
Politecnico di Torino

**Master Degree in Data Science**

**Master Thesis**

# AI-ENABLED AOI: A DEEP LEARNING-BASED INNOVATIVE APPROACH TO IMPROVE THE MANUFACTURING PROCESS

Supervisor

Prof. Barbara Caputo

Candidate

Valerio Di Eugenio

Matricola : 278979

**October 2021**

# Abstract

Reducing scraps has always been a crucial target in the manufacturing industry. In this thesis, an Artificial Intelligence application, integrated with an Automatic Optical Inspection system, is proposed to achieve this specific target. In particular, an unsupervised segmentation based on differentiable feature clustering, combined with a template-matching algorithm, is performed on images to identify and extract the present components one by one. Then, every single component is analyzed to verify whether it meets some imposed requirements necessary to be assembled in the final product. For this purpose, either a deterministic approach based on specific features of the components or a more general solution are tested and the results compared. Finally, to make the application perform on the assembly line, a new innovative architecture designed and build completely from scratch is presented.

# Summary

# 1. Background

## 1.1. Automatic optical inspection systems: an overview

Timely and accurate detection of defects helps industries to apply quality control and stabilization strategies to maintain competitive edge over competition. The target is to get as close as possible to achieve 100% qualified products: among the techniques used to assess the product quality, optical inspection approach for defect detection is one of the most common procedures used in industry. Optical inspection techniques can be performed by human inspectors (manual optical inspection) or in an automatic way by using an image sensor and processor, and the latter takes the name of automatic optical inspection (AOI). The fast and increasing development of technologies has pushed the shift in the success of AOI over manual for quality monitoring. The gap in inspection speed and accuracy played a determinant role and it is harder and harder to be filled: the most advanced systems are capable of detecting tiny defective deviations also with low intensity and contrast difficult to detect through humans naked eyes. Moreover, according to a research carried out by M.-J.-J. Wang and C.-L. Huang [1], human vision inspection capabilities declines with the repetitive dull routine jobs because of the fatigue.

Diving more into the AOI systems, they are able to identify a variety of surface feature defects such as nodules, missing components, scratches and stains as well as the more common dimensional defects

such as open circuits shorts and thinning of the soldier. Two key elements drive this process: the image capturing system and the running application or software for the image analysis.

The former may present many variants that depend on the complexity of the quality control to carry out: it may involve a single camera or more than one to provide a better imaging or even a 3D capture, and there is also the possibility to move them to the best position through a camera-specific software.

In general, the vision systems allow two different kinds of acquisitions (from now on, one single camera is taken as reference):

- Streaming video: the camera takes a streaming video and extracted frames from. The captured frame then enables a still image to be generated on which the signal processing is performed. This approach is not highly accurate but guarantees very high speed.
- Still image capturing: the system is placed relatively close to the target and takes a picture by responding to an external or internal input.

In general, the characteristics described so far are evaluated, with respect to the use case to handle, to build an architecture which guarantees a good balance between accuracy and speed.

The accuracy of the output can be improved also by studying the most appropriate illumination system. It's fairly common that the surface of the components to analyze is enlightened by several light sources and these are carefully chosen according to the way the material of the components refracts the light itself and to the kind of defects to spot: by selecting the correct light type and making it diffuse

homogeneously it is possible to stress out various defects more easily and this leads to a reduction in processing steps as well as to a simplification of the entire task.

The most used types of lighting are the fluorescent, LED, Infra-red or ultra-violet, each of one presenting its strengths and weaknesses. Apart from the kind of lighting, also its positioning plays an equally important role, and it has to be tackled so that the entire area of interest is equally and homogeneously covered.

in the standard AOI application, the image captured is processed and then compared with the knowledge the machine has of what the part should look like, and through this comparison the AOI system is able to detect and highlight any defects or suspects areas. According to this, the most used methods in Automated Optical Inspection application are:

- Template matching algorithm, to identify the parts on an image that match a predefined template given in advance.
- Pattern matching algorithm, to find pre-determined patterns among sequences of raw data.

Recently, AOI algorithms were further enhanced by integrating them with machine learning techniques and deep learning, especially Convolutional Neural Network[2], which often could improve the result and speed up the detection process remarkably. The reason behind is that CNN are built to deal with image data and are able to extract important descriptive features requiring less preprocessing than other algorithms. [3]

As such AOI systems form a very useful element in manufacturing environment: they enable to measure and monitor the quality of the production in a smart and precise way and that is the reason way more and more modern companies are appearing to this technology.

## 1.2. Convolutional Neural Network

Convolutional neural network are more and more widely used and integrated in AOI systems. Being the main structure used in this work, it is important to have a short description of their principal elements and their workflow.

Convolutional Neural Networks represent a huge breakthrough in image recognition. They belong to deep learning family and are recognized to be the state of the art for images classification task. Usually these are composed by convolutional layers for feature extraction and fully connected layers for the classification task. Feature are detected from input images by applying filters (kernels), a set of weights and bias, of smaller size which slide over the whole picture returning the sum of the dot product between them and the filter-sized patch of the input they overlap with. Each filter is designed to spot a particular feature and, by covering all the image in its sliding steps, is able to detect it anywhere in the input. This property is commonly called translation invariance. The depth of the kernel coincide in the majority of the cases (with some exceptions) with the channels of the image and, by computing a sum of the dot products, it returns a single value for each overlapping-patch with the input image,

composing the so called feature map. Each filter returns a single feature map and, as a consequence, the number of feature maps after each convolutional layer is equal to the number of kernel used. In general CNNs apply multiple filters in parallel to gain more information about several patterns to exploit during the classification phase through the fully connected layers. The last layer discriminates over the target classes and a cost function, which gives a quantitative information about how far off the mark the predicted output is, is calculated. The error is a function of the internal parameters of the model and to achieve an accurate prediction it is necessary for it to be minimized and, in neural networks, it is done by backpropagation: the current error is propagated backwards to previous layers where it is exploited to update weights and bias in order to reduce it. The parameters are modified using a function called optimization function: there exist different optimization functions to change the parameters, but all of them are based on the calculation of the partial derivative of the loss function with respect to the wights (i.e. the gradient). Every dimension of the gradient indicates the direction to reach the maximum of the loss function that, instead, must be minimized, and that is the reason why the weights are updated by subtracting the values of the gradient multiplied to a small scalar called learning rate. The learning rate is the hyperparameter that controls how much to change the model in response to the estimated error. Its selection is usually challenging because a small value of it may lead to a long training process that could get stuck, whereas a too large value may result in learning a sub-optimal set of weights too fast or an unstable training process. Furthermore, sometimes it can happen that the

updates are too small, resulting in a globally meaningless step in term of learning. This causes weights to be no more able to change and the convergence will be really slow or absent (problem of vanishing gradient). On the contrary, if the derivative term is too large, the algorithm is not able to reach the minimum of the loss function because it performs too much wide steps(exploding gradient problem). There are different solutions to handle these two problems: to set a different learning rate, to evaluate to add some normalization layers, to use a specific weights initialization and also to use a proper activation function can mitigate them. The activation functions have an important role in the architecture of the net and are useful not only to tackle the abovementioned problem, but these help meaningfully the net learning. These functions keep the output values from each layer restricted to a certain limit: this plays an important role because the input of the activation function is the weighted sum of the values of the previous nodes plus a bias. If the result is not mapped in a limited range, it can go really high in magnitude especially in case of deep network with millions of parameters, causing computational issues.

Another advantages carried by the activation functions is to add non-linearity into a neural network: they allow the model to approximate also non-linear function by creating complex mappings between input and outputs and not only linear correlations, which would be not sufficient to learn at all the relationship among data and the non-linear pattern among them that instead usually occur.

Some of the most common activation function are presented below:

- Sigmoid function: it is a S shaped function which maps the input between [0,1]. It is especially used for models whose aim is to predict the probability as an output, but it is not widely used because it can suffer from the aforementioned vanishing gradient.

$$S = \frac{1}{1+e^{-x}}$$

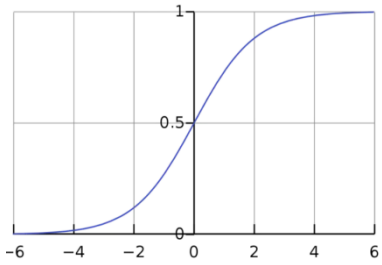


*Figure 1: Sigmoid function*

- Softmax function: it is commonly used in the final layer of a multi-class classification. It takes as input a K-dimensional vector and normalizes it in a probability distribution such that the sum of the elements is equal to own.

The standard unit softmax function $\sigma : \mathbb{R}^k \rightarrow \mathbb{R}^k$ is defined as:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{\xi=1}^{w-1} e^{z_i}} \; for\; i = 1, \ldots, K \; and \; z = (z_1, \ldots, z_K) \in \mathbb{R}^k$$

Where $z_i$ are the *i*-th element of the input vector. The denominator acts as a normalization term and ensures that the output values of the function will sum to 1.

- ReLu (rectified linear activation function): it is a non-linear activation function that outputs the maximum value between zero and the input value.

$$f_{(x)} = max(0, x)$$

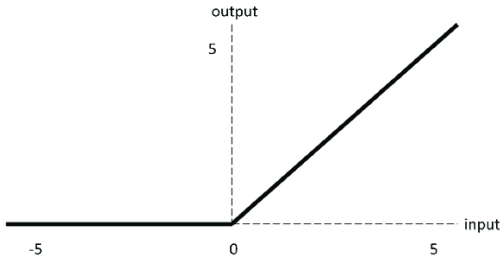


*Figure 2: ReLu activation function*

It overcomes the vanishing gradient problem allowing models to learn faster and perform better: it helps to prevent the exponential growth in the computation required to operate the neural network.

It is also efficient and fast given that not all the neurons are activated. Indeed, it is able to output a real zero output unlike other activation functions which output a value really close to zero. This means that negative inputs can output true zero values allowing the activation of hidden layers in neural networks to contain one or more true zero values. This is called a sparse representation and is a desirable property in representational learning as it can accelerate learning and simplify the model.

The issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly. That means any negative input given to the ReLU activation

function turns the value into zero immediately in the graph, which in turns affects the resulting graph by not mapping the negative values appropriately. [4]

## 1.3. Problem statement

### 1.3.1. Manufacturing point of view

The growing market competitiveness has caused many companies to focus on the cost-effectiveness of their processes and, in the manufacturing field, this attention declines into the control of the three macro-categories of costs: production, qualitative, and maintenance costs. Over the last years, artificial intelligence is proving to be a precious ally to this trend: data-analysis based algorithms are more and more timely in forecasting failures and images-analyzing algorithms are becoming more and more efficient in precisely detecting defects in components.

The production flexibility is one of the pillars of the Lean Manufacturing, whose principles represent the ground for a company to reach the operative excellence. Bosch VHIT, leader in the vacuum and oil pumps in the automotive sector, does not represent an exception: different families of products are worked in a small number of assembly lines extremely flexible, where are processed similar and highly standardized components. This causes the increase of the probability to generate defectiveness during the production process, connected to the assembly of wrong components due to the mixing of these ones among different families.

It is important to go more in depth by carrying on an example to give concreteness to the above statements: at a certain point of the production, a determined component, a rotor in this case, must be

installed into the final pump. There are three different families of rotors, each one marked with a specific number of small circles on the couplings: components with zero circles belong to family A, those ones with one circle to the family B and finally those ones with two circles to family C.

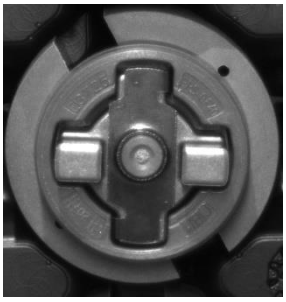

Figure 3: A-class rotor

Figure 4: B-class rotor

Figure 5: C-class rotor

Rotors of different categories have different mechanical characteristics and must be applied to specific pumps for specific clients. In order to do this, the assigned operator pulls it out from a box (also technically called KLT) containing thirty of these, all supposed to belong to the same and right category. This last critical constraint could sometimes miss and it can happen that, in the previous stage, rotors of different families are placed in the same box and passed to the next step. When this situation occurs, the line operator, who is not in charge to visual check if the rotors are correct because of cycle-time issues, assembles these in the final product, but the pumps containing the wrong rotors will result to be scraps in the final quality check out.

*Figure 6: KLT of rotors*

### 1.3.2. Technical point of view

The hardest issue to tackle regards isolating rotors in the input image. In order to spot the location of single components the image semantic segmentation approach is taken into account.

Semantic segmentation is a computer vision task that assigns a semantic label (*e.g.*, object class) to every pixel in an image such that the pixels sharing certain characteristics are assigned the same labels [5]. By doing this, it is possible to partition the digital image into various subgroups, corresponding to particular objects, and separate them from the rest.

This task is particularly challenging when objects involve substantial appearance variations due to changes in pose, scale and illumination, or objects boundaries are distracted by occlusion and background clutter.

From a mathematical point of view, let:

- $I = \{v_n \in \mathbb{R}^3\}_{n=1}^{N}$ be the input image, with *N* equals to the total number of pixels;
- $f = \mathbb{R}^3 \rightarrow \mathbb{R}^P$ be a feature extraction function;
- $x_n \in \mathbb{R}^P$ be a set of *p*-dimensional feature vectors of image pixels;
- $\{c_n \in Z\}_{n=1}^{N}$ be a cluster labels that are assigned to all the pixels, through $c_n = g(x_n)$;
- $g: \mathbb{R}^P \rightarrow Z$ denote a mapping function that returns the label of the cluster centroid closest to $x_n$.

When *f and g* are trainable whereas $\{c_n\}_{n=1}^{N}$ are fixed, it deals with a supervised segmentation approach. $\{c_n\}_{n=1}^{N}$ is known and represents the pixel-level class-specific annotations for each image, providing the model precise locations and boundaries of objects. In the training phase both images and their corresponding annotations are used, and the parameters for *f* and *g* are optimized by gradient descent if *f* and *g* are differentiable.

In general, convolutional neural networks (CNNs) have been successfully applied to semantic image segmentation in *supervised* learning scenarios, for instance, in autonomous driving and augmented reality games sectors. Supervised approach is a really powerful and successful approach, but the scarcity of fully-annotated data, due to their expensive annotations costs, is the biggest obstacle that prevent many deep learning approach from widely applied. [5][6]

To make up for this, *Weakly-supervised* segmentation techniques, which require annotations less detailed than the accurate pixel-level ones, have been taken hold.

Starting from weak annotations such as image-level tags, object bounding boxes, labeled points and scribbles, these aims at generating a training target $\{c_n\}_{n=1}^{N}$ that is, consequently, used with the input images set to update the model in the training phase.

In this direction CNN-based segmentation algorithms have been becoming more and more relevant in the literature: these follow an iterative process that alternates between the two steps aforementioned: (1) training target generation by the weak labels and (2) optimization algorithm for training a CNN-based model from generated target.

The task is challenging because coarse annotation provide no precise information regarding pixel localization whereas sparse annotations (points and scribbles) lack broad region coverage, even though two approaches are developed in order to face these issues: conditional random fields are exploited in order to expand sparse labels to the whole image whereas class activation maps to spot the location of coarse labels and iteratively refine the image segments.

Moreover, the danger of this approach lies into the fact that a possible error in training target generation might reinforce the entire algorithm to update the model in an undesired direction, so the convergence is not always guaranteed. [7]

CNNs are not often used in completely unsupervised scenarios; however, they have great potential for extracting detailed features from image pixels. When dealing with this approach also $\{c_n\}_{n=1}^{N}$ are unknown and to be predicted, there are no annotations to start from but they are randomly predicted and more and more refined with iterations. [8]

Its prediction is jointly optimized with the parameters of *f* and *g* for the image pixels clustering: in other words, it deals with the prediction of the optimal $\{c_n\}_{n=1}^{N}$ with fixed $f$ and $g$, and training of the parameters of *f* and *g* with fixed $\{c_n\}_{n=1}^{N}$.

In this project, this last approach is adopted as the first relevant step for the rotors extraction.[6]

## 1.4. Proposed solution

Before presenting the methods exploited it is important to state that in some steps a prior knowledge about rotors characteristics is exploited to fulfil the declared scope.

The tasks composing the problem are faced in a separate way and then concatenated in an unique flow.

The first one to be handled is the components extraction and it is tackled by combining in sequence three different steps:

1) An unsupervised learning technique for image segmentation is performed to identify the target class.

2) A color isolation phase to divide the interested class from the others. It returns a masked image where the rotors are supposed to be the unique classes to have white pixels.

3) A template matching algorithm to spot the exact location of the rotors. The target to identify has been decided to be a segmented masked rotors and it is then used as the starting point to crop the single ones.

The second one regards the classification of the components extracted. To this end, a standard approach of Convolutional Neural network is adopted in order to discriminate among the different classes.

Further improvements in both the steps are then implemented and the results are compared

# 2. Dataset acquisition

The image used as training set for the unsupervised learning must have been captured directly from the assembly line. For this purpose, the AOI vision system composed by a monochrome camera and an illuminator has been purchased and integrated in a way that the camera, placed in the middle of a hole drilled in the superior part of the bell-shaped illuminator, can frame from above the entire box with the components. A more in-depth description of the architecture and the logic behind will be provided in chapter 6.

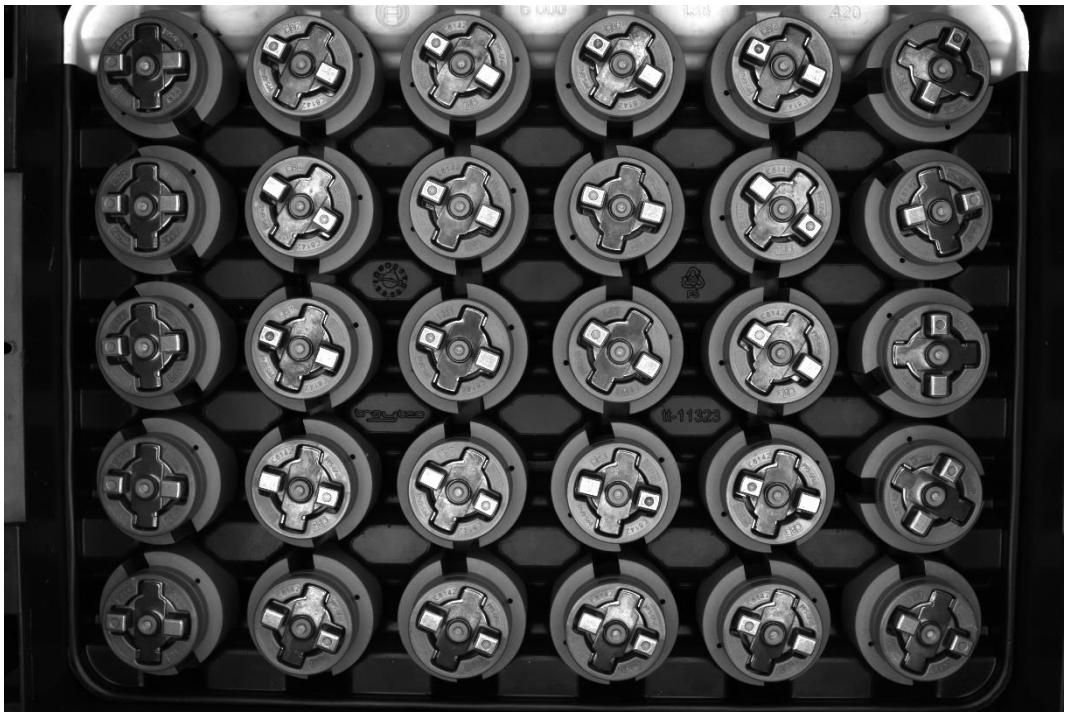

*Figure 7: Image captured by the camera*

A hundred of images are collected for developing the rotors extraction algorithm.

## 2.1. Rotors extraction

## 2.2. Unsupervised image segmentation based on clustering technique

As cited in the first chapter, the process of manual data annotation is time and resources consuming and it is rare to find already annotated data that could be fit for the specific work. That is the reason why an unsupervised learning technique to segment target objects in the images has been implemented. Generally, unsupervised image segmentation is performed to discriminate among general labels, such as “background” and “foreground”. The ideal case would be to mark the rotors as foreground and the rest of the box with a unique background class, but the nature of the dataset makes this task not easily attainable: the assembly line where the vision system is built on is placed in a dynamic working environment where the light condition are not stable in time and with many operators facing their job around there. The illumination certainly mitigates the effects, but the conditions in general are not always the same to guarantee a uniform dataset. Therefore, this approach aims at partition an image into an arbitrary number of salient regions without any previous knowledges: the focus is only on the cluster of the rotors and the rest of the pixels assigned to a unidentified number of labels are not considered. Actually this does not represent a problem but, instead, as described

shortly, turns to be a constraint to make this kind of algorithm well performing.

The milestones of the algorithm implemented are the pixel-level feature extraction and the clustering of the feature vectors which allow to obtain the segments. The architecture chosen to perform the task is a Convolutional Neural Network presenting the following structure:

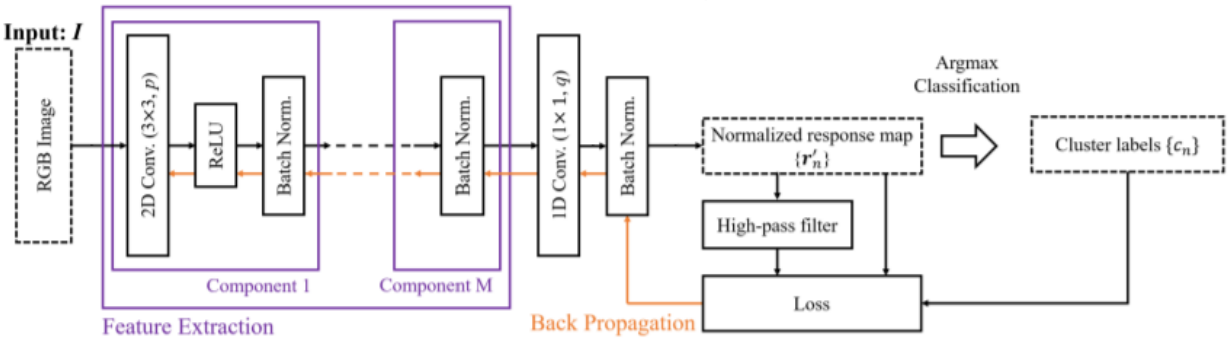


*Figure 8: Net architecture*

The first part consists on a feature extraction module to extract deep feature from a given input RGB image $I = \{v_n \in \mathbb{R}^3\}_{n=1}^N$ whose pixel values are normalized to [0,1]. This feature extraction module consists on two-dimensional convolution layer with *p* filters of size 3x3, ReLu activation function and a batch normalization function which repeat M times with M considered as a parameter to set. Here the batch corresponds to N pixels of the input image. This module outputs a *p*-dimensional feature map $\{x_n\}_{n=1}^N$ that becomes the input of a linear classifier plus a batch normalization function that produces a normalized response map $\{r_n'\}_{n=1}^N$. The final classification is then performed through an argmax function that assigns the cluster label

$c_n$ by selecting the dimension that has the maximum value in $r'_n$. Intuitively, this is equal to clusters the feature vectors into *q* clusters. It is also possible to consider *q* representative points placed in at infinite distance on the respective axis in a q-dimensional space and that each pixels is assigned to the nearest point.

In the training phase, the algorithm jointly optimizes feature extraction and clustering function by minimizing a loss which imposes the following constraints:

*Constraint on feature similarity*: the concept behind is to assign the same label to the pixels that share similar characteristics. These labels are assigned, as already described, by applying the argmax function to the normalized response map, and are further used as the target in a common Cross Entropy loss. The input parameter, instead, is the response map itself. Given this, the constraint on feature similarity is given by the abovementioned Cross Entropy loss which enhances the distribution of the feature vectors assigned to the same class to be as much similar as possible. Formally:

$$L_{sim}(\{r'_n, c_n\}) = -\sum_{i=1}^{n} \mathrm{c}_i \log(r'_{n,\mathrm{i}})$$

The minimization of this loss function ensure the network weights to be updated to expedite the extraction of more efficient features for clustering

*Constraint on spatial continuity*: this constraint encourages the algorithm to assign to a pixel a class which is the same of the

neighboring ones. The idea relies on the fact that pixels spatially close to each other are most likely belonging to the same instance. Mathematically speaking, in order for this to be encouraged, the L-1-norm of horizontal and vertical differences of the response map is computed:

$$L_{con}(\{r'_n\}) = \sum_{\xi=1}^{w-1} \quad \sum_{\xi=1}^{w-1} \quad \left\| r'_{\xi+1,\eta} - r'_{\xi+1,\eta} \right\| + \left\| r'_{\xi,\eta+1} - r'_{\xi,\eta} \right\|$$

Where W and H represent respectively the width and the height of the input image, whereas $(\xi, \eta)$ are the coordinates of the pixel $r'_{\xi,r}$ in the response map.

According to what just explained, the final total loss computed by summing up these two kinds of loss: *L* = $L_{sim}(\{r'_n, c_n\}) + L_{con}(\{r'_n\})$.

Furthermore, another constraint regarding the number of cluster labels is taken into account in the training phase:

*Constraint on the number of unique cluster labels*: in unsupervised segmentation there is no prior information about the number of segments the image should be divided in. In general the number of cluster labels should be adaptive to the image content. As described before, in the final step of each iteration the feature vectors are clustered into *q groups, and it is possible to write the i*-th cluster of the final response map as: $\mathrm{C_i} = \left\{ \mathrm{r'_n} \in \mathbb{R}^{\mathrm{q}} | \mathrm{r'_{n,i}} \geq \mathrm{r'_{n,j}},\ \forall \mathrm{j} \right\}$ with $\mathrm{r'_{n,i}}$ indicating the i-th element of $\mathrm{r'_n}$. $\mathrm{C_i}$ can also be an empty whole and therefore the number of clusters labels can fall in a range between 1 and *q*, and let $q'$ denote that number. Initially a large number for *q* is

set, then with the iterations similar and spatially close pixels are integrated in the same groups, causing a reduction of $q'$. The aforementioned constraints embedded in the loss function encourage the grouping pixels and this could lead to the simplest solution $q' = 1$ (Fig 9). Mathematically, this is translated in the intra-axis normalization process, through batch normalization, from $\{\mathrm{r_n}\}$ to $\{\mathrm{r'_n}\}$ before applying the argmax function.After the normalization each axis has zero mean and unit variance and this gives to each $r'_{n,i}$ an even chance to be the maximum value across the axes. This operation ensures that many cluster indices achieve the maximum value for any n=1,..N (remembering N being the total number of pixels). As a consequence, this leads to a preference of having a large $q'$.

It is not properly a constraint: *q* states for the dimensions of the tensor after the fully connected layer.

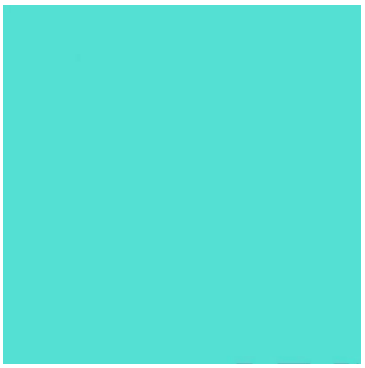

*Figure 9: Output of the model trained with a low value of q due to pixels grouping*

Basically, the training phase is composed by two parts: the forward process where the cluster labels are predicted for each pixels, and a backward step where the network weights are updated through the

loss backpropagation. This forward-backward process is repeated K times to obtain the final clustering labels prediction. [6]

The algorithm is trained by using the hundred images collected and iteratively validated by computing the average loss on 30 images randomly selected from a wide database to control the training trend

## 2.3. Template matching algorithm

Template matching algorithm is an high-level machine vision technique that identifies the parts of a given image that match an input predefined template and it is useful in locating certain features in a given image. More in depth, provided a reference image of an object and the image to be inspected, the algorithm identifies all input image locations at which the object from the template image is present [10]. The template slides at every possible location over the input image and each time a similarity measure, called image-correlation, is computed in order to give a quantitative indication regarding the analogy between the template and the portion of the image it overlaps with.

Among the existent similarity measures, for this study the Normalized Cross-correlation is selected: it consists on a simple sum of pairwise products of corresponding normalized pixel values of the images [11]. In formula:

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

In general, the normalization is necessary in order to strongly mitigate the bias towards brighter pixels. The multiplications, in fact, yield higher results for brighter part in the image because bright pixels have an higher numerical value than dark pixels. Through the normalization, the operations are computed between pixel values in ranges [0,1] instead of in [0,255].

Before applying this algorithm, another preprocessing step is executed in order to facilitate the template matching work: in the segmented mage, the color that most likely corresponds to the rotors class is extracted by selecting the one associated with the cluster labels assigned to the majority of the pixels and successively all the instances of that color are firstly isolated and then masked, setting their value to (255,255,255) corresponding to white, whereas the rest to the black (0,0,0). This kind of preprocessing is applied in order to clean the image from potential noises due to the fact that a completely unsupervised segmentation could lead to a not completely precise discrimination of the object instances, leaving some pixels belonging to some classes misclassified with another class label.

It is important to say that the Template matching is a naive approach: the algorithm is sensible to every tiny deviation from the input template and it is not able to recognize a corresponding instance in the image if it is differently oriented.

Many studies are carried on in literature in order to tackle this issues and to make the algorithm more robust and rotation-invariant (like

Greyscale-based Matching or Edge-based matching algorithm [12][13]), but for this study the basic naive approach is sufficient for the scope: a picture representing the ideal segmented rotor is used as a template: it presents some asymmetries but they do not affect the performance.

*Figure 10: Rotor template*

The algorithm returns the coordinates of the pixel corresponding to the upper left corner of the portions of the image whose similarity measure with the template is greater than an arbitrary threshold. This one is set empirically: it is taken the average value of the similarity measures over five hundreds of matches and it is decreased of three time its standard deviation.

At this point, starting from the returned coordinates and exploiting a prior knowledge about the rotors size in terms of pixels, every component is cropped in the original input frame and the thirty rotors image are obtained.

The images obtained are initially used as training set for the classification algorithm. During the finale flow, instead, every extracted rotor will be fed into the subsequent model.

# 3. Rotors classification

## 3.1. Architecture

The second step of the project consists on the classification of the components extracted to check if they all belong to the same and right family. As discussed before, 'right' means that the class of the rotors must be the one that fits with the final products the operators are assembling in that specific moment. To this end, a variable indicating the class to use is exposed in the IoT Gateway which manages data of the assembly lines and it is read by the python script through an API call. A more in depth description of the hardware architecture and the integration logic implemented is discussed in the next chapters.

For the scope, the AlexNet structure is exploited: AlexNet was the first Deep convolutional neural network to achieve meaningful results on the 2012 ImageNet LSVRC-2012 challenge, where the input was an image of one of a thousand of different classes, obtaining an accuracy of almost 85% against the 74% of the second-best one. [14]

It consists of 5 convolutional layers and 3 fully connected layers. The filters of convolutional layers have a size of (11,11) in the first one, decreases to (5,5) in the second until (3,3) in the others. The first, the second and the fifth convolutional layers are followed by an overlapping Max Pooling layer, and the output of the last one feeds the fully connected layers. Each layer of the new is followed by a ReLu activation function which sets all negative values to zero. This was the innovation introduced with AlexNet architecture which allows a much

faster training with respect to the architectures having Tanh or Sigmoid activation functions. [15]

The dataset used to train the net is composed by the rotors extracted in the previous step. For the purpose, a training set and a validation set of respectively two thousands and seven hundreds images are used.

Data augmentation is performed in order both to reduce the overfitting scenario and to make the model train with components presenting as many rotation angles as possible: in fact, rotors can present different rotation with respect to its center and not being convolutional neural networks  rotation invariant, this could cause them to consider the orientation as discriminative feature for some classes, that must be avoided.

In particular three different transformations are performed:

1. Vertical flip on the images with probability of 0.5 to be applied.
2. Horizontal flip on the images with probability of 0.5 to be applied.
3. A random rotation of the images of 45 degrees with probability 0.5

# 4. Results

In the following chapter the results are reported and analyzed. In particular, the results are shown before for the single algorithm analyzed and then the best model are aggregate in a unique flow and the overall accuracy is computed.

In detail, the final dataset is composed by 1000 KLT images and the algorithm will mark each as good or scrap by applying in sequence the selected models.

The metric used to evaluate the performance of the algorithm is the accuracy, given the nature of the project: one single rotor image misclassified leads to marking the KLT the rotors is as scrap. According to this, the accuracy is the best metric to give a quantitative information about the overall performances.

For each combination of hyperparameters, the reported accuracy and loss values are the average of 5 runs and only measures on the validation set are considered.

## 4.1. Unsupervised segmentation technique

Premise: each model performs in a different way assigning independently every class to a random color. Even using the same dictionary {class : “color”} it is not possible to track the assigned classes and force different models to classify with the same labels the pixels supposed to be in the same group. That is the reason why the color of the labels across different models will be different.

The following table show the results obtained:

| Learning rate | Epochs | Optimizer | N° Conv Modules | Similarity loss weight | Continuity loss weight | Final validation loss |
|---|---|---|---|---|---|---|
| 0.01 | 300 | SGD+MOM | 2 | 1 | 1 | 2.35 |
| 0.01 | 700 | SGD+MOM | 2 | 1 | 1 | 1.70 |
| 0.01 | 1000 | SGD+MOM | 2 | 1 | 1 | 0.97 |
| 0.01 | 1000 | SGD+MOM | 2 | 1 | 1 | 0.93 |
| 0.001 | 700 | Adam | 2 | 1 | 1 | 1.06 |
| 0.001 | 1000 | Adam | 2 | 1 | 1 | 0.95 |
| 0.001 | 2000 | Adam | 2 | 1 | 1 | 0.72 |
| 0.001 | 3000 | Adam | 2 | 1 | 1 | 0.52 |
| 0.001 | 3000 | Adam | 2 | 1 | 5 | 0.53 |
| 0.001 | 3000 | Adam | 2 | 1 | 5 | 0.97 |
| 0.001 | 1000 | Adam | 2 | 1 | 10 | 1.22 |
| 0.001 | 1500 | Adam | 2 | 1 | 10 | 0.92 |
| 0.001 | 1000 | Adam | 2 | 1 | 50 | 1.43 |
| 0.001 | 3000 | Adam | 2 | 1 | 50 | 0.49 |
| 0.001 | 1000 | Adam | 2 | 1 | 100 | 1.85 |
| 0.001 | 5000 | Adam | 2 | 1 | 100 | 0.65 |

The starting number of labels is arbitrarily set to 100 and kept for each run, so as the batch size which is kept to 10. The Adam optimizer from the beginning has provided better results in term of validation loss and therefore it has been more used. Finally the number of epochs has been augmented with the increase of the continuity loss weight: the more the magnitude, the more the number of epochs needed from the algorithm to converge. The similarity loss weight also has been tested

but it didn’t carry to meaningful improvements, therefore the results are not shown in the table.

*Figure 11: input image*

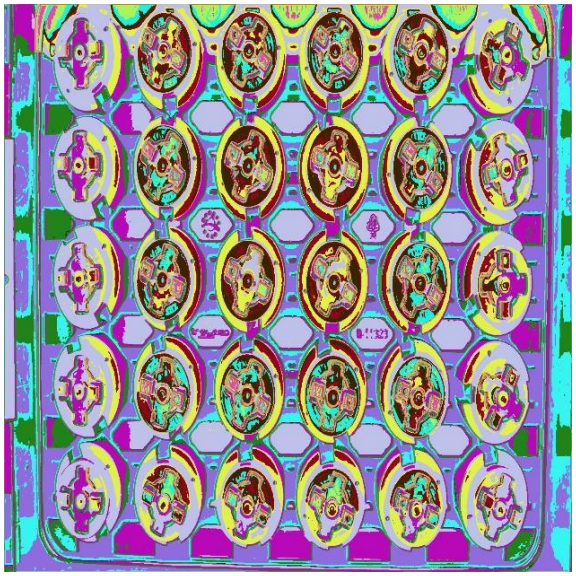

*Figure 12: Continuity loss weight:1*

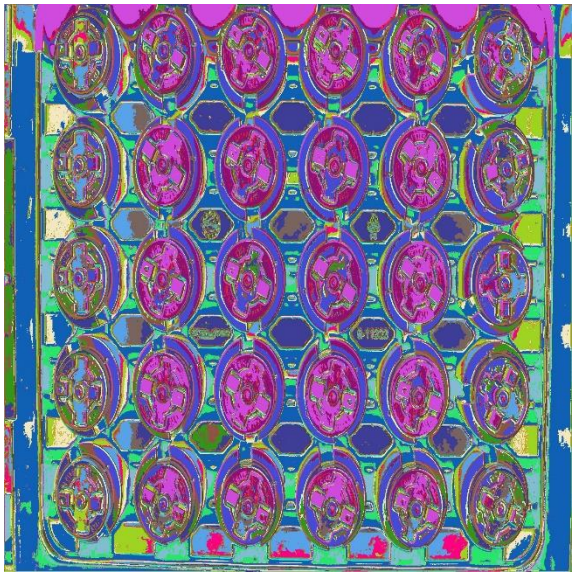

*Figure 13: Continuity loss weight:1*

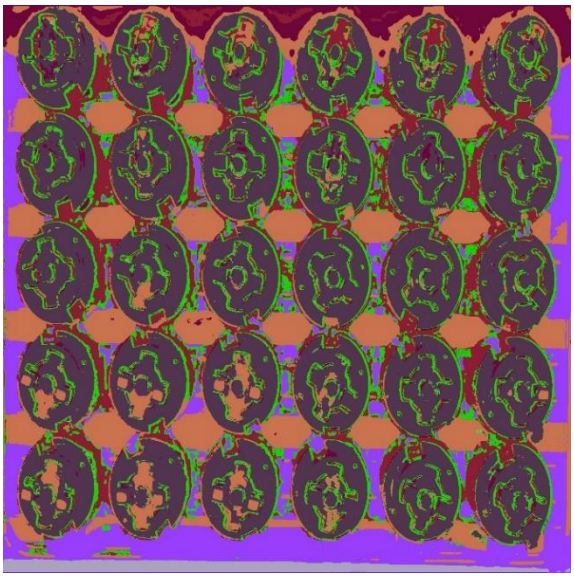

Figure 14: Continuity loss weight:50

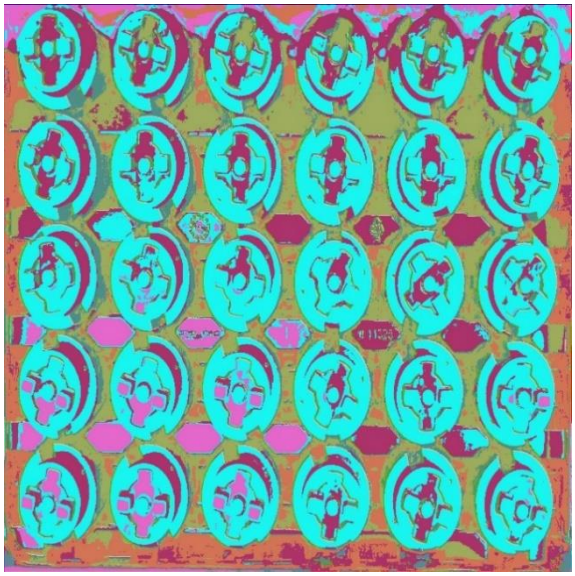

Figure 16: Continuity loss weight:50

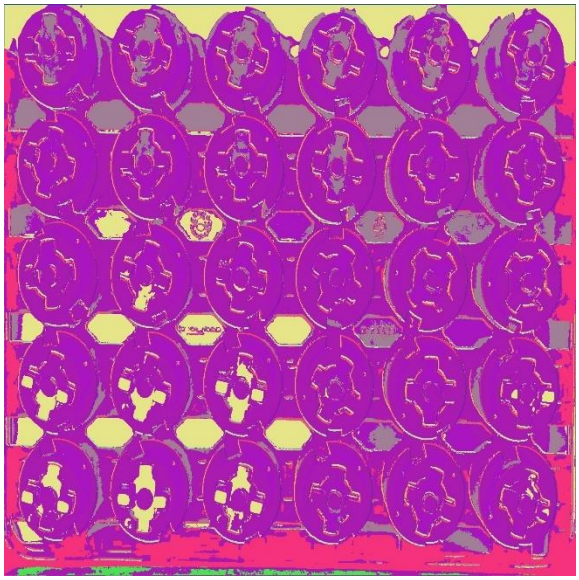

Figure 17: Continuity loss weight:100

Figure 15: Continuity loss weight:100

The figures represent the outputs of different models tested on the same input image. The table has shown as the hyperparameter that most affected the others is the weight of the continuity loss, and that's the reason why it is the only one reported in the captions.

These outputs throw the light on how this substantially makes the models perform in different way: the less it is the more the image is segmented in detail. In fact, the segments of the images in Fig.1 and Fig.2 spot also small texts and digits on the background and are highly sensible to any deviation on the surface, while in those in Fig.8 and Fig.9, with the maximum weight tested, the pixels are far more grouped expanding the rotor class also to the background.

It is inferred that the optimal weight changes depending on the degree of details it is desirable.

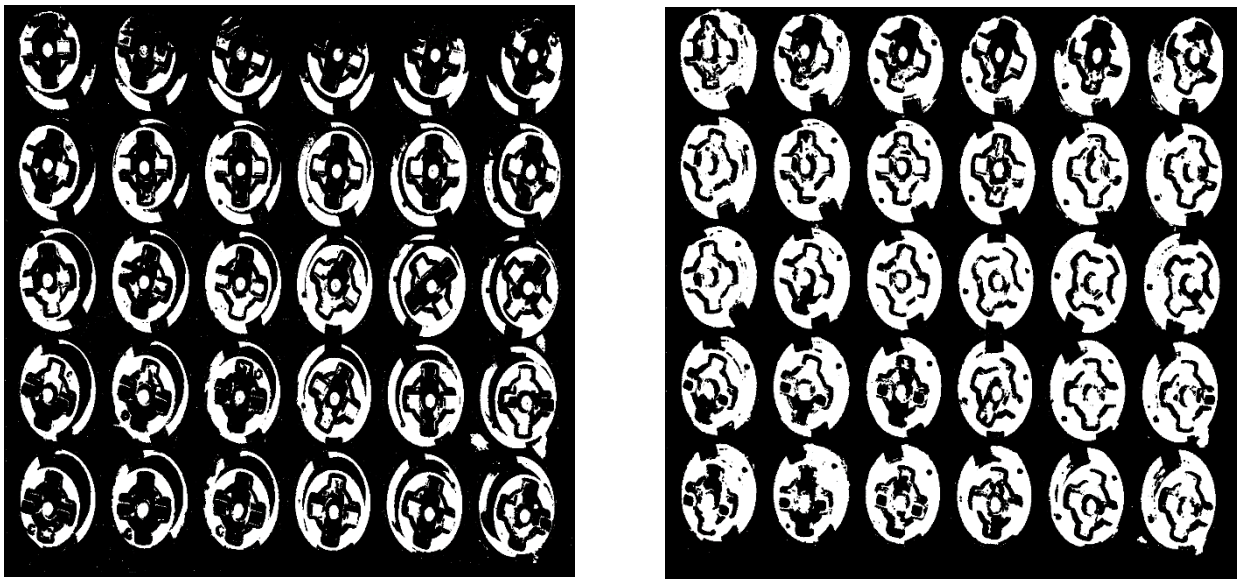

*Figure 18: examples of resulting masked image obtained after segmentation with continuity loss weight = 50 plus color isolation*

## 4.2. Template matching algorithm

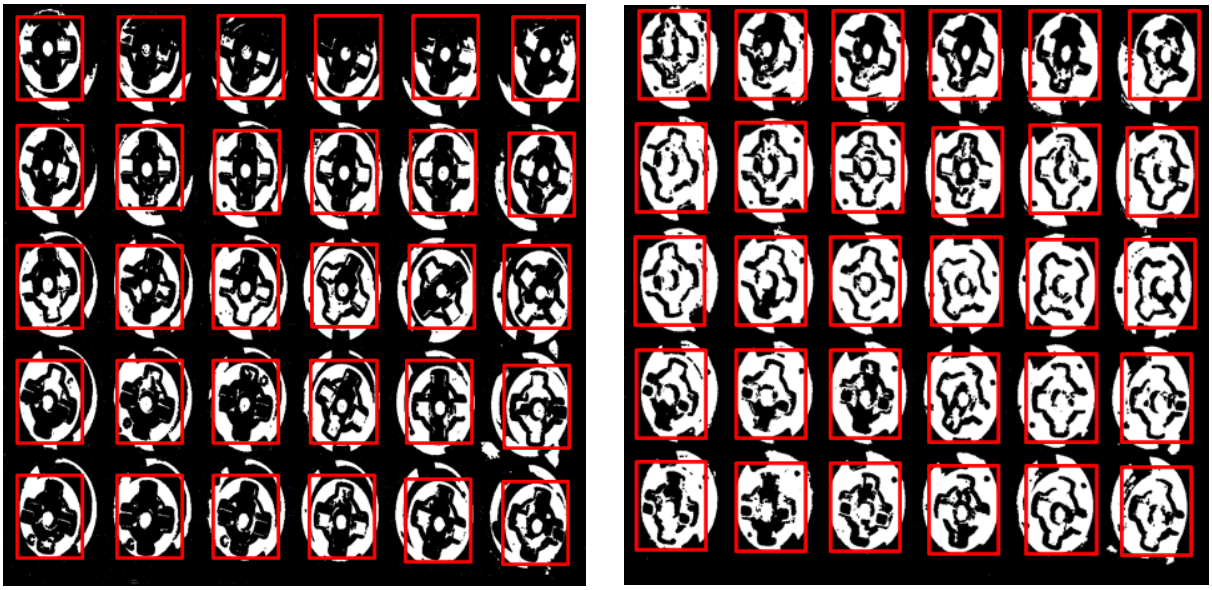

*Figure 19: Output of the template matching algorithm*

Having set the right threshold as discussed in chapter 2.3, the algorithm is able to spot every rotors in the masked images. According to this, the coordinates of each one is extracted and used to crop the rotors in the initial input image, obtaining thirty single rotors images to use in the classification phase.

## 4.3. Classification with CNN

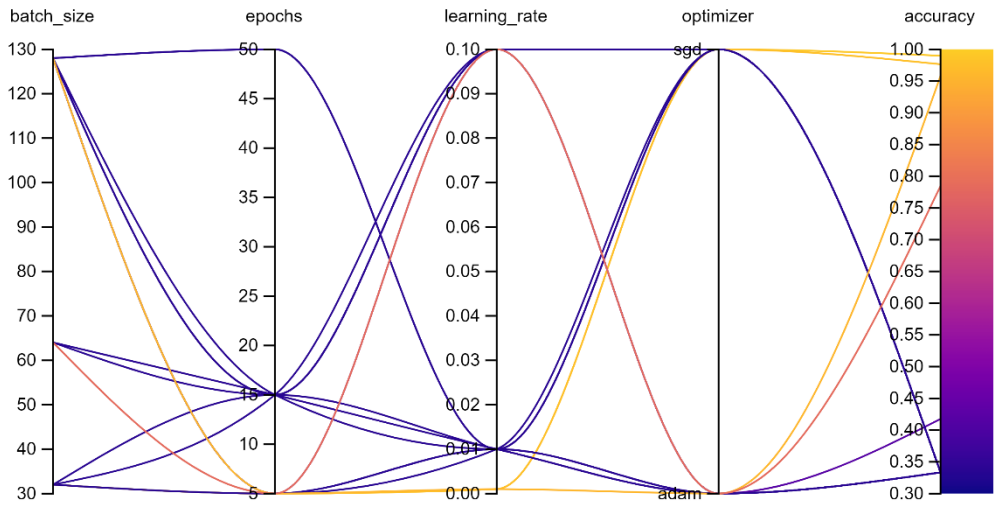


*Figure 20: accuracy over different combination of parameters*

Fig.10 shows the accuracy on the validation set over different combination of hyperparameters tested. The images are not so hard to discriminate, the translation-invariance property of the CNN spots the discriminative feature in the image regardless its location, resulting in a quite simple classification. A simple hyperparameters tuning is performed to gain hyperparameters that guarantee good performances.

The table below shows some of the best results obtained

| **Epochs** | **Learning rate** | **Optimizer** | **Batch size** | **Step size** | **Weight decay** | **Val. Accuracy** | **Val. Loss** |
|---|---|---|---|---|---|---|---|
| 20 | 0.001 | Adam | 64 | 10 | $5e^{-5}$ | 0.94 | 0.21 |
| 20 | 0.001 | Adam | 64 | 5 | $5e^{-5}$ | 0.93 | 0.32 |
| 10 | 0.001 | Adam | 128 | 5 | $5e^{-5}$ | 0.93 | 0.29 |
| 10 | 0.0001 | Sgd+Mom | 128 | 5 | $5e^{-5}$ | 0.93 | 0.43 |
| 15 | 0.0001 | Adam | 128 | 5 | $5e^{-5}$ | 0.95 | 0.19 |
| 15 | 0.0001 | Sgd+Mom | 32 | 5 | $5e^{-5}$ | 0.94 | 0.20 |

## 4.4. Final results

The two selected model are finally combined and the overall performances evaluated:

| Segmentation | | | | | Classification | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **LR** | **Epochs** | **Optim** | **Similarity loss weight** | **Continuity loss weight** | **LR** | **Epochs** | **Optimizer** | **Batch size** | **Accuracy** |
| 0.001 | 3000 | Sgd+Mom | 1 | 1 | 0.001 | 20 | Adam | 64 | 0.34 |
| 0.001 | 3000 | Sgd+Mom | 1 | 1 | 0.001 | 20 | Adam | 64 | 0.33 |
| 0.001 | 1000 | Adam | 1 | 50 | 0.001 | 10 | Adam | 128 | 0.86 |
| 0.001 | 3000 | Adam | 1 | 50 | 0.0001 | 10 | Sgd+Mom | 128 | 0.85 |
| 0.001 | 3000 | Adam | 1 | 100 | 0.0001 | 15 | Adam | 128 | 0.28 |
| 0.001 | 3000 | Adam | 1 | 100 | 0.0001 | 15 | Sgd+Mom | 32 | 0.22 |

The results clearly shows how the final accuracy is determined by the Similarity loss weight.

For the current project, the segmentation target is to separate as well as possible the rotors from the background in order to make them more distinguishable so that the template matching phase is able to retrieve the right coordinates to extract them precisely. For this reason, models optimized with a continuity loss weight equal to 50 are the most suitable for the scope and, indeed, is the one used in the combination reaching the best performances.

Models with continuity loss equal to 100 tend to excessively group pixels and when the main color in the image is isolated the rotors are

not clearly distinguishable, confusing the template matching algorithm. Similarly, segments of images output by a model trained with a continuity loss weight equal to 1 are much more detailed and the main color does not cover enough pixels to output a sufficiently recognizable rotor-shape when isolated.

# 5. Improvements

## 5.1. Convolutional Block Attention modules

One on the main pillar of the unsupervised segmentation technique implemented in the first part of the process is the extraction of the features from the input image. The aim of the improvement proposed is to make CNN learn and focus more on the important information rather than learning non-useful background information. To this end, channel and spatial attention modules are integrated in the feature extraction part of the net. The attention idea is inspired by the human few-shot learning behavior: to recognize a sample of any object, humans tend to locate firstly the most relevant regions which better discriminates the objects under the lens. In a similar way, given a feature map, spatial and channel attention modules generate an attention map for each feature to highlight the target object. Since convolution operations extract informative features by blending cross-channel and spatial information together, the improvement proposed aims at emphasizing meaningful features along those two principal dimensions: channels and spatial axes. To this end, channel and spatial modules are sequentially applied in each block of the feature extraction phase. [16]

*Channel attention module*: it produces a channel attention map by exploiting the inter-channel relationship of features. Since each channel of feature map is considered as a feature detector, channel attention focuses on ‘what’ is meaningful given an input image. It

firstly aggregates spatial information of feature map by using both average-pooling and max-pooling operations, generating two different spatial context descriptors: F_{avg}^c and F_{max}^c which denote respectively average-pooled features and max-pooled features. Both descriptors are then separately forwarded to a multi layer perceptron with one single hidden layer, and the output feature vectors are merged by using a element-wise summation. The hidden activation size is scaled by a factor of ten in order to reduce the parameters and to make the architecture computationally less heavy.

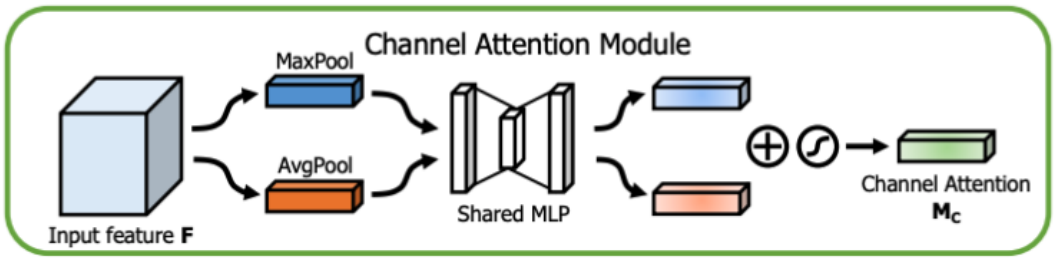


*Figure 21: Channel attention module*

Basically, the channel attention module is computed as:

$$M_c(F) = \sigma(MLP(AvgPool(F)) + MLP(MaxPool(F)))$$

Where σ denotes the sigmoid function.

*Spatial attention module*: The spatial attention module is obtained by exploiting the inter-spatial relationship of features, focusing on 'where' is located an informative part. To compute it, it is firstly applied an average-pooling and max-pooling operations along the channel axis, which is demonstrated to be effective in highlighting

informative regions, and the resulting maps are concatenated. The resulting descriptors is convolved by a dedicated standard convolutional layer to generate the final spatial attention map, that is then normalized by the sigmoid function.

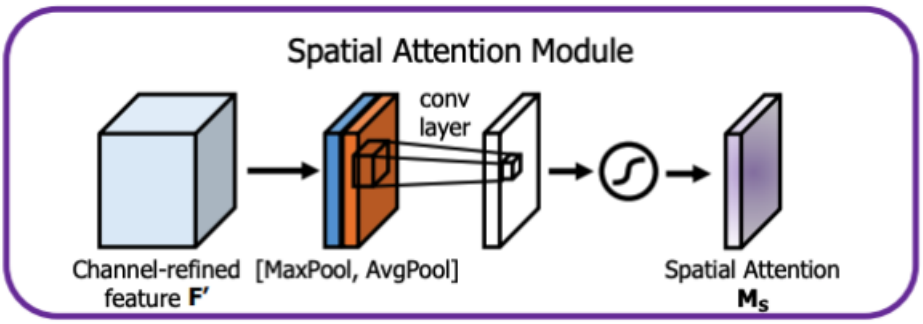


*Figure 22: Spatial attention module*

In formula, the 2D spatial attention map is obtained as follows:

$$M_s(F) = \sigma(f^{7x7}([AvgPool(F); MaxPool(F)]))$$

Where $\sigma$ still denotes the sigmoid function and f^{7x7} the filter size 7x7 convolutional operator. A larger kernel is used in order to increase the receptive field of the network.

The generated attention maps are leveraged in a sequential way:

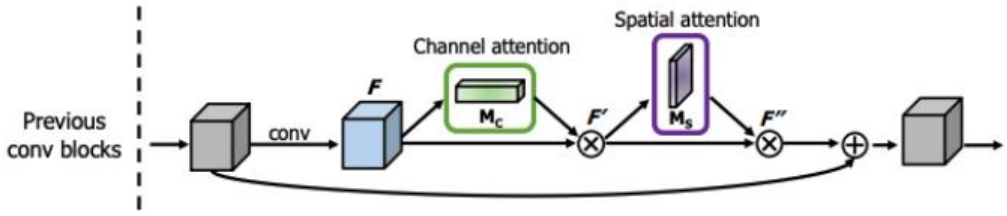


*Figure 23: entire attention modules flow*

First of all, the Channel attention map is applied on the input map, followed by the application of the spatial map.

$F' = M_c(F) \otimes F$

$F'' = M_s(F') \otimes F'$

During the multiplication, channel attention values are broadcasted along the spatial dimension and viceversa to make the operation possible. The obtained feature map $F''$ is finally summed to the previous input convolutional layer. [17]

## 5.2. Hough circles transform

Hough transformation is a method used to find primitive object forms such as lines and circles, as for in this case. Basically, through Hough Circles Transform is it possible to locate circles in a given image by determining its parameters. [18]

The first step is to find any shape in an image is to detect the edges of the objects and isolate them from the rest. The most common approach to perform this task is the canny edge detector which is briefly explained in the following steps:

1. Application of the gaussian blur to smooth the image in order to remove some noise. It corresponds to convolve the image with a kernel. Being the image in two dimension (it has to be firstly converted to Grayscale) the same one dimensional kernel is used to blur the image in both the dimensions, one at a time. The blurring operation is performed pixel by pixel to the one which is at the center of the kernel everytime: to blur the highlighted pixel, its value is set to a new one corresponding to the weighted average of the color values of the pixels within the

kernel. The weights are assigned according to the closeness to the central pixel according to the gaussian function $G(t) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$ where x corresponds to the distance from the center pixels, and $\sigma$ controls the shape of the function. According to this, the original pixel's value receive the highest weight and the neighboring a smaller one depending on how they are far away.

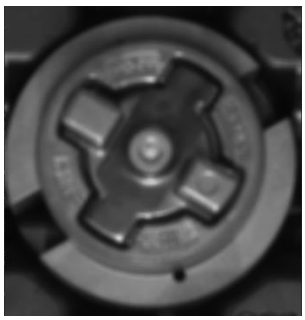

*Figure 24: blurred rotor image*

2. *Gradient calculation*: the concept behind this step relies on the fact that pixels belonging to one object are likely to have similar intensity values and gradient magnitude among these is low. On the contrary, boundaries between objects often present sharp transition in pixel intensities and this results in a high gradient magnitudes. Given this, boundaries are the edges the algorithm aims at finding.
3. Once the image is smoothed, the derivatives of the image with respect to x, ($I_x$, which represents) ,and y, $I_y$, are computed by convolving the image I with Sobel kernels Kx and Ky. Then, the

magnitude and the slope of the gradient are calculated in the following way: $|G| = \sqrt{I_x^2 + I_y^2}$ and $\theta(xy) = arctan\left(\frac{I_y}{I_x}\right)$. These give information about how fast or gradually the image changes in intensity in its point and, as a consequence, the probability that those part of the image represent an edge.

*Figure 26: Gradient with respect to x*

*Figure 27: Gradient with respect to y*

*Figure 25: Combined gradients*

4. *Non maximum suppression*: the image magnitude results in thick edges, while the most ideal case is to have thin ones. In fact, the gradient intensity level is between 0 and 255 and it is not uniform. According to this a non maximum suppression is performed: the algorithm goes through all the points on the gradient intensity matrix and finds the pixels with the maximum value in the edge direction. Then, it keeps only the pixels with the same intensity lying in the same edge by setting to zero the other. In this way, the edges on the final result have the same intensity (likely the one of white pixels)

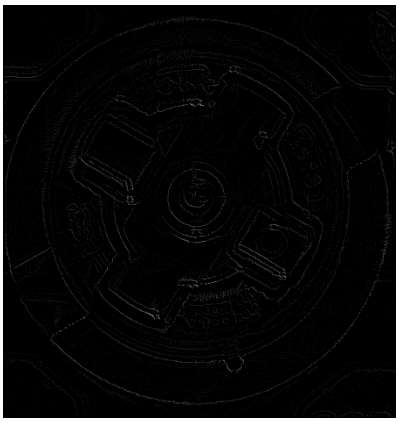

*Figure 28: Image processed by non maximum suppression*

5. *Double thresholding*: this step aims at identifying three classes of pixels: strong, that are pixels with high intensity, weak, ones whose intensity is not strong enough to be counted in the strong class but yet not small to be enclosed in the non-relevant category, and non-relevant, the ones that do not contribute to the edge definition. A double threshold is set in order to identify strong and non-relevant pixels, whereas those whose value are higher than the non relevant threshold but lower than the strong one, are marked as weak.

*Figure 29: Image after double thresholding*

6. *Hysteresis*: based on the results obtained in the previous step, Hysteresis aims at transforming weak pixels that has at least one strong pixel in their proximity, to a strong one.

*Figure 30: Image after hysteresis*

At the end of these steps, an image showing the edges of the input is obtained and is used to perform the Hough circles transform.

It is important to recall that any circle is described by the following parametric equations: $(x-a)^2+(y-b)^2=r^2$

Where *(a,b)* are the coordinates of the circle center whereas *r* indicates the radius.

The Hough Circles Transform maps the edged images from the cartesian space, whose axes are (x,y), in a parameter dimensional space, whose axes are respectively (a,b, r). Since it is possible in this case to exploit a previous knowledge about the length of the radius of the rotors in terms of pixels, the parameter space is compressed in two dimension (a,b) corresponding to the coordinates of the center of the circle in the original space. For each point (x,y) of the edge in the cartesian space, a circle of radius r centers in (x,y) is drawn in the

parameter space and every intersection between circles is tracked in an accumulator matrix. The accumulator matrix is a matrix A[a,b] where each element has a value, also called voting number, equal to the total number of intersections occurring at those (a,b) coordinates in the Hough parameter space. In other words, every time a new circle is drawn in the Hough space, the voting number of the points through which this is passing is increased by one. At the end, the local maxima of the matrix are taken into account and these correspond to the circles centers in the original space.

.

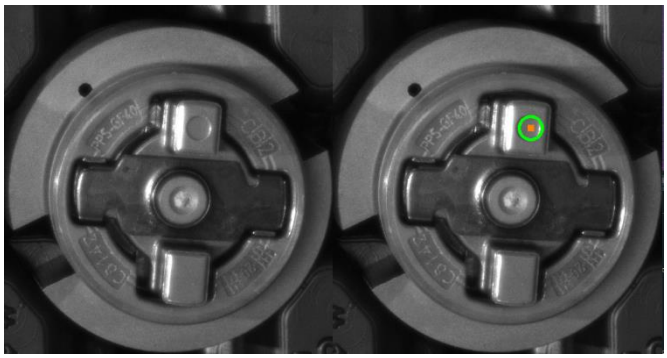

*Figure 31: Hough Circles transform algorithm*

In this implementation, the classification is performed by simply counting the tuple of coordinates (a,b,r) returned from the algorithm, each one corresponding to a identified circle.

For the purpose, the radius is set by exploiting a prior knowledge about the length of the rotors circle diameter in terms of pixels, which is divided by two. [19][20]

## 5.3. Results improvements

| Segmentation | | | | | Classification | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **LR** | **Epochs** | **Optim** | **Continuity loss weight** | **CBAM** | **LR** | **Epochs** | **Optimizer** | **Batch size** | **Accuracy CNN** | **Accuracy HCT** |
| 0.001 | 3000 | Sgd+Mom | 1 | True | 0.001 | 20 | Adam | 64 | 0.34 | 0.34 |
| 0.001 | 3000 | Sgd+Mom | 1 | False | 0.001 | 20 | Adam | 64 | 0.33 | 0.33 |
| 0.001 | 1000 | Adam | 50 | True | 0.001 | 10 | Adam | 128 | 0.86 | 0.92 |
| 0.001 | 1000 | Adam | 50 | False | 0.0001 | 10 | Sgd+Mom | 128 | 0.87 | 0.91 |
| 0.001 | 3000 | Adam | 100 | True | 0.0001 | 15 | Adam | 128 | 0.29 | 0.28 |
| 0.001 | 3000 | Adam | 100 | False | 0.0001 | 15 | Sgd+Mom | 32 | 0.28 | 0.28 |

Through the results it is possible to note how the Hough Circles Transform algorithm used for the classification task significantly improves the performances when the rotors are well segmented. Where they are not, it is not enough to improve only the performances of the classifier because if it takes as input a wrong segmented portion of image, no matter the classifier used. Given that a single misclassified image is enough to mark the entire KLT as scrap, the HCT is not able to reduce the percentage of scraps in these cases.

On the contrary CBAM does not carry any significant improvement on the overall accuracy: it is true that on average the accuracy of the model trained with some determined combinations of hyperparameters is 1% higher, but it is not possible to exclude that it

is due to the randomness, also given the fact that for some few others performances with CBAM are decreased by 1%.

This means that the feature extraction module were already able to extract meaningful information from the input images.

# 6. Hardware Integration on the assembly line

The flow described so far represent the software side of the Automatic Optical Inspection system. In this chapter are described the design, the logic and the hardware choices made in order to integrate it in the assembly line whose performance has to be analyzed.

It is important to get a step back and better contextualize the background the system is inserted in: at the end of the processes, the final assembled product is controlled through another AOI system supplied by an external company. This performs a visual check of the pumps by scanning different regions and controlling if each of them respects some pre-set parameters: if the product does not meet even only one requirement, it is marked as a scrap. It has been registered that a relatively high percentage of scraps are due to the assembly of the wrong rotor in the final product, and that is the reason why an it has been decided to design and build completely from scratch an AOI-based visual check to analyze rotors only. In order to do this, a standalone structure is built and installed next to the assembly line.

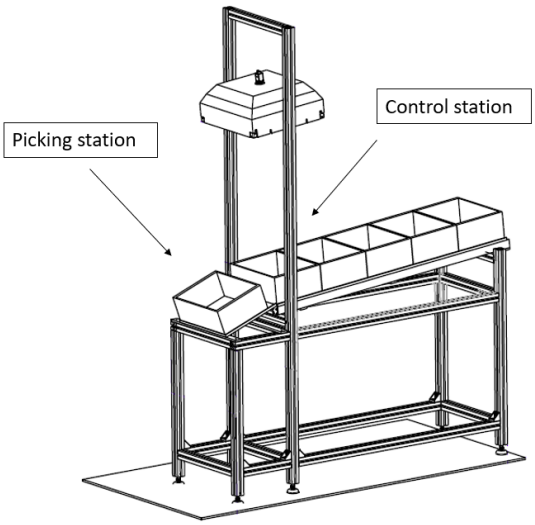


*Figure 32: standalone architecture installed on the assembly line*

A box of components slides automatically through a roller conveyor in the control station and it is blocked by a small step. A presence sensor detect the box and set its boolean flag variable from 0, which states for objects absence, to 1, which states for object presence. These are the only two values allowed. The sensor is connected to the dedicated pc, where the algorithm runs, through a digital input module which is able only to manage electrical signal. The digital input module, which is connected to the pc through a USB door, has several channels and it is configurable and manageable through python dedicated libraries. The algorithm reads the channel where the input signal is sent every 2 milliseconds and perform a simple difference with the previous variable in order to check if there is a new KLT to control. In particular, the current value is subtracted from the previous one: if the difference is non-negative (i.e. equal to 0 or 1) the box is not changed or there are no boxes in the control station; if the difference is negative (i.e.

equal to -1) it means that a new box is in the control station and a new picture has to be taken. To this end, the algorithm triggers the camera, which is connected to the pc by Ethernet cable, by sending it another electrical signal through an output module which is managed in the same way as the input one described above: the module is connected to the pc and by using some dedicated python libraries it is possible to make it send the electrical signal from one of its specified channel. The camera, which is set in advance by its software to the trigger mode, capture the image of the KLT from above and saves it in a pc folder. The algorithm reads from that folder every 10 seconds and everytime finds a new image it imports it and feeds the AI model which outputs a feedback. According to this one, an electric signal is sent again to a different channel of the output module: if it is positive, it is sent through a channel that makes a light bulb turn on with a green light, otherwise it is sent to another channel making the light bulb spread a red light. The image is finally moved from the folder to a dedicated database, stored with some descriptive metadata and the response of the algorithm. If the feedback is negative, the algorithm shows in a screen the wrong rotors inside the KLT by drawing a bounding box around them.

Each KLT is emptied in about an hour, so there is no cruciality in the responsive time of the algorithm. The delicate step is in between where the already checked KLT is pulled in the picking station and a new one slides in the control station. This operation takes really a few milliseconds and the algorithm must be responsive to immediately detect the boolean variable change of the sensor in order to make its

working logic consistent: that is the reason why the script listens to the sensor channel every 2 milliseconds, which is the most robust span of time to not loose any KLT switch.

# 7. Conclusions

The goal of the thesis was to design and built an efficient Automatic Optical Inspection from scratch in order to check if some components meet some predefined requirements, and mark them as good or scrap. Market offers different AOI solutions, but it has been decided to develop it internally in order to have the total control over every side composing the final system. The study started from the algorithm, went trough the architecture and hardware configuration until an analysis of the logic to allow the system to detect the presence of a new box of component in the right position and therefore to capture the frame. Colleagues of other department took care of building the standalone structure and to center all wiring in an electrical panel. The idea of using input and output modules to manage electrical signal in order to make the different hardware components to communicate has been developed completely by us as well as the algorithms composing the software part. The segmentation achieves good results even if completely unsupervised and, combined with a template matching algorithm is able to extract the single components centered in the crops in the majority of the case. As far as the classification is concerned, better results are obtained by using the algorithm customized on specific features of the components, even though standard convolutional neural network provided good results as well.

The entire solution has just been deployed in production for the testing phase. When a KLT is marked as scrap, the operators remove it and leave it under the quality department control. The aim, for the

future, is to find a logic to make the operator able simply to remove the wrong rotors displayed bounded in the screen and launch again the algorithm for the second check.

Finally a mention to Convolutional Neural Network structure, which reveals once again to be structures able to perform different tasks with interesting results thanks to their descriptiveness in extracting information from images.

# 8. Bibliography

[1] M.-J.-J. Wang and C.-L. Huang, ‘‘Evaluating the eye fatigue problem in wafer inspection,’’ IEEE Trans. Semicond. Manuf

[2] F. Timm and E. Barth, ‘‘Novelty detection for the inspection of lightemitting diodes,’’

Expert Syst. Appl., vol. 39, no. 3, pp. 3413–3422, 2012

[3] Y.-J. Cha, W. Choi, and O. Büyüköztürk, ‘‘Deep learning-based crack damage detection using convolutional neural networks,’’ Comput.-Aided Civil Infrastruct. Eng., vol. 32, no. 5, pp. 361–378, May 2017.

[4]A Gentle Introduction to the Rectified Linear Unit (ReLU)

by Jason Brownlee on January 9, 2019 in Deep Learning Performance

[5] A 2021 guide to Semantic Segmentation by Anil Chandra Naidu Matcha

[6] Unsupervised Learning of Image Segmentation Based on Differentiable Feature Clustering.

Wonjik Kim , Member, IEEE, Asako Kanezaki , Member, IEEE, and Masayuki Tanaka, Member, IEEE

[7] Deep graph cut network for weakly-supervised semantic segmentation.

Jiapei FENG, Xinggang WANG & Wenyu LIU

[8] Universal weakly supervised segmentation by pixel-to-segment contrastive learning.

Tsung-Wei Ke Jyh-Jing Hwang Stella X. Yu

[9] Weakly Supervised Semantic Segmentation Using Superpixel Pooling Network.

Suha Kwak,1,2 Seunghoon Hong,2 Bohyung Han2

[10] Fabric appearance testing

panelX.BinjieHuJ.

[11] Fast Normalized Cross-Correlation

Jae-Chern Yoo & Tae Hee Han

[12] Grayscale Template-Matching Invariant to Rotation, Scale, Translation, Brightness and Contrast

Hae Yong Kim Sidnei Alves de Araújo

[13] Edge-Based Template Matching and Tracking for Perspectively Distorted Planar Objects

Andreas Hofhauser, Carsten Steger, Nassir Navab

[14] AlexNet CNN Networks – Deep Learning Engineer Italia

[15] Understanding AlexNet: A Detailed Walkthrough

Azel Daniel

[16] An Attention Module for Convolutional Neural Networks

Zhu Baozhou1 , Peter Hofstee12, Jinho Lee3 , Zaid Al-Ars

[17] CBAM: Convolutional Block Attention Module

Sanghyun Woo , Jongchan Park , Joon-Young Lee , In So Kweon

[18] Real time circle detection by simplified Hough transform on smartphones

Viktor J. Schneider

[19] Canny Edge Detection Step by Step in Python — Computer Vision

Sofiane Sahir

[20] Circle Hough Transform - AI Shack