. 129, no. 4, pp. 1038–1059, Apr. 2021, doi: 10.1007/s11263-020-01400-4.

[17] B. Mussay, M. Osadchy, V. Braverman, S. Zhou, and D. Feldman, “Data-Independent Neural Pruning via Coresets,” Jul. 2019, [Online]. Available: http://arxiv.org/abs/1907.04018

[18] B. Farnham, S. Tokyo, B. Boston, F. Sebastopol, and T. Beijing, “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems SECOND EDITION.”

[19] USENIX Association., ACM SIGMOBILE., ACM Special Interest Group in Operating Systems., and ACM Digital Library., *Papers presented at the Workshop on Wireless Traffic Measurements and Modeling : June 5, 2005, Seattle, WA, USA*. USENIX Association, 2005.

[20] J. Ghorpade, “GPGPU Processing in CUDA Architecture,” *Advanced Computing: An International Journal*, vol. 3, no. 1, pp. 105–120, Jan. 2012, doi: 10.5121/acij.2012.3109.

[21] S. Chetlur *et al.*, “cuDNN: Efficient Primitives for Deep Learning,” Oct. 2014, [Online]. Available: http://arxiv.org/abs/1410.0759

[22] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” Dec. 2019, [Online]. Available: http://arxiv.org/abs/1912.01703

[23] M. Huh, P. Agrawal, and A. A. Efros, “What makes ImageNet good for transfer learning?,” Aug. 2016, [Online]. Available: http://arxiv.org/abs/1608.08614

[24] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, “Grad-CAM: Why did you say that?,” Nov. 2016, [Online]. Available: http://arxiv.org/abs/1611.07450

[25] N. Detlefsen *et al.*, “TorchMetrics - Measuring Reproducibility in PyTorch,” *J Open Source Softw*, vol. 7, no. 70, p. 4101, Feb. 2022, doi: 10.21105/joss.04101.

[26] L. Ferrer, “Analysis and Comparison of Classification Metrics,” Sep. 2022, [Online]. Available: http://arxiv.org/abs/2209.05355

[27] “Lecture Notes in Artificial Intelligence Subseries of Lecture Notes in Computer Science.”

[28] J. N. Mandrekar, “Receiver operating characteristic curve in diagnostic test assessment,” *Journal of Thoracic Oncology*, vol. 5, no. 9, pp. 1315–1316, 2010, doi: 10.1097/JTO.0b013e3181ec173d.

[29] P. Probst and B. Bischl, “Tunability: Importance of Hyperparameters of Machine Learning Algorithms,” 2019. [Online]. Available: http://jmlr.org/papers/v20/18-444.html.

[30] T. Yu and H. Zhu, “Hyper-Parameter Optimization: A Review of Algorithms and Applications,” Mar. 2020, [Online]. Available: http://arxiv.org/abs/2003.05689

[31] D. Gudovskiy, S. Ishizaka, and K. Kozuka, “CFLOW-AD: Real-Time Unsupervised Anomaly Detection with Localization via Conditional

Normalizing Flows," Jul. 2021, [Online]. Available: http://arxiv.org/abs/2107.12571

[32] A. Bhattacharyya, M. Hanselmann, M. Fritz, B. Schiele, and C.-N. Straehle, "Conditional Flow Variational Autoencoders for Structured Sequence Prediction," Aug. 2019, [Online]. Available: http://arxiv.org/abs/1908.09008

[33] D. Berrar, "Cross-validation," in *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, vol. 1–3, Elsevier, 2018, pp. 542–545. doi: 10.1016/B978-0-12-809633-8.20349-X.

[34] R. Dror, S. Shlomov, and R. Reichart, "Deep Dominance-How to Properly Compare Deep Neural Models," Association for Computational Linguistics. [Online]. Available: https://github.com/

[35] D. Ulmer, C. Hardmeier, and J. Frellsen, "deep-significance - Easy and Meaningful Statistical Significance Testing in the Age of Neural Networks," Apr. 2022, [Online]. Available: http://arxiv.org/abs/2204.06815

[36] P. C. Álvarez-Esteban, E. del Barrio, J. A. Cuesta-Albertos, and C. Matrán, "Models for the assessment of treatment improvement: the ideal and the feasible," Dec. 2016, doi: 10.1214/17-STS616.

[37] A. Hazra, "Using the confidence interval confidently," *J Thorac Dis*, vol. 9, no. 10, pp. 4125–4130, Oct. 2017, doi: 10.21037/jtd.2017.09.14.

[38] K.-H. Yuan and K. Hayashi, "Bootstrap approach to inference and power analysis based on three test statistics for covariance structure models," 2003. [Online]. Available: www.bps.org.uk

[39] S. Sundaram, IEEE Computational Intelligence Society, and Institute of Electrical and Electronics Engineers, *Proceedings of the 2018 IEEE Symposium Series on Computational Intelligence (SSCI 2018) : 18-21 November 2018, Bengaluru.*