UNIVERSITÄT
LEIPZIG

IOM
Leibniz-Institut für
Oberflächenmodifizierung e.V.

Implementation of Artificial Intelligence for Defect Detection

Assessment of capabilities of supervised techniques
for small data sets and objects

Natasha Alexandra Hrycan Robachuk

Supervisors:
Prof. Dr. André Anders
Dr. Stefan Zahn

A thesis presented for the degree of
Bachelor of Sciences in Physics

Faculty of Physics and Earth System Sciences
Universität Leipzig
Germany
2024

Abstract

The quality of gas permeation coating films is affected by the presence of defects on the surface, which can be observed with scanning electron microscopy. The defects can be quantified and classified using machine learning approaches, such as object detection models. This work evaluates the limits of two object detection approaches in a dataset of 44 images containing typical defects of gas permeation barrier coatings. The size of the dataset allowed one person to label all images within a week.

The first approach was to train object detection models with labels from our dataset on different architectures (EfficientDet, Faster R-CNN and Roboflow Object Detection 3.0), proving that these models have a low precision for the detection of small objects (6,5%, 2,4% and 12%, respectively).

The second approach combined the architecture YOLOv8 with the library SAHI, which uses a slicing method to improve the detection of small objects. This architecture was expected to adapt for the detection of defects from our dataset, as no labels were provided for training. By varying the parameters of the slicing method, we obtained a precision of 11% and 10% for small objects.

These techniques showed that improvement is still needed for a feasible application in the analysis of coating films. Based on the results, it is recommended to carry out training on the custom dataset, include appropriate pre-processing filters to highlight small defects and implement slicing methods to improve the detection of small objects.

Declaration

I hereby declare that I am the sole author of this thesis, and that all external resources and materials are respectively marked down. I authorize for this work to be checked with the respective software for anti-plagiarism. I declare that this work has never before been presented to another examination board, nor has it been published.

Signed: ________________________________
Natasha Alexandra Hrycan Robachuk
Author

Acknowledgements

This thesis is dedicated to the memory of my dad and my grandmother, who blessed me with their wise words and support since I was a child.

I would like to thank my supervisors, Prof. Dr. André Anders and Dr. Stefan Zahn, for opening the doors of IOM so I could participate in this work. This thesis would not have been possible without their guidance and contributions.

Contents

Chapter 1

Introduction

Computer vision is a field of computer science and artificial intelligence that allows computers to retrieve information from images. This technology can be applied in material sciences and engineering for several purposes, such as the detection of defects in surfaces. Computer vision uses neural networks to build object detection architectures, with which it is possible to localize and segment several objects within an image, as well as to classify the detections. These capabilities become a useful tool for analyzing large amounts of data collected through experiments. Most architectures for object detection are trained with large common object datasets, which do not include objects seen in experiments. This means that the object detection models will find it difficult to detect objects from experiments, as these objects were not used while training the model. In material sciences, some objects might appear solely in very specific tasks and for that reason, task specific training might be necessary. This is possible by providing labels for training to the architecture. Labeling instances is both a tedious and a complicated task: it requires a lot of time and precision to draw bounding boxes or masks, and it can be challenging to properly label images with a high density of objects. Therefore, this work intends to compare object detection techniques for its application in the field of material sciences, considering time and computational resources limitations.

In order to analyze the attainability of these techniques, images of gas barrier coatings obtained with a scanning electron microscope (SEM) were selected for examination. The dataset examined in this work contained 44 images and the objects were labeled within a week. These images contained defects of several types: from pinholes and small particles, to trapped bubbles and scratches. These defects affect the leakage of gases and thus, the protecting properties of the coating. The high resolution of the SEM is ideal for identifying small defects, specially pinholes, that affect the gas permeation barrier. A sufficient number of images must be generated to have a proper idea of how the film surface looks. This leads to the consideration of using machine learning for object detection to count and classify defects in protection barriers.

This thesis explores the possibilities of convolutional neural networks for detecting defects in coating films. The main goal is to compare different algorithms and architectures for object detection, as well as to identify their strong and weak points. The next chapter introduces the theoretical background to the reader, where key concepts are described. Chapter 3 presents the employed techniques, obtained results and the comparative analysis between object detection models. The last chapter presents the conclusion, an outlook on how to improve the application of these algorithms and suggestions for future work. The appendix includes additional figures for the interested reader.

Chapter 2

Literature Review

2.1 Ceramic gas permeation barrier coatings

As mentioned by Hanika et al. in [1], everyday objects, such as chocolates and chips, need coating films in their packaging to protect themselves from changing their properties after exposure with air and humidity. Just as with chocolate and chips, applications in the field of renewable energies also require protection against leakage of gas products or contamination of their production with fluids from the outside. In this context, barrier coatings are essential for flexible solar cells and hydrogen gas storage systems, as it reduces the leakage of hydrogen gas. Therefore, the production of gas barrier coatings in the field of energy applications is one of the main focuses at IOM ([2]). In order to improve the permeability of these films it is necessary to analyze the results of the production techniques, for which it is beneficial to determine the amount and types of defects present in the films.

Polymer films are gaining recognition over traditional materials, like glass and metals, for long term protection ([1]). High-barrier polymer materials offer a better protection capacity when compared to conventional polymer materials. Nonetheless, it should also be taken into account that most of these films need specific conditions to be created, which can be translated into higher manufacturing costs.

Permeation is the penetration of a fluid (gas or liquid) through protective layers. In the case of homogeneous films, the permeation P can be defined by knowing the solubility S and diffusion coefficient D, due to the following relation: $P = D \cdot S$. Considering Fick's first law for the flux density, the permeability Q (also referred as transmission rate) and the permeation coefficient P are related as follows:

$$Q = \frac{P}{d_f} = \frac{J}{A \cdot \Delta p} \tag{2.1}$$

where d_f is the thickness of the film, A is the area of the film affected by the permeation of the fluid, J is the stationary flux though the film and Δp is the pressure drop. In the work of Hanika et al. ([1]) it is discussed the influence of the defect area and distance between defects on the permeability Q: small defects with high frequency have a larger negative impact, in contrast to bigger defects with lower frequency, demonstrating the importance of taking into account small defects. The flux per defect is lower when defects are located closely, meaning that the proximity of a defect reduces the concentration gradient of neighbouring defects. Thicker films reduce the permeability Q and reach a limiting value or critical thickness d_{crit}, which can give us an effective film thickness when considering the costs of manufacturing processes. The conditions for determining the critical thickness and the size of defects at which interactions between them start to occur are defined in the work of Hanika et al. ([1]).

In agreement with the statements mentioned above, da Silva et al. remark the effect of defects on the permeation of a coating film ([3]). It was shown the direct correlation of thickness of SiO_2 films and the density of defects in the coatings. Even though thicker coating films can significantly reduce the oxygen and water vapor transmission rates, the presence of defects can lead to an important decrease in the permeability of the surface. One of the defects that has the biggest contribution to fluid leakage are pinholes. Pinholes offer an easy passage for any fluid to escape, and when located in proximity to other pinholes, this can affect the permeability in an equivalent manner to a pinhole with a bigger radius. This is the reason why it is important to count not only the amount and sizes of the defects, but also the density of these defects on the surface, specially for the small pinholes in close proximity. There exists also a temperature dependence for some fluids and their transmission rates, which is subject to chemical interactions. The work of da Silva et al. ([3]) also presents a permeation decrease factor distribution for independent holes with a normal distribution, which is the ratio between the transmission rate per unit time of the studied surface to the transmission rate per unit time in the absence of the coating films

At the Leibniz Institute of Surface Engineering (IOM) flexible transparent oxide thin films have been developed [2]. Applications include flexible solar cells, where it is esential for the protective films to be transparent so that the amout of light perceived by the solar cell remains unaffected. Metal oxide thin films do not interact with oxygen, hydrogen or water, for example, common compounds that can affect the objects being protected. It seems reasonable to relate the permeation quality with the presence of defects, such as pinholes or pores. These defects work as a tunnel for gases or liquids to permeate through, and as mentioned earlier, it is important not to underestimate the effect of many small defects, as they build up the transmission rate fast. These thin layers are developed by UV photoconversion, allowing them to be about 100 nm thick. During the application of the protective layer on top of the flexible polymer surface, other types of defects may arise, such as oxygen bubbles, scratches and particles laying on top, underneath or included in the protective layer. These defects are presumed to have an effect on disrupting the permeability of the thin protective film and thus, they should be taken into account for further investigation. It is important to mention that the production of this thin oxide protective film is performed at low temperature and ambient pressure. These characteristics allow to work with substrates sensitive to heat and to potentially reduce costs compared to vacuum processes. The images used for analysis correspond to thin transparent oxide films (single layer SiO_2 films, about 100 nm thick) scanned with an electronic microscope in the nanometer scale, see Figure 2.1 for two selected example images.

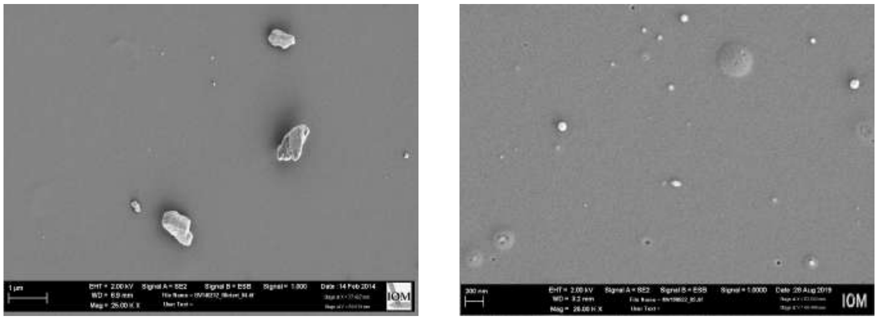


Figure 2.1: Images of the SiO_2 single layer coating film obtained by means of electron microscopy. The left image shows small particles laying on top of the film. The right image shows more particles, oxygen bubbles trapped during the process of applying the coating and pinholes.

2.2 Neural networks for object detection

Neural networks are a group of neurons linked together, which analyze inputs to give us an output, just like the human brain. In the case of image analysis with neural networks, the color code of each pixel will be the input features for the neurons, and several neurons can form a layer within the neural network. Each of these neurons would be "activated" in a range of [0,1], which corresponds to the normalized value in gray-scale of the color of that pixel, 1 being white and 0 being black. Depending on the architecture, the first layer of a neural network could be a pooling filter, a convolutional layer or a layer of fully connected neurons activated according to the pixels of the image. In order to understand the architecture, we focus first on explaining a fully connected neural network for image detection and classification, where the last layer is fully connected and provides the label of the object (i.e. pinhole, scratch, bubble, etc.). A neural network is considered fully connected if all neurons of every layer are connected to every corresponding neuron of the subsequent layer. A fully connected neural network (FCNN) works well only for the classification of small images, it is able to recognize the detected object as long as it is always located in that position ([4], [5], [6]).

In the case we have n inputs x, after passing through filters (if necessary) they will compose the first layer of the neural network, V_0. At the end of the neural network the output layer V_{f+1} provides the outputs y, here f is the number of hidden layers between the input and output layers. In the case $f = 1$ we are talking about an artificial neural network, and for $f \geq 2$ this would be a deep neural network. The neurons of every layer after the input one are activated by the so called activation function. The sigmoid function is commonly used as an activation function for a neuron in the network, which depends on the product of the activated values and the correspondent weight (one can think about the weight as making certain feature more or less important for the determining what object is it that we are detecting). Besides the sigmoid function, Rectified Linear Units (ReLU) and hyperbolic tangents can also be used as activation functions. Another important element is the bias of the function, which is a constant added after the weight has been applied, and this bias acts just as its original meaning, it will shift the outcome towards one direction or another and influence which will be the final label. For a fully connected simple neural network with an activation function $\phi(z)$ and 2 hidden layers, the value of the neurons in the neural network is described as follows:

$$V_{1j} = \phi_1 \left(w_0^{(1)} + \sum_{i=1}^{n} x_i w_i^{(1)} \right) \quad \text{for} \quad j = 1, ..., M_1 \tag{2.2}$$

$$V_{2k} = \phi_2 \left(w_0^{(2)} + \sum_{j=1}^{M_1} V_{1j} w_i^{(2)} \right) \quad \text{for} \quad k = 1, ..., M_2 \tag{2.3}$$

$$y_l = \phi_3 \left(w_0^{(3)} + \sum_{k=1}^{M_2} V_{2k} w_i^{(3)} \right) \quad \text{for} \quad l = 1, ..., O \tag{2.4}$$

where w_0 is the bias for each layer, w is the weight that modifies the input for the next layer, M is the number of neurons of the hidden layers and O is the number of outputs ([7], [8]). For a simple artificial neural network with 4 inputs, 2 outputs and 2 hidden layers, the neuron building block is shown in Figure 2.2. This is an example of a deep neural network, were the hidden layers analyze low, medium and high level features of the object intended to detect. The higher the level of the feature, the more exclusively this feature relates to the object we want to detect. Other additional hidden layers can deconstruct the object we want to detect into separate features or conduct dimensionality reduction ([5]). The problem can be summarized as follows: the task of the computer is to calculate all the weights and biases that will process the pixels of the image in the specified neural network and ultimately give us the label that we expected.

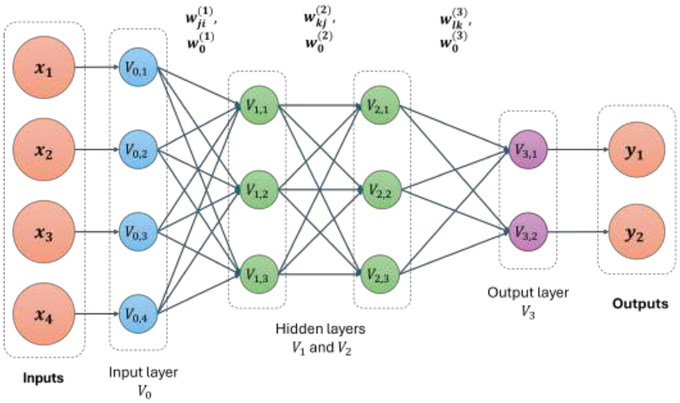


Figure 2.2: Scheme of a general artificial neural network with 4 inputs, 2 outputs and 2 hidden layers. Source [8].

Object detection starts with object localization, where we start with a bounding box where the labeled object is, so to let the neural network analyze the features within the bounding box. Object localization is done with at least 4 points, that can be defined with 4 lines in the x and y axis that intersect and define the bounding box. The bounding box is localized according to the characteristic features that the FCNN has learned about the object and its usual location within the image. Since the FCNN learned about the object by means of its characteristic features and location, the classification will depend on both parameters. This is a limitation as the detector will expect the object to always be located in the same place.

As it was mentioned, a FCNN has the ability to learn about the features of an object as long as it is located in the same position in the image, but that is usually not the case when analyzing defects. A convolutional neural network (CNN) adds convolution layers (not exclusively), where neurons are connected only with neurons within their receptive field. Therefore, neurons will not be fully connected with each other and the network will learn to recognize patterns, regardless of its location. CNNs might use additional filters to reduce the computational load of the network, such as pooling layers. A pooling layer is used to subsample the input image and leads to a reduction in the number of parameters and memory usage. Just like a convolution layer, the pooling layer only connects neurons that are within the receptive field (the size of the receptive field can be customized), but it does not add any weight to the neurons. If we have a receptive square field of size 2x2 pixels with different color values in each pixel and we want it to be downsampled to 1 pixel, a pooling layer can help us choose what pixel value should the downsample have. Depending on what we need to highlight, the downsample could be: the value of the maximum color value in the square (max pooling), the value of the minimum color value in the square (min pooling) or the average of all color values in the square (average pooling). Pooling layers lighten the computation load and allow the CNN to use smaller masks for convolution layers, so the model can analyze objects of smaller sizes in more detail. The architecture of a typical CNN with the described elements is illustrated in Figure 2.3.

There are several approaches for a CNN for localizing objects in the image, for example, the traditional way consists of a sliding window traveling across the image and sending each “snapshot” of the window to the CNN to analyze it (Figure 2.4). The CNN will retrieve an objectness score, which is the estimated probability of the image containing parts of the object intended to detect. This process requires the neural network to process every "snapshot" of the bounding box, which will also change its size to detect object of different dimensions.

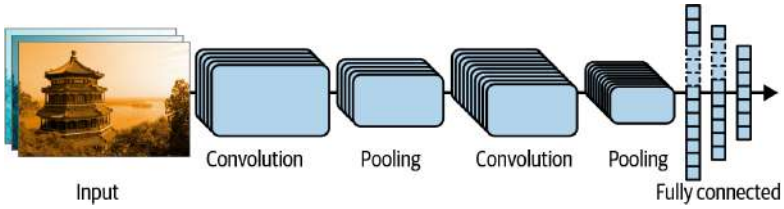


Figure 2.3: Architecture of a typical CNN. The first convolution layer has a large kernel to reduce the spatial dimension of the image, the pooling layer reduces the size of the input and the subsequent convolution layer has smaller kernels to focus and object detection and further classification with the fully connected network. Source [4].

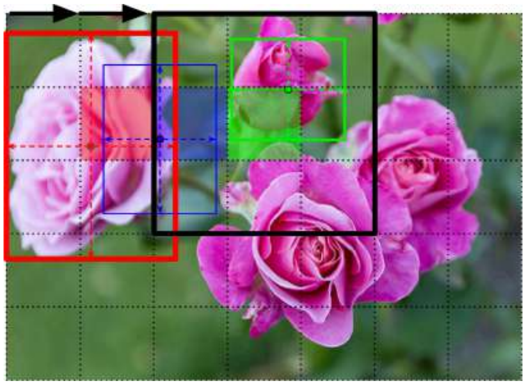


Figure 2.4: Sliding windows of a CNN for detecting multiple objects in an image. Source [4].

One of the proposed solutions for this computational heavy problem is known as You Only Look Once (YOLO), which was presented in 2016 by Joseph Redmon et al. ([9]). In this architecture, instead of sliding the window for classification all across the image several times, YOLO only considers objects whose bounding box center lies within a cell of each grid cell, and also gives an output of 2 bounding boxes, making it possible to detect objects that are really close to each other. The central trick is that the output layer is a convolution layer instead of a dense one. In contrast with the sliding windows from Figure 2.4, the CNN will now generate feature maps at which the model only looks once (Figure 2.5). This architecture has saved so much computational power to the extent that it is one of the most used architectures for real time object detection.

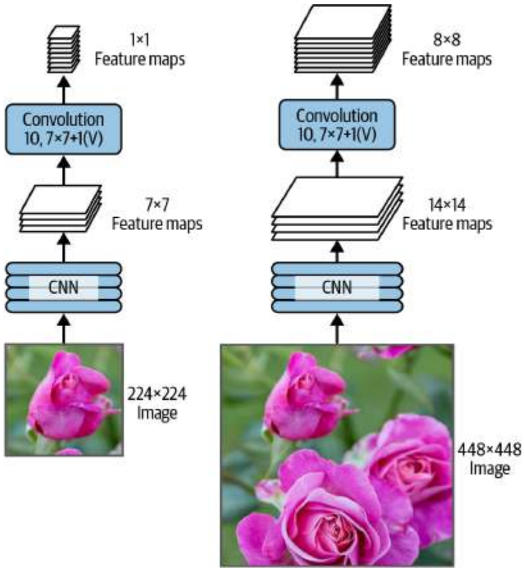


Figure 2.5: An illustrative example of how the same CNN processes a small image (left) and a big one (right). Source [4].

2.2.1 Overview of employed supervised algorithms

Tensorflow is an open source library of deep neural networks based on tensors, hence the name, for large-scale machine learning ([10], [4]). It provides a large collection of models for object detection, which were pre-trained on the COCO 2017 dataset ([11]). Even though this dataset contains common objects, these models can be trained on a custom dataset for object detection.

A basic step in order to train a custom dataset on a pre-trained architecture is to set labels for objects of interest so the model can be trained to identify them. Before training, the images will be divided in at least 2 groups: the training set, which will be the images and labels that will teach the model how the objects one wants to detect look like, and the test dataset, which will be used after training to judge how well the optimized model works on data that is not part of the training procedure. Commonly, the training set can be split and a small portion of it becomes part of the validation set. The validation data is not included during training and it is employed to identify overfitting or tune hyperparameters of the model.

The first architecture tested for the supervised training was EfficientDet-D3 with 896x896 input resolution, trained in COCO, inference speed per image of 95 ms and COCO mean average precision (mAP) of 45,4%. The family of detectors EfficientDet uses fewer parameters and significantly reduces the number of floating-point operations during inference, which translates in reduced computational complexity. This architecture uses a bi-directional feature pyramid network, which allows multi-scale feature detection, meaning that the architecture has a better ability to detect objects in various sizes. This model is able to perform well with a lighter load on computational resources compared to other architectures. One example is that the model D0 achieves similar accuracy as YOLOv3, but using 28 times less floating-points on inference, which is translated as the amount of calculations performed ([12]).

Another of the models used in this work is Faster R-CNN with Resnet-50 (v1) with 640x640 input resolution, also trained on COCO, initialized from Imagenet classification checkpoint, with an inference speed per image of 53 ms and COCO mAP of 29,3%. R-CNN stands for Region based Convolutional Neural Network and ResNet-50 is a deep convolutional neural network with 50 layers. This is a two-stage object detection architecture that uses a Region Proposal Network (RPN) to propose the areas where objects can be detected. This model was chosen due to its inference speed and to test if the two-stage object detection architecture brings any advantage in the detection of small objects ([13]).

When comparing both architectures, EfficientDet requires fewer parameters and less calculations when performing inference than Faster R-CNN. It is important to note that the input resolution on the employed architectures are not the same, so for a more detailed computational analysis it is necessary to provide equal starting parameters.

2.2.2 Onto the flexibility of pre-trained architectures for object detection

Pre-trained architectures were employed for object detection. This transfer learning technique allows to significantly reduce the computational time for training on new but similar objects. Nonetheless, our dataset contained too few labeled instances for some classes. Therefore, the capabilities of pre-trained models to localize new objects in an image were studied. Since the employed architectures were pre-trained on common object datasets, the found object will not contain the correct label, as this type of defects were not included during training. However, the bounding box of the prediction can be employed to extract the segmentation mask of the object and store it for classification with an unsupervised clustering algorithm, for example. If the characteristic features are correctly chosen, the clusters will contain similar defects and we can choose a random element in the cluster to label the group. This method could reduce significantly the amount of time employed for the detection and classification of defects, as we would only need to label the representatives of the clusters instead of all objects. Furthermore, underrepresented objects in the dataset might form an own cluster and thus, be identified overall easily. During this work, solely the capabilities of k-means clustering were studied with object size, average color of all pixels and 10% of the lightest and darkest pixels as input features. More advanced techniques based on CNN architectures (see [14] and [15]) were not studied but might be focus of future work.

Another problem of CNNs is that objects with a really small area (compared to the complete image) are difficult to detect, even for common objects. An example of this problem in real life, we have drones trying to detect and classify objects at ground level: even though the model has been trained in this kind of objects (person, car, tree, etc.) and the camera has a good resolution, the model will find it difficult to detect objects that are too far. This problem is also the case for the images used in this work: they are visible but most models could not detect the defects, just because they appeared to be too small. A solution for the "small object problem" in object detection was proposed by Akyon, Fatih Cagatay et al. in [16]. The approach is named Slicing Aided Hyper Inference (SAHI) and fine-tuning for small object detection. This software slices the image into smaller images, so that small objects would appear "larger" when compared to the complete image, which is now just a slice of the original image ([17]). The slicing method allows one to choose the size of the slices, smaller slices will be able to detect smaller objects, and also the overlap percentage, so that objects are not cut in half and, as a result, the model detects it as 2 objects or none.

SAHI offers choices for the detector model one prefers: YOLO, Hugging Face, MMDetection, Detectron2, Torchvision, DeepSparse and others. This work has implemented a single-stage detector, YOLOv8, with the library SAHI for object localization.

The object clustering was done using the k-means algorithm, which is an algorithm capable of clustering elements of a dataset in a few iterations after randomly initializing k centroids for k regions (the visualization of the regions with boundaries in a scatter plot is called a Voronoi diagram). These centroids are re-localized and the boundaries of the regions move as this happens. The aim of the centroid localization update is to reduce the Within Cluster Sum of Squares (WCSS), which corresponds to the distance of the points in the Voronoi region to the centroid. This way when the algorithm converges, one has obtained a Voronoi diagram where the points are localized in the region where they are the closest to, and the centroids are located at the most effective position to include points that share the most similar characteristics. The optimal number of cluster k is determined by the "elbow" when plotting the WCSS as a function of the number of clusters k. In this graph one should choose the number of clusters k before the function starts decreasing slowly. Additionally, the silhouette score was also employed for determining the amount of clusters. The silhouette score involves the mean distance to other instances in the same cluster and the mean nearest-cluster distance to define the silhouette coefficient. A silhouette score close to 0 tells us that the instance is well inside its own cluster. This last method for defining k is usually preferred as it gives a clear peak for the appropriate k, whereas the elbow method may not provide a clear distinction for the most effective k. In this work the scatter plot corresponds to the area of the defect against the average color of the defect in a normalized scale [0,1], in cases where the defects are larger one could choose other characteristics for classification, like length and width. It is important to note that k-means clustering does not work properly when clusters have varying sizes, densities or non-spherical shapes. If any of these points is present, it is necessary to either choose different input vectors for the scatter plot, or use another mathematical approach for clustering.

2.2.3 Performance metrics for supervised training

In order to analyze the performance of an object detection model, determining quantitatively how many times our model detected correctly or incorrectly an object can give us a better understanding to configure the hyperparameters of our model. Hyperparameters are parameters of the learning algorithm (for example, type of optimizer, learning rate and batch size) and tuning them allows to build a better machine learning system ([4]). For this aim, the following definitions are necessary to quantify the success on the classification task:

- True positive (TP): the bounding box encloses a correct detection of an object and labels it properly. As an example, the bounding box encloses a flower and labels it as a flower.
- False positive (FP): the bounding box encloses an incorrect detection or a non-existing object, but it labels it as an existing object or the classifier labels it incorrectly. As an example, an object detection model detects a person in the background, even though there is nothing there, or a classifier for birds labels the dog as a bird.
- False negative (FN): an object is not found by the detector. As an example, an object detection model that fails to find a flower in the image, even though there is one.

These definitions and the ones that will follow are extracted from the references [4], [18], [19]. Now that the basic concepts for representing quantitatively the performance of our model, we can introduce the most used metrics for it.

Precision and Recall

The precision P of a model is defined as the ability for it to detect correctly relevant objects. It is defined as the rate of correct predictions over the total number of detected objects:

$$P = \frac{TP}{TP + FP} \tag{2.5}$$

On the other hand, the recall R is the ability of the model to find objects in our image. It is given by the ratio between correctly detected instances, and the total amount of instances present in the image:

$$R = \frac{TP}{TP + FN} \tag{2.6}$$

One should take into account that a poor model will not have high precision and recall continuously, so if this is the case we will have to give up on precision to detect more instances in the image. With these two definitions we can construct a useful curve, known as the Precision x Recall curve. This tool gives us an idea of how the model is evolving. Ideally we would want the precision to increase as the recall increases.

Average Precision

Considering that the precision of our model can vary as the recall changes, the average precision AP can give us the averaged value of the precision along all recall values. This can be determined by knowing the maximum precision at various values of recall (0%, 10%, 20% and so on up to 100%) and then calculating the mean among those maximum values. In the case we have several classes or objects to detect and classify from the image, one can calculate the mean average precision mAP by obtaining the AP for each class and averaging it among all classes, giving us a better metric for our object classifier.

Some metrics may be classified by the size of detected objects, as a detector could have a low overall mAP as a result of difficulty for detecting only a specific size of objects. For example, a detector may be fine-tuned so that it has a high precision only when detecting small objects, but is completely obsolete for medium and large objects. For instance, a “small” object in object detection can be as small as 2 pixels and go up to 32^2 pixels of area, a “medium” size object has an area between 32^2 to 96^2 pixels, and finally a “large” object occupies an area between 96^2 to 100.000^2 pixels.

Intersection over Union

All previous metrics did not account how well positioned is the predicted bounding box. Our object may have detected, for example, a dog, but if the bounding box is only enclosing its face instead of the whole body of the dog, the trust on the model decreases. The model must be able to find the instance and properly bound it and all of its parts with the bounding box, otherwise the model may start deviating its idea of what an object looks like, deforming it and worsening its ability to understand what the object looks like.

The IoU is a metric that evaluates the accuracy of the detector for localization of objects. If we define the area of the ground truth bounding box as A_{gt} and the area of the predicted bounding box as A_p, the definition of IoU

is given by:

$$IoU = \frac{A_{gt} \cap A_p}{A_{gt} \cup A_p} \tag{2.7}$$

One can take as an example Figure 2.6, where we expect the detector to find a flower with some grass around. The predicted bounding box encloses most of the flower, but not all of it, and this misplacement can lead to bigger confusions if objects are located too close. The detector should be able to detect properly the boundaries of the object, so that in the process of training and validation, the detector does not shift its “understanding” of what a flower looks like. If the IoU is low, the detector will perceive the background or neighboring objects as part of what one actually wants to detect.

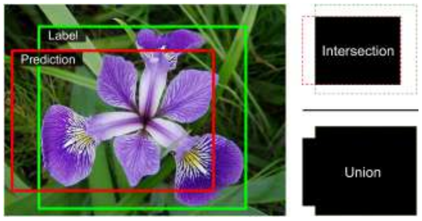


Figure 2.6: IoU metric explained graphically. Source [4].

Loss function

Another concept used when working with parameters of neural networks is the loss function, which is a mathematical tool that minimizes the error between the predicted and expected instances. This function is essential for optimizing the network and avoid the use of additional resources that could contribute to the the training in more meaningful processes. We say that the model performs well if the “loss” is low. Instead of training the model with several different hyperparameters, the loss function will fine-tune them to regularize the training, in order to avoid overfitting. Overfitting will produce a model that can predict instances if they are part of the training data, reducing the adaptability of the model for real world applications. The loss function will therefore provide the regularization needed to restrict an overfitting decision tree.

Depending on the task of the neural network, it will use different loss functions. EfficientDet uses a focal loss function, which is a modified version of the cross-entropy loss function. This is a loss function used for classification by predicting probability distributions, and it is also included in the loss function for Faster R-CNN ([20]). Faster R-CNN computes the regression loss function with the smooth L1 loss function, also known as the Huber loss. This function is a combination of the mean squared error and the mean absolute error, where the error is parametrized by a threshold value δ ([4], [21]).

Tensorflow Object Detection evaluation

Tensorflow has a pre-determined function for running the evaluation job along the checkpoints that appear while training the model. For evaluation on the models used with Tensorflow, we have used the COCO detection metrics. Some of the metrics that will be used are defined as follows:

- **Detection Boxes Precision/mAP:** gives the mean average precision over all classes. If not specified the IoU threshold, it means this is the mAP averaged over IoU thresholds from 0,5 to 0,95 in steps of 0,05. It could also specify a threshold of 0,5 or 0,75 IoU, or mAP for small, medium and large objects.

- **Detection Boxes Recall/AR:** this represents the average recall with a certain amount of detections (1, 10 or 100) and can also be classified by size (small, medium or large).

- **RPN Localization loss:** gives the loss of the bounding box regressor for the Region Proposal Network (RPN).

- **RPN Objectness loss:** this is the loss of the classifier that determines if the bounding box corresponds to an object or it is simply background.

- **Box Classifier Classification loss:** determines the loss for the classification of the object into one of the classes (pinhole, particle, scratch, etc.).

- **Box Classifier Localization loss:** this is the loss of the bounding box regressor.

Chapter 3

Analysis and Discussion

This thesis deals with a multi-class object detection problem, which translates to the application of algorithms that perform detection and classification tasks. We use two approaches that use pre-trained architectures for object detection as a base. Both approaches use the same dataset with gray-scale images that show defects on a surface. The dataset consists of 44 images with 9 classes of defects: particle, bubble, pinhole, trapped particle, hole, scratch, small scratch and deformation. An important remark is that in several instances, the images contain defects with nuances that can make it more difficult for the detector to perform its task properly. These details affect also the performance of software that do not use neural networks for object detection, such as the ImageJ Particle Analysis plug-in ([22], [23]). Some examples of these details are overlaying defects, shadows and the difference in colors among defects, which make it more challenging to apply a color threshold filter for image segmentation.

A detailed overview of the steps performed for training the models is illustrated in the Appendix (1). The following sections in this chapter explain how the data was manipulated, the configuration details and outcomes of the different utilized architectures and the analysis of the results.

3.1 Data collection and pre-processing

The images used for testing the object detection algorithms correspond to the images of transparent ceramic coating films obtained by means of scanning electron microscopy (see Figure 2.1 for example images). The images were collected at IOM during the last 10 years and contain typical defects observed in gas permeation coating films. For processing these figures, it was first needed to cut out the lower section of the images, which contained information about the filename and metadata, as this information box could produce false detections when testing the object detection algorithm. Once all images were resized to the same size, the task of labeling each of the instances started. The software LabelImg ([24]) was used for this, the bounding boxes with the corresponding labels were later exported in the form of PASCAL VOC annotations. These annotations contain information about the location of the bounding box in the image and the label that was assigned to it. This format allows the later generation of TensorFlow records for training the model. Annotations are therefore not affected by filters, segmentation or pooling, as the bounding boxes would be located in the same position after such transformations. One of the main issues corresponded to the enhancement of pinholes, as they are small in size compared to other defects, and many times can be confused with the background or noise when testing a detector. A "minimum" pooling filter was used to highlight the pinholes and create a more intense contrast with the background. Additionally, the apparent size of the pin was slightly increased. Also known as min pooling, it creates a down-sample by selecting the minimum pixel value of the region, changing the "color" of the batch of pixels.

3.2 Comparative analysis of the algorithms

3.2.1 Supervised Machine Learning

EfficientDet D3

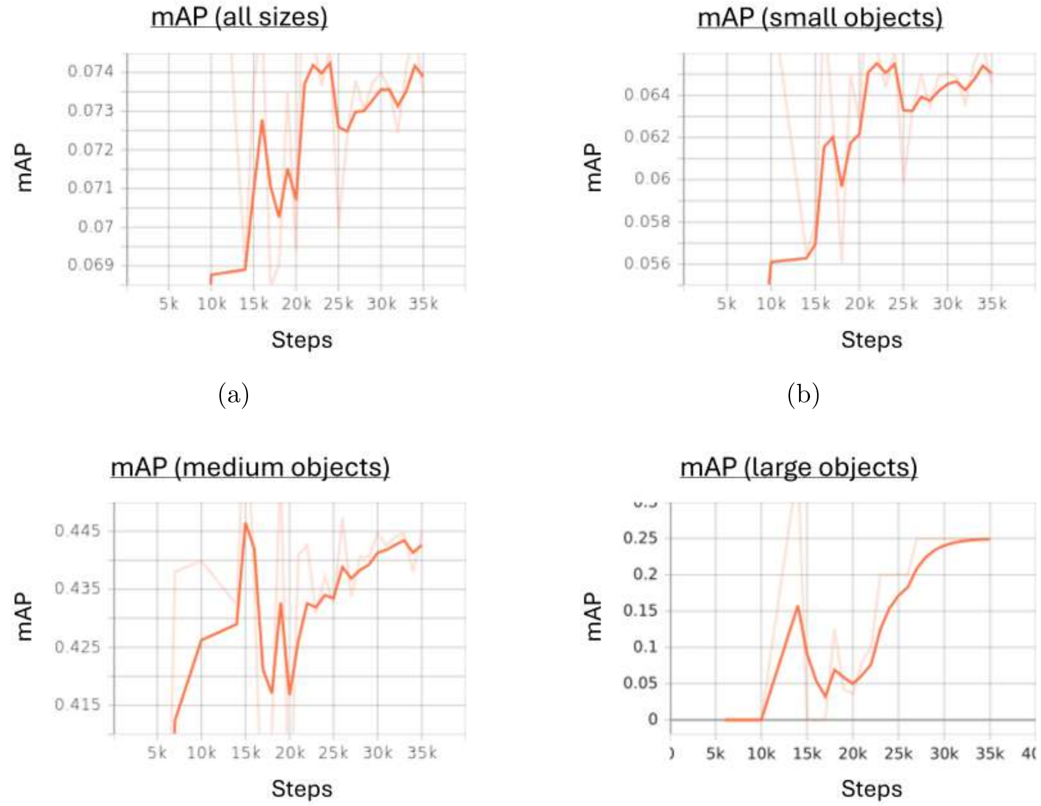
EfficientDet D3 with input resolution 896x896 required around 6 hours for training and testing using 4 nodes with 4 GeForce RTX 2080Ti each. The model run a batch size of 8 images over 25.000 steps. The batch size was varied to compare the influence of this parameter on training. Unfortunately, studies with batch sizes larger than 8 were limited by the available computational resources during the thesis. The images were randomly separated into a training set (approximately 63% of the original dataset) and a test set (approximately 37% of the original dataset). The configuration of random scale crop and pad to square was used for data augmentation.

After 25.000 steps of training, the model achieved an overall mAP of 7,4%

(Figure 3.1a), which was heavily influenced by the low mAP for small objects (Figure 3.1b) that only reached a mAP of 6,5%. The model had a better performance with medium and large size objects, reaching approximately 44,3% and 25% of mAP, respectively (Figures 3.1c and 3.1d). These metrics give us the understanding that this model performs better when the task involves the detection of medium and large size defects. Please note, the mAP of 44,3% for medium sized objects is comparable to the mAP of 45,4% of the same architecture optimized and applied to the COCO 2017 dataset ([25]). Therefore, increasing the dataset for training might result only in minor improvements for medium sized objects.

Similarly to the case of the general mAP, the average recall over 100 detections was shifted due to the low average recall for small defects. The overall average recall reached 13,5% (Figure 3.2a) and the recall for small objects may have been affected by over-training, as it decreased its value over training and ended up labeling correctly only 10% of the detected instances (Figure 3.2b). The medium and large size average recall curves present a steady improvement in the latter stages of training, reaching satisfactory results of 57% for medium sized defects (Figure 3.2c) and 50% for large size defects (Figure 3.2d). It might be worth to investigate if a larger dataset can improve the mAP and recall due to the simple shape of the objects. Nonetheless, the problem with the small defects is part of the "small body problem", which will be presented in more detail in the following subsection.

Figure 3.3 contains the provided ground truth images with labels and the detection that the model performed after training on the custom dataset. In accordance with the high values of precision and recall for medium size objects, it is possible to appreciate that the model finds it easier to detect, for example, medium size particles. Smaller particles and pinholes are not even detected by the model, meaning that there should be further work to ensure that the proportion of the defects is closer to the minimum threshold for bounding boxes provided by this model.



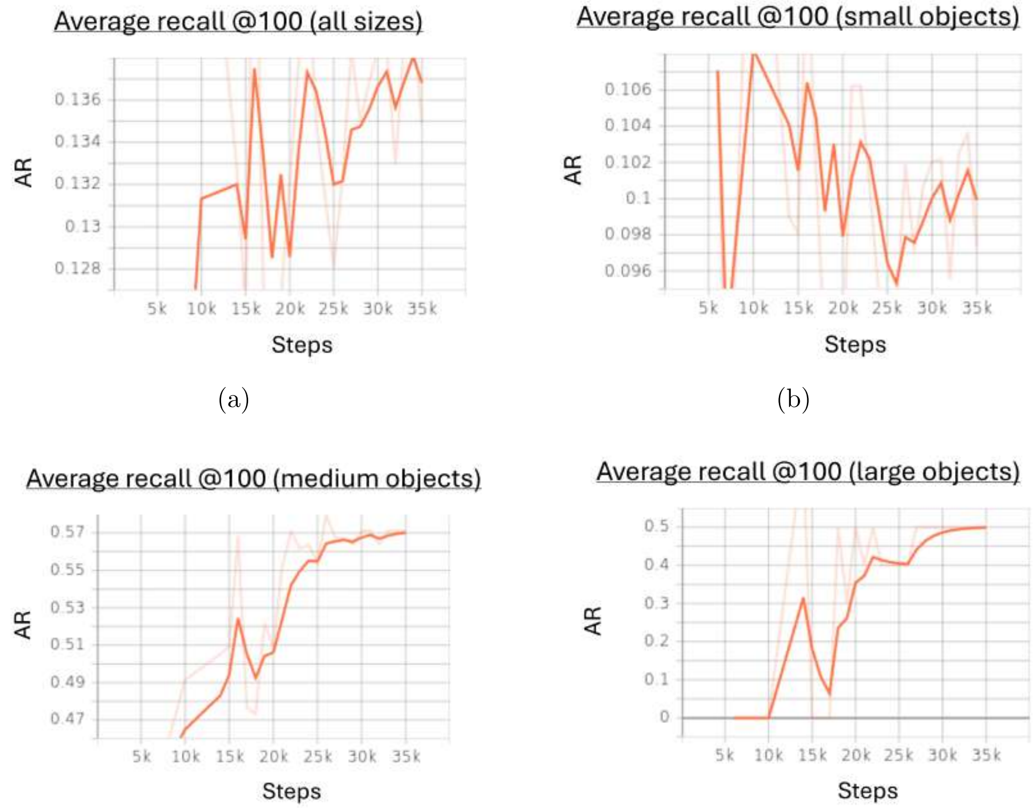
(a)

(b)

(c)

(d)

Figure 3.1: Mean Average Precision (mAP) for the detector EfficientDet D3 for different types of defects: all sizes (a), small defects (b), medium size defects (c) and large defects (d). The bold orange curve corresponds to the smooth version of the faded orange curve in the back.



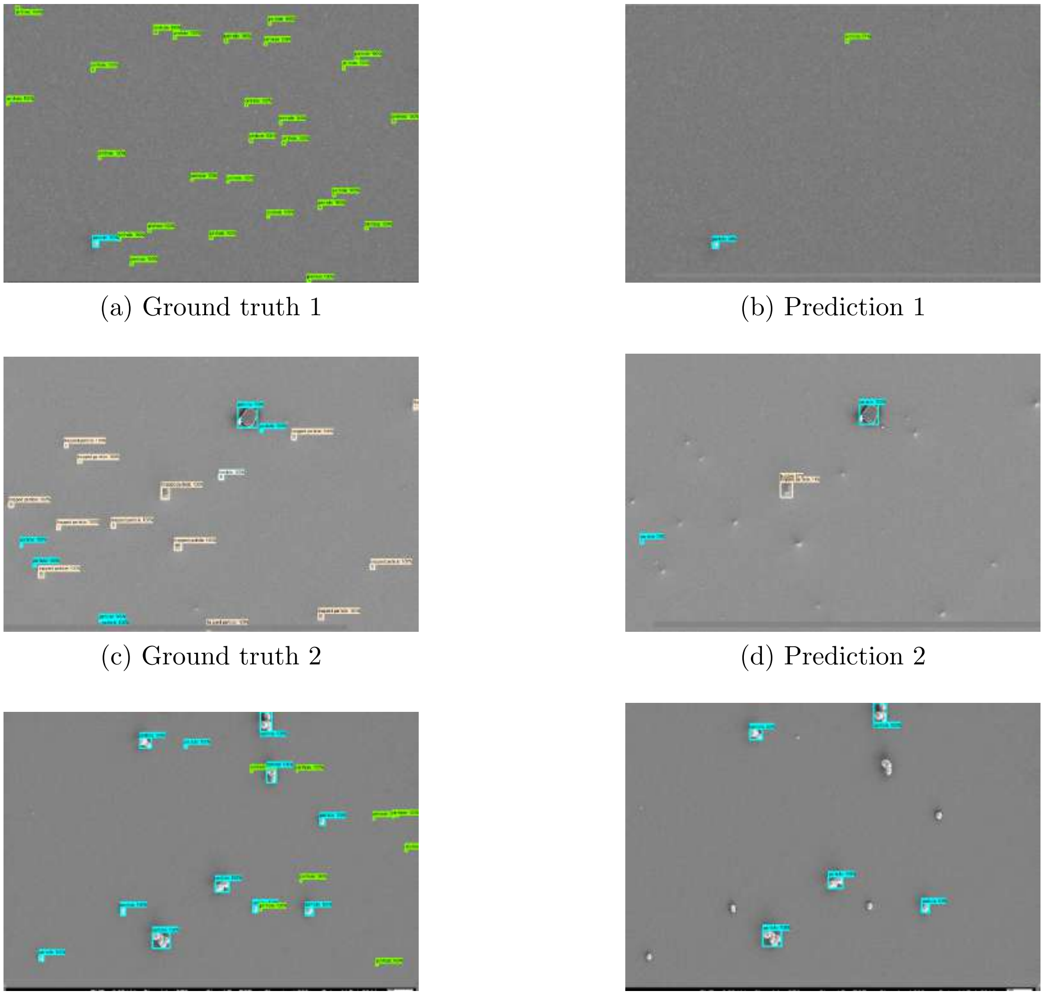
(a)

(b)

(c)

(d)

Figure 3.2: Average recall over 100 detections for the detector EfficientDet D3 for different types of defects: all sizes (a), small defects (b), medium size defects (c) and large defects (d). The bold orange curve corresponds to the smooth version of the faded orange curve in the back.



(a) Ground truth 1

(b) Prediction 1

(c) Ground truth 2

(d) Prediction 2

(e) Ground truth 3

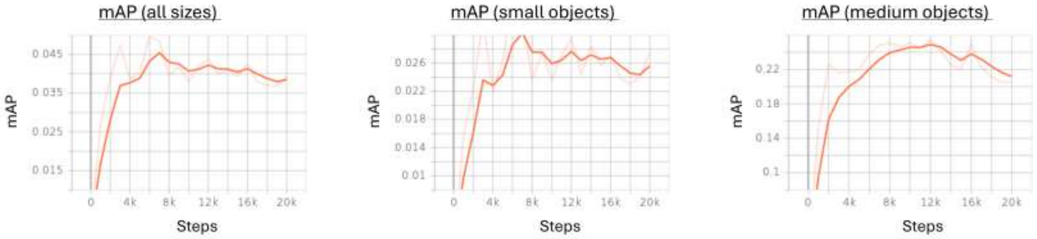
(f) Prediction 3

Figure 3.3: Side by side comparison of the predictions of the ground truth labels provided to the model EfficientDet D3 and the obtained predictions.

Faster R-CNN

Faster R-CNN with Resnet-50 (v1) and 640x640 input resolution was trained in 20.000 steps with a batch size of 8 images, and took approximately 13 hours to complete training and testing using 4 nodes with 4 GeForce RTX 2080Ti each. The images were randomly separated into a training set (approximately 63% of the original dataset) and a test set (approximately 37% of the original dataset). For data augmentation it was chosen the random horizontal flip configuration.

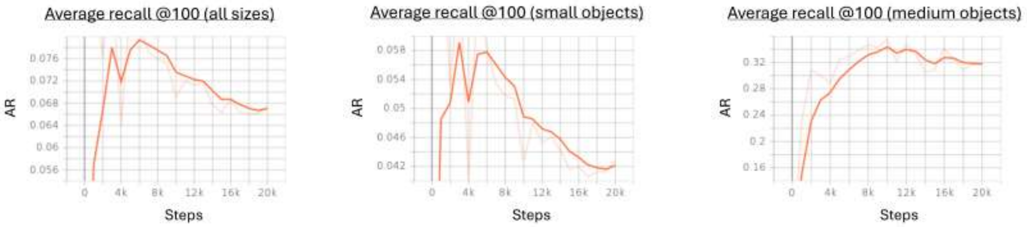
This architecture reached an overall mAP of 4% (Figure 3.4a). Similarly as for the EfficientDet D3 model, this was strongly affected by the mAP for small objects. The dataset does not contain a sufficient amount of large labeled instances to properly train this architecture, resulting in an obsolete model to detect large objects in images. The low mAP for small objects (around 2,4%, see Figure 3.4b) is presumed to be a result of the higher difficulty of reach the minimum required IoU, which was set to 70%. Misplacing a bounding box for a small object only by a couple of pixels will have a greater effect on the IoU when compared to a misplacement for a medium or large object. On the other hand, the mAP for medium-sized objects is around 20% (Figure 3.4c), which is closer to the mAP of this architecture for the COCO 2017 dataset (mAP = 29,3%).



(a) (b) (c)

Figure 3.4: Mean Average Precision (mAP) for the detector Faster R-CNN Resnet50 V1 for different types of defects: all sizes (a), small defects (b) and medium size defects (c). The bold orange curve corresponds to the smooth version of the faded orange curve in the back.

Just as in the case of the mAP, the recall was also affected by the size of the detected instances. All the average recall figures in 3.5 correspond to the average recall over 100 detected instances. The overall average recall almost reached a peak of 8% at 6.000 steps (Figure 3.5a), but then the average was heavily affected by the low recall obtained for small instances (Figure 3.5b), which started worsening the ability to properly label detected instances and finalizing with only a 4,2% of recall. As in the case of the mAP, the average recall for medium sized object performed better and by the end of training 32% of the medium sized objects were labeled correctly (Figure 3.5c). Overall, the Faster R-CNN model cannot compete with the investigated EfficientDet D3 model for the given small dataset. Some evaluation images from this architecture are available in the Appendix (2) if the reader is interested in the results of this architecture.



(a) (b) (c)

Figure 3.5: Average recall over 100 detections for the detector Faster R-CNN Resnet50 V1 for different types of defects: all sizes (a), small defects (b) and medium size defects (c) . The bold orange curve corresponds to the smooth version of the faded orange curve in the back.

Roboflow 3.0 Object Detection (Fast)

The software Roboflow [26] is a tool for training an object detection model. It is suitable for people with beginner programming skills and optimizes hyperparameters by itself. However, it has limitations for more advanced programmers as the configuration of the pipeline is restricted and no information about the use of computational resources is provided. The employed model was Roboflow 3.0 Object Detection (Fast) (see [27]) with checkpoint COCOn.

For pre-processing, the images were auto-oriented and resized to a size of 640x640. The original dataset consists of 44 images, which were divided randomly into three groups: training set (30 images, 68% of the dataset), validation set (8 images, 18% of the dataset) and test set (6 images, 14% of the dataset). As the training set contains only 30 images, it was chosen for data augmentation a 90° rotation in the directions: clockwise, anticlockwise and upside down. After randomly selected augmentation of the training data, the set was extended to 87 images, which is a considerable larger amount of information for the architecture to be trained on.

Example images of detections obtained after training the model on the test set are depicted in Figure 3.6. The Table 3.1 expresses the average precision by class, which helps understand which are the types of defects that are more difficult for this model to detect. Note that because the images were randomly shuffled into train, validation and test set, some types of defects as scratches and holes (and bubbles in the case of the test set) are not present in the Table, as the images in the validation and test sets did not contain any images with this type of defects. The model, as expected, can detect medium and large size objects with higher precision (like bubbles and particles). Smaller defects (for example, pinholes) and more complex defects (like holes on top of bubbles, or defects in close proximity) are more difficult to detect. The software suggested a larger dataset and pointed out that some classes were underrepresented. This mostly affects defects like scratches, holes and deformations, as the pinholes are among the top labeled defects in the dataset.

The model effectively decreased all types of loss as the model was evolving, as one can see in Figure 3.8, which is what one would expect from the model. The mAP@50 (maP over IoU threshold of 50%) is higher than the mAP@50-95 (maP over IoU thresholds of 50% to 95%) all along the training of the model, which suggests that our model can detect defects, but most

	Validation set (%)	Test set (%)
All	55	37
bubble	80	-*
particle	67	75
pinhole	29	12
trapped particle	47	24

Table 3.1: Average precision by class for the model Roboflow 3.0 Object Detection (Fast). *There were no examples of defects labeled as "bubble" in the Test set, as the images were randomly split between the sets.

of the times the bounding boxes are misplaced, leading to lower IoU values for the detected instances. This model with almost 200 epochs for training took around 1 hour to complete the training, validation and testing. This model had a mAP of 55,5%, precision of 53,1% and recall of 55,9%, which is the best performance among all supervised models tested. One of the downsides of this model is that the platform does not allow access for obtaining the details of the neural network for replication, and the training does not allow customization. It is possible to export the model to test it with other images, but this platform gives little possibilities for developers to retrieve or customize the layers of the network.

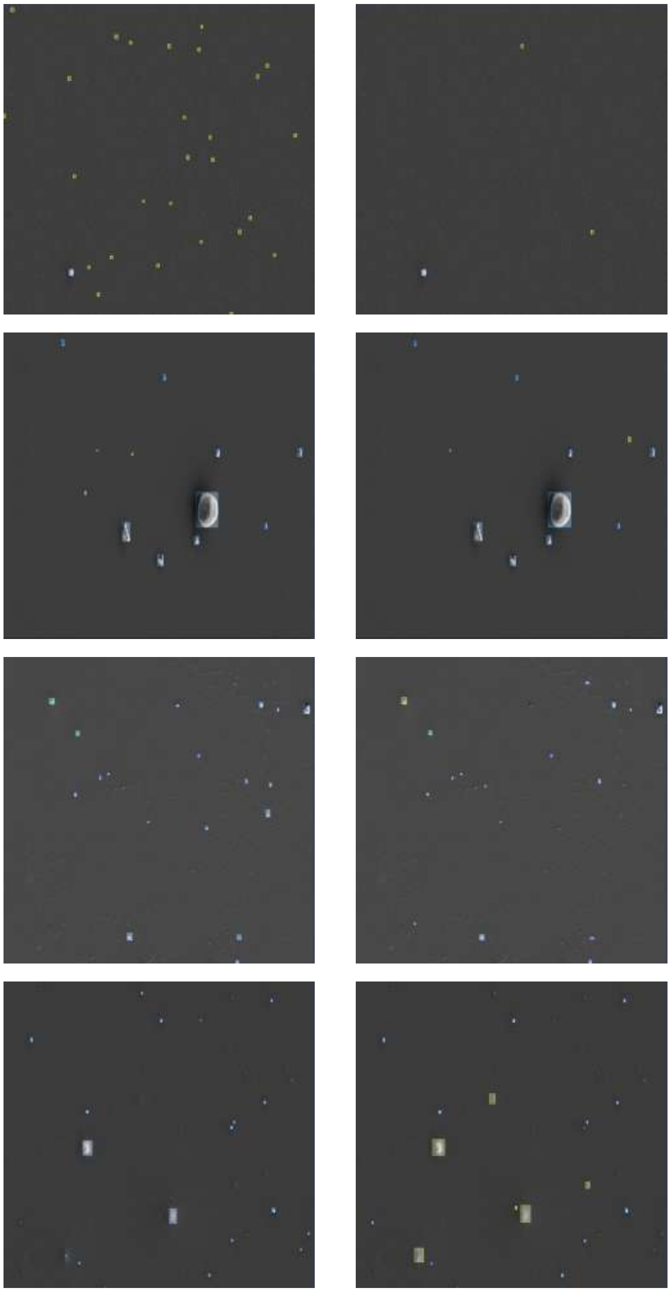


Figure 3.6: Testing of the supervised ML algorithm using Roboflow 3.0 Object Detection (Fast): labeled images provided for training (left column) and detection results on the Test dataset after training (right column).

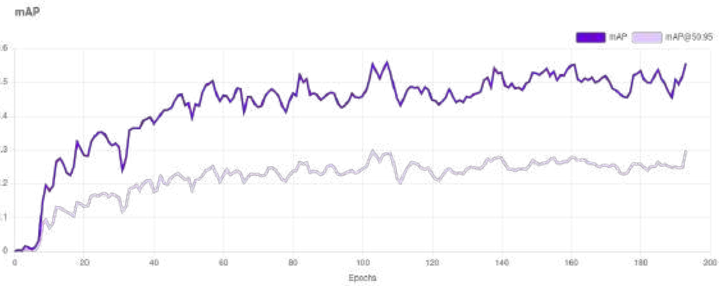


Figure 3.7: Mean Average Precision evolution through epochs of the model Roboflow 3.0 Object Detection (Fast).

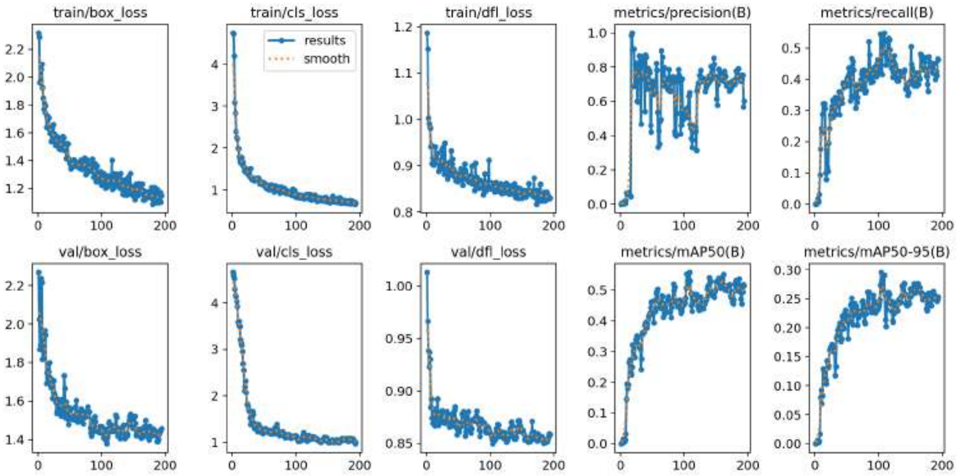


Figure 3.8: Metrics evolution through epochs of the model Roboflow 3.0 Object Detection (Fast).

Summary

The supervised algorithms presented in this section work well for medium and large objects. In our case, there are few objects of those sizes in the dataset, which leads to underrepresented classes when performing training. This issue can be solved by extending the dataset with focus on medium sized objects and providing more labeled instances for the model to learn about them. Nonetheless, small defects, such as pinholes, affect significantly the quality of a permeation barrier ([1], [3]). Even though small object classes were abundantly represented in the training dataset, the models did not provide sufficiently good precision and accuracy when detecting them. For small objects, we obtained the following mAP for small objects: EfficientDet with 6,5%, Faster R-CNN with 2,4% and Roboflow Object Detection 3.0 with 12%. Based on these results, we understand that the main problem with this architectures is the difficulty for detecting small objects. This is a recurrent issue in object detection, known as the “small object problem”, which will be studied in more detail with a different approach.

3.2.2 Towards reliable detection of small objects

As described in section 2.2.2, this work used the model YOLOv8 with the library SAHI to improve the ability of the model to detect smaller objects. SAHI was the most precise and computationally efficient resource for solving the “small object problem”, and YOLOv8 was chosen due to its speed and sufficiently good precision for detecting objects, as well as its compatibility with SAHI. This section contains the analysis of two models that use both SAHI with YOLOv8, but have different configurations for the slicing of images: one with bigger slices (SAHI+YOLOv8 Big) and another with smaller ones (SAHI+YOLOv8 Small), to show how it affects the detection model (Table 3.2 contains the details for reproduction of the training). The time shown in the table corresponds to testing the model with no GPU support on the set of 44 images. To give the reader an idea, when trying to set the slice height and width to 50 pixels, this generates 540 slices on our images, and increases the runtime to approximately 7 hours and 30 minutes. This may be a better configuration for detecting the smallest types of defects, but will require GPU support for analyzing the images. All images prior to training underwent a minimum pooling filter, which aimed to enhance the appearance of dark defects, specially pinholes.

Model	Slice height/width	Number of slices	Overlap height/ width ratio	Runtime
SAHI+YOLOv8 Big Slices	180 pixels	54	0,4	40 min
SAHI+YOLOv8 Small Slices	90 pixels	140	0,2	1 hour 45 min

Table 3.2: Configuration for training with SAHI+YOLOv8.

The classification task was not performed by YOLOv8, as this model employs parameters optimized on a common object dataset. The defects of our dataset are not part of the common object dataset, and since no custom training was performed on the model, the predictions provided wrong labels. Therefore, the model SAHI+YOLOv8 was used only for object detection purposes. After detection of the instances, the segmented area of the detected instance and the segmentation mask were extracted from the result dataset. The segmentation mask defined the limits of the detected instance inside the bounding box, with this information it was possible to retrieve the value of the normalized color values of the pixels inside the mask and average them. The area and average color of the defect were used for clustering the detected instances into groups of defects using the k-means algorithm. The amount of clusters k was chosen after the analysis of the WCSS as a function of the possible number of clusters k.

The model was only evaluated on its ability to detect objects. This means, only bounding boxes were considered in the analysis, ignoring the prediction labels. Because of this, many metrics are not available as they depend on the classification ability of the model (i.e. mAP and recall). It was possible to determine the precision of the models by comparing solely the predicted bounding boxes with the ones that were expected. To understand better the results, the precision by sizes was calculated, so we can determine which sizes of defects are easier to be localized by these models.

SAHI YOLOv8 - Big Slices

Taking into account that the generalized capabilities of the architecture YOLOv8 was solely used for the purpose of detection and localization of the defects, the k-means clustering algorithm was used to try to classify the defects considering their size and their average normalized color. These two features were considered the most relevant considering there were two main types of small defects: pinholes and particles. Pinholes are smaller than particles, therefore we expected a slight separation between clusters in the small area region. On top of this, pinholes are dark defects, while particles usually appear much lighter in color. This also suggests that along the normalized color axis there could be separated clusters. Considering the explained features, it was decided to perform the k-means clustering on a scatter plot, where the x axis corresponds to the area of the defect, and the y axis relates the average normalized value of the color of the defect.

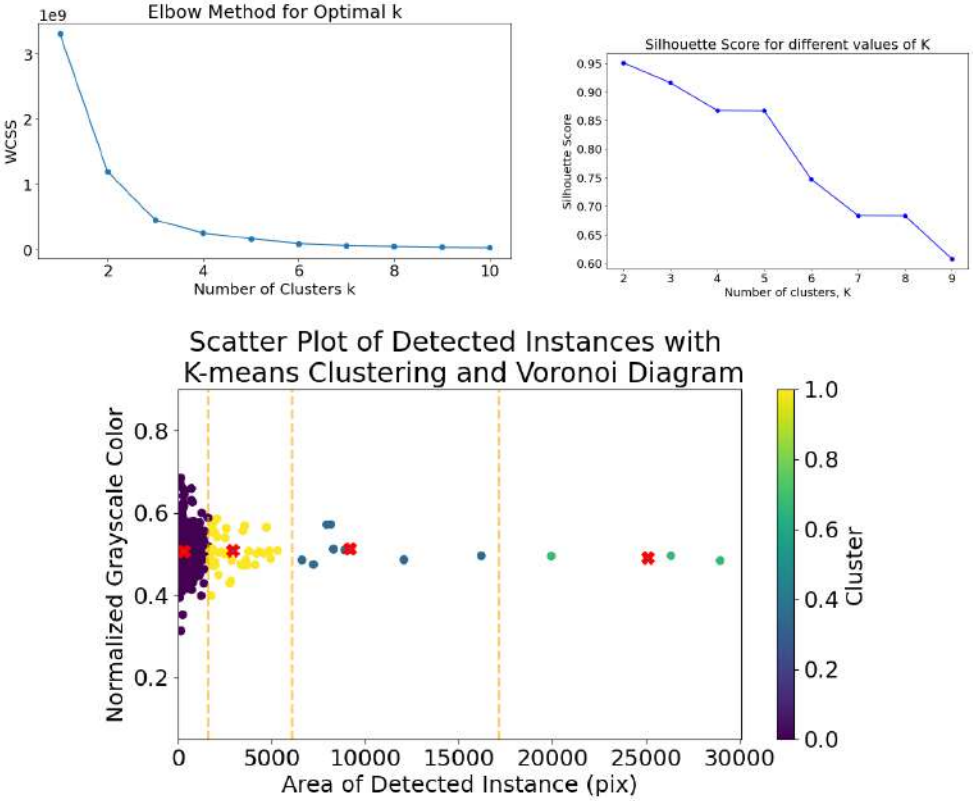


Figure 3.9: WCSS as a function of the suggested amount of clusters k for the SAHI+YOLOv8 Big model (upper left image), the silhouette score (upper right image) and the generated Voronoi diagram with k-means clustering with $k = 4$, parametrized by the area and average normalized color of the defects (lower image).

In order to decide the amount of clusters k, the WCSS as a function of the suggested amount of clusters k was presented as a plot in Figure 3.9. This graph shows how the sum of square distances between the points in the cluster and centroids vary depending on how many clusters are available. If the WCSS is too high (i.e. low k values), some points are too far from the centroid and most likely do not belong to that Voronoi area. The "elbow" is the point in the curve where the amount of clusters k is optimal, increasing k will not give any significant difference for the WCSS and most likely result in subgroups with no significance. In this case, the elbow is located at $k = 4$, past this value the gain in WCSS is not noticeable. The silhouette score was also determined but it did not have the characteristic peak, but rather started at a high value for the silhouette score. This led to errors while testing for $k = 2$, as the randomly initialized centroids were located in areas of low density and we needed a minimum of 4 points for drawing a Voronoi region.

The Voronoi diagram in Figure 3.9 classified the defects according to the size, as the only region where there is a significant difference according to the color is the small area region. For our data, k-means clustering does not seem to be a reasonable way to classify defects. The clusters did not have a spherical-like shape and the there was a large difference in the densities along the scatter plot. These problems could also explain the lack of peak in the silhouette score. It is necessary to consider a more complex input vector for classification since there was no visual formation of clusters.

SAHI YOLOv8 - Small Slices

Following the same logic explained for the Big model, now with smaller slices, the WCSS vs. number of clusters k graph was used for determining the appropriate amount of clusters (see Appendix 3). In Figure 3.10 one can see that just as the previous case, the silhouette score started at the highest silhouette score value rather than having a peak in the curve. It was suggested that the source of the problem is the same as the one explained for the model Big Slices. When comparing to the Voronoi diagram of the Big model in Figure 3.9 is the limit in the x axis, in the model it goes up to 30.000, whereas for the Small model the limit is 10.000, so even though both models had 4 Voronoi regions, these regions do not enclose the same defects. The Voronoi diagram of the Small model classifies in more detail the defects that we saw in the first 3 Voronoi regions of the Big model. This aligns with the comments made on the analysis for the Big model, if we reduce the slicing size, our scatter plot will focus on smaller defects (which are believed to have a bigger effect on the permeability) and therefore one would be able to distinguish small particles from pinholes, for example, as the distribution by color is wider only in the region of smaller areas. This classification can provide information about the defects that are believed to have a greater impact on the permeability and can serve as a solution for detecting small defects better than other models, with a lower amount of work as labeling is not necessary for this method.

Side by side comparison of the models

As mentioned earlier, the metrics for classification were not available because the model YOLOv8 is pre-trained on a common object dataset. If one would like to obtain the recall of the model one would need to find a model that is pre-trained on objects similar to the ones needed, or train a model on the custom dataset, which will end up being a supervised learning task, described in section 2.2.1. It is still possible to analyze qualitatively the performance of the model in different configurations and determine which defects were more precisely detected.

Some of the obtained predictions are depicted in Figure 3.11. The green bounding boxes correspond to the ground truth, i.e. the labels known to be true and given for the metric evaluation, while the red bounding boxes correspond to the predictions made by each model. As mentioned before, this approach used the adaptability of the model YOLOv8 to detect objects it has not seen before, so labels were not assigned.

One of the most recurrent problems while testing the algorithms was the over-count of detections, i.e. detecting multiple objects when there is only one. Images 3.11c and 3.11d are good examples to illustrate this issue, one can see that several instances overlap several prediction boxes where there is only one ground truth box. Regarding the ability of detecting small defects, results vary: images 3.11e and 3.11h show that the big model found more of the small defects present in the image, even though the smaller slices of the Small model should have been able to detect more of them, as they would appear “bigger” in the slice. On the other hand, images 3.11g and 3.11h show that the small model detected more of the really small instances, which correspond mostly to pinholes, one of the objects that have been the most difficult to detect in all models. Bigger objects like oxygen bubbles were not

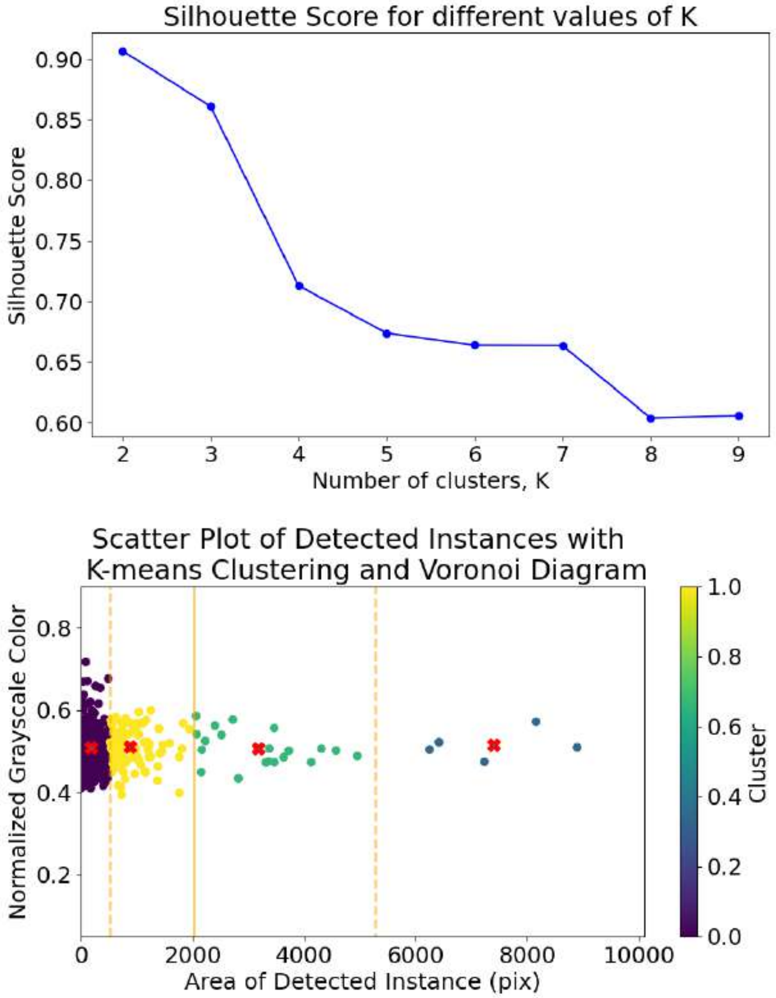
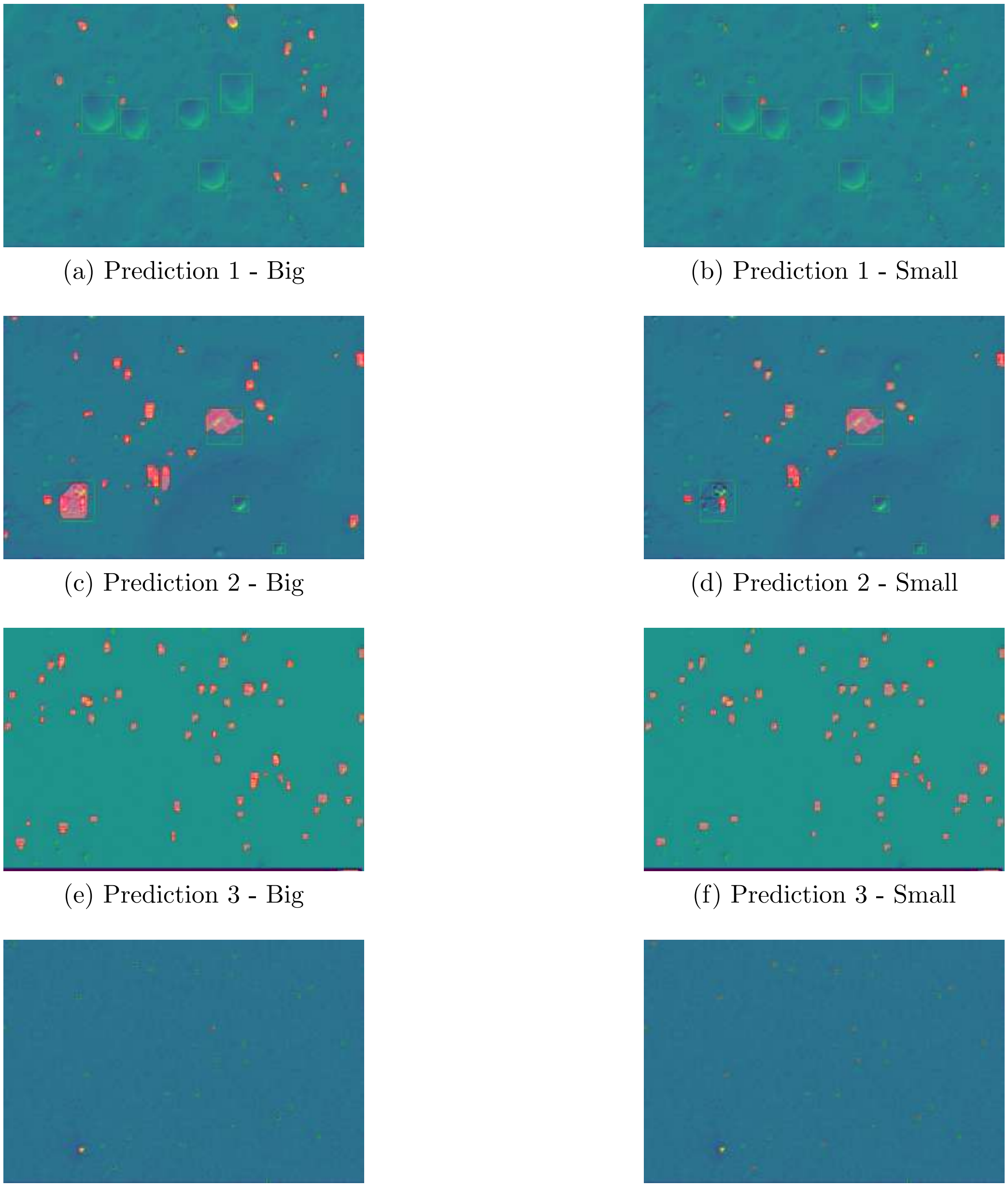


Figure 3.10: The silhouette score (upper image) and the generated Voronoi diagram with k-means clustering for $k = 4$, parametrized by the area and average normalized color of the defects (lower image). Just as in 3.9, no reasonable clustering is visible.

detected (images 3.11a and 3.11b), this is because of the size of our slices: if the defect is bigger or almost the same size as the slice, it will not be detected as the model will take it as a background object. The prediction batch in both cases drives to the same conclusion: this model cannot detect objects of several sizes with the same configuration. The most effective approach could be to analyze the presence of defects separately in groups that are defined by the size of the defect (big, medium, small and extra small, for example). One could use the model several times to detect defects of different sizes (i.e. configure the slice size and test the object detector on the same dataset) and subsequently compare the detections. This will avoid double-counting defects and will result in a more universal object detector, as it will be able to work with all sizes.



(a) Prediction 1 - Big

(b) Prediction 1 - Small

(c) Prediction 2 - Big

(d) Prediction 2 - Small

(e) Prediction 3 - Big

(f) Prediction 3 - Small

(g) Prediction 4 - Big

(h) Prediction 4 - Small

Figure 3.11: Side by side comparison of the predictions of the model SAHI+YOLOv8 Big (right column) and SAHI+YOLOv8 Small (left column).

Summary

The architectures used in this section were trained on a common object dataset. The accuracy of this approach might be improved if YOLOv8 is trained on a dataset with labeled defects from coating films. In order to improve the precision when detecting small objects, the approach was to use a slicing method that allows the detector to perceive small objects more easily. The models were not adapted with labeled instances to our dataset, so they provided labels from the common object dataset.

It was possible to extract information from the results and calculate the precision of localization of the bounding boxes for different sizes (see 3.3). This method provided a sufficiently high precision for small objects (10% for Small Slices and 11% for Big Slices) if one considers that the models were not re-trained to detect defects. For reference, the mAP for custom trained models were: 6,5% for EfficientDet D3, 2,4% for Faster R-CNN and 12% for pinholes with Roboflow. It is important to remember that the mAP is the average precision across classes, which means that this metric also takes into account labels. It is not correct to compare the precision of bounding boxes and the mAP straightforward, but it can give an idea of the potential improvement that could be made if custom trained models use also slicing aided inference for improving small object detection.

Model	Precision of bounding boxes		
	Small	Medium	Large
SAHI+YOLOv8 Big Slices	11%	15%	12%
SAHI+YOLOv8 Small Slices	10%	19%	-*

Table 3.3: Precision of localization for the bounding boxes when testing the model SAHI+YOLOv8 for different sizes of detected instances. *There was no data for large objects detected by the model SAHI+YOLOv8 Small Slices, as the slices were smaller than the defects they were expected to detect.

3.3 Strengths and weaknesses

When using machine learning for defect detection, it is necessary to identify which are the priorities for which this model can be more efficient or better than conventional image analysis software. Below are listed some of the points that may be taken into account when choosing between architectures. Another point to take into account is that the dataset contained few examples of certain types of defects, leading to underrepresented classes when training the models for this custom dataset. It is also important to remember that there are many other architectures available, and the ones used in this work were chosen because of time and resource constraints, while trying to maximize the results.

- **Less time and effort consuming:** The model YOLOv8 + SAHI is the one that required the least amount of work to setup. Since the parameters of the YOLOv8 network were optimized for the detection of common objects and not for surface defects, the capabilities of this approach might be significantly improved if the parameters of YOLOv8 are optimized on a large dataset of labeled defects. As a result, subsequent unsupervised clustering of objects might be obsolete. Varying the size of the slices will result in a detector focused on smaller or bigger defects. By combining detectors for different sizes of defects, it is possible to obtain a more universal object detector, able to detect defects regardless of the size.

- **Easiest to use:** The model Roboflow Object Detection 3.0 and its platform are suitable for beginner programmers and users that do not want to deal much with the specifics of neural networks. It also allows to export the trained model to test on more images and warns about underrepresented classes. Customization or analysis of the layers of the neural network is not possible, which can be a problem when trying to improve the architecture.

- **Customization available:** Models from the Tensorflow Object Detection Zoo are the models that can be customized best and were studied in this work. The architecture EfficientDet has less parameters, which can be enough for medium-experience programmers to start customizing layers of neural networks, and it is faster than the model used of

Faster R-CNN. Faster R-CNN has more options to explore on the design of the neural network and as it is a two-stage object detector, it requires more resources. For experienced programmers this architecture will allow to fine tune the hyperparameters and this can result in an architecture that outperforms the others presented in this work. Furthermore, the optimized model might be combined with the library SAHI to detect small objects better.

Just as some models performed well for object detection, other models were also tested and were not included in the analysis as they gave really low performance metrics. Some of them are listed below with a brief explanation on the complications that came through while testing the model.

- **ImageJ Particle Analysis:** This plugin from ImageJ ([23]) is able to create masks for particles in an image. This method does not use neural networks for detecting images, but only the masks provided from a binary image. It counts and provides a summary with the location and area of the detected instances. First one needs to apply color thresholding, Gaussian or mean pooling filters, watershed separation. Please note, there were still problems with shadows and defects of "opposite" colors (light particles and dark pinholes cannot be detected at the same time, for example). This plugin finds the borders of objects from a binary image, meaning that the processed image should be already segmented into 2 colors only. One can also fix borders manually, but this ends up being more time consuming than just counting particles by hand. This software cannot classify defects or identify defects laying on top of each other, and the manipulation of images individually can consume a lot of time. This approach does not require any significant computational resources and thus, can be run on any work station.

- **Stardist:** This is an object detection model based on Tensorflow trained on a database of images from experiments (mostly cells and nuclei), focused on multiclass detection of small objects (see [28], [29], [30]). As the model is trained on a database focused on experiments, the expected results were more promising than the ones obtained from common object datasets. Unfortunately, there is one important restriction for this detector, and that is that all defects must be star-convex objects. Many of the defects in the database have a star-convex shape, but there were too many particles that did not fit in with the requirements of the model to be detected. This model may be able to outperform the models presented in this thesis if the shape of most of the defects has a more consistent, spherical-like shape. Another advantage of this model is that it can be trained on a custom dataset, but it uses a different annotation system with masks instead of bounding boxes.

- **Increasing resolution of the input layer:** The EfficientDet and Faster R-CNN architectures have versions with a higher input resolution and precision, but at a really high computational costs. Heavier versions of these models were tested but the available computational resources were not enough, as the nodes kept running out of memory (the maximum tested was 104 Gb of memory and 4 GPUs per node). These restrictions require access to a high performance computing cluster with systems optimized for such tasks.

3.4 Discussion

The dataset consisted of 44 images. The careful labeling of all images took several days due to the large number of defects in each image. The dataset of this study represents a size where labeling can be done by a single coworker during his lab routine. The following 5 architectures were evaluated: EfficientDet D3, Faster R-CNN Resnet 50 V1, Roboflow Object Detection 3.0 and YOLOv8 + SAHI for small and big inference slices. The supervised training was carried out by providing images with labeled instances and later evaluating them to compare the quality of the detector. For the SAHI+YOLOv8 method, a pre-trained architecture was used to locate defects on the images, which were sliced to improve the identification of small defects. This model was pre-trained on a common object dataset, which means that the labels provided by the model were unrelated to the real labels of the defects. Due to this detail, it was not possible to calculate the mAP across classes, as the ground truth and predicted labels are completely different, and in the same argument the average recall was not determined as it needs the accuracy of the label prediction. A summary of metrics for the supervised models for all sizes of defects is provided in Table 3.4.

Even though the mAP and average recall of the first two models are really small, this is mainly influenced by the bad detection of the numerous small

Model	mAP@0.5	Average Recall
EfficientDet D3	14,4%	13,5%
Faster R-CNN Resnet50 V1	11%	8%
Roboflow Object Detection 3.0	37%	55,9%

Table 3.4: Results of mean average precision for predictions with an IoU threshold of 50% (mAP@50) and average recall obtained on supervised training models after training on a custom dataset with images and labels. The metrics correspond to all sizes of defects present in the image.

objects. Once isolating the metrics for medium size defects, the EfficientDet D3 architecture has a mAP of 44,3% and average recall of 57%, while the Faster R-CNN Resnet50 V1 model reached a mAP of 20% and an average recall of 32%. It is important to determine if one needs higher precision or ability to identify objects correctly (recall) when choosing which architecture works better for an experiment. There is a trade-off between precision and recall in object detection, so in most cases it is difficult to reach a high percentage for both of them.

Even with sufficient labeled images, pre-trained object detectors find it difficult to identify defects, specially small ones (i.e. pinholes and particles). The SAHI+YOLOv8 model presented a potential improvement on the detection of the small defects. Unfortunately, we cannot assure that the detector can identify defects of different sizes at the same time due to the slicing of the image. One would need different configurations to get a better precision for each size. The obtained data also suggests that a combination of the library SAHI with a model trained on a case specific dataset could potentially give better results and provide correct labels.

Even though the models were trained on a custom dataset with labels, some types of defects were underrepresented and this decreased the quality of the detector when running the test set. Underrepresented categories is a normal situation in real life applications and could probably only be solved with a larger dataset or by synthetically generated data. For a detector to classify properly instances, we need a good balance among all classes so the detector is well trained on differentiating between them.

The unsupervised clustering based on k-means still needs further improvement. We used a scatter plot with the average normalized color of the segmented area against the area of the mask. The segmentation masks of the predictions are "polluted" with too many pixels from the background, which ends up shifting the actual value of the average normalized color of the defect. As an example for the small region, pinholes are dark (values towards 0) and particles are lighter (values towards 1). However, the value in both cases ends up shifted towards the middle due to background contribution. Thus, splitting in clusters is not reasonable. This conclusion is supported by the silhouette score showing no peak and already starting at its maximum value. The next step on this issue would be using normalized histograms of pixel color distribution of the defect as input vector of the average color.

We have two suggestions for further improvement: the first one consists on selecting more characteristic features for the clustering plot, for example in the work of Thirunavukkarasu et al. ([31]) we can see a better example of how k-means clustering can be used for the classification of defects. The second suggestion is that more advanced techniques are needed for clustering, both when processing the segmentation masks and when choosing the type of clustering algorithm. Unsupervised learning techniques with deep clustering and contrasting cluster assignment have shown to be relevant tools for visual features by Caron et al. in [14] and [15] respectively. Another work that presented issues for defect classification with the k-means clustering algorithm is the work of Altmann et al. ([32]), where we agree that supervised classification with labels performs slightly better than unsupervised clustering based on local features, and that a bigger set of data is needed to achieve better classification results.

K-means clustering for image segmentation in combination with an adequate sequence of pre-processing can generate suitable binary masks that can improve the precision when trying to detect defects, as shown by Aslam et al. in [33]. The defects in metal surfaces that were studied did not include small defects with similar characteristics as pinholes, but still demonstrated an effective method that can be modified with advanced clustering algorithms for classification. Another work that uses the k-means clustering algorithm for segmentation to improve the localization of defects is the one proposed by Li et al. ([34]), which combined this algorithm with particle Swarm optimization for images of seal parts. This type of techniques could potentially improve the generation of segmentation masks for further processing, but would still need a special configuration so that small defects are not taken as part of the background. Further methods for image segmentation for defects in metal surfaces are discussed by Tan et al. ([35]), where some defects look similar to the ones present in our database, such as the patches, inclusion and scratches.

The work of Wang et al. ([36]) demonstrated that the model YOLOv5 provided effective results for the detection of small defects in metal surfaces. The proposed scheme combines an improved version of YOLOv5, data augmentation and an asymmetric loss function to obtain a mAP of 74,1%. This supports the idea that the model SAHI+YOLOv8 can be improved with custom training in a sufficiently large dataset, customization of the structure of the architecture and the use of more effective pooling filters that avoid the loss of defects with the background.

Chapter 4

Conclusion

The main motivation of this work was to investigate the capabilities of machine learning approaches for object detection with a small dataset of images containing different types of defects. The foremost application was to detect and classify defects of ceramic coating films prepared at IOM. The following conclusions can be drawn from the results and discussions in the previous chapter:

- A dataset of less than 100 images might be usable for custom training for object detection recognition with architectures based on common object datasets, as long as the objects to be detected are sufficiently large and there is a balanced amount of labels among classes. However, this method has the following limitations: in real world applications, we do not have a balanced set to avoid underrepresented classes and labeling instances can be a tedious time-consuming task with an increased difficulty when drawing the ground truth bounding boxes.

- Small objects are difficult to detect. Therefore, approaches similar to SAHI must be applied for slicing the images and make the small defects appear "larger" for the detector. Smaller grids will improve the precision at localizing small defects, but this represents a limitation at the same time, as larger defects will be confused for the background. Screening at different scales and further comparison of predictions could provide a more accurate model to localize instances of all sizes.

- For a fast approach, some architectures that were pre-trained on a common object dataset can adapt to detect new objects, such as the case of a dataset with defects of a material. The adaptability is limited when it comes to small objects, but this could be improved by using the slicing inference method. Subsequent efficient classification can be done by means of clustering algorithms, such as k-means, as long as the algorithm and input features for clustering are appropriate.

Outlook

The following points are suggestions for future work based on the experience of this thesis.

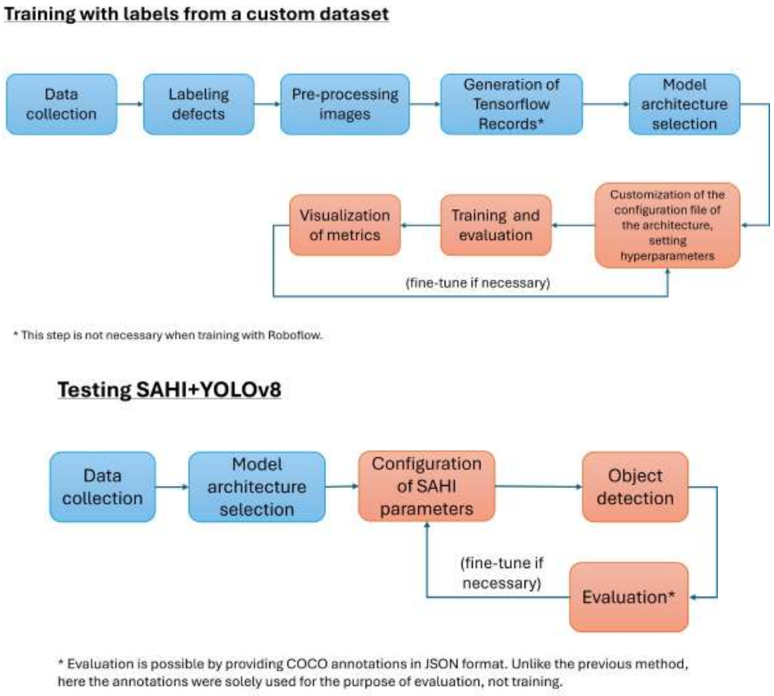
Creation of a dataset: A sufficiently large dataset would contain at least 100 images, ideally close to 1.000 or more, as most datasets for training architectures are in the order or hundreds of thousands. This amount of information would be able to provide enough information for neural networks to locate and identify objects with better accuracy. It is important that classes are not underrepresented, as that would have the same effect than having a small dataset. If the dataset contains images from different experiments with common objects to be found in this area, this can serve as a tool to train architectures and modify the neural networks according to the provided batch of training images. The election of architecture would depend on the computational resources available, training time and resolution. Architectures with access to customization will allow to adjust the parameters to fit better with the type of data provided, as most layers are designed to detect common objects. Labeling and organization of this dataset will require a considerable amount of time, but it may serve as a tool to provide quicker and more precise analysis in the future.

Usage of unsupervised learning techniques: By using clustering algorithms on the pixels, it is possible to "highlight" characteristic features of the defects and make a higher contrast with the background, increasing the chances of being detected. Unsupervised object detection approaches such as CutLer ([37]) can be used for the localization of the defects. This method identifies objects by the similarity of pixels. Therefore, pre-processing might improve the capabilities to detect objects. Depending on the type of images and objects, different configurations for the initial pixel segmentation must be studied to understand how the clustering algorithm affects the subsequent object detection. Implementing an approach like SAHI will have good chances to identify small objects. After object localization, the detected instances can be classified with unsupervised clustering algorithms (for example, k-means, DBSCAN, agglomerative clustering) according to characteristic features (for example, area, perimeter, average color). The type of clustering algorithm and characteristic features will depend on the type of objects to be classified. It is important to consider which features are the most distinct among different classes, and what shape the clusters have. In case this works, the result is a fully unsupervised approach for object detection and clustering. Clusters can be easily assigned to a certain defect type by randomly selecting representatives from a cluster. Thus, only a few images with a single defect must be labeled. This is much easier than drawing bounding boxes and label each of them as it is done with supervised learning techniques.

K-means clustering: The two main types of small defects are particles (light colored) and pinholes (dark colored), which could be separated by using the k-means clustering algorithm. This algorithm uses the masks produced by instance segmentation to determine the “average” color of the detected instance. One of the problems in this work was that masks were not accurate enough when determining the limits of the defects, meaning that many masks considered the background as part of the defect. The background in most images was a gray shade, which in the normalized gray-scale axis would be located close to the middle. Masks that considered the background as part of the defect ended up shifting the value of the “average” color of the defect and drove it towards the middle. This was visible in both models where k-means clustering was employed. The clustering ended up being a classification by size only with no influence of the color for classification. One suggested solution for this problem is averaging only over a sub-group of pixels, consisting only of 10% of the lightest and 10% of the darkest pixels of the mask. This way, the average will be shifted towards the most intense of these two groups: particles will be shifted closer to light color values, and pinholes will go towards darker color zones. There will still exist a shift from the actual color of the defect, but this technique may be enough for separating two clusters in the region of small defects.

Appendix

Figure 1 illustrates the process of how the different methods were used for object detection. The upper image corresponds to the approach with additional training on a custom dataset. The lower image does not implement additional training, but a slicing method to improve the detection of small objects.



Training with labels from a custom dataset

Data collection
Labeling defects
Pre-processing images
Generation of Tensorflow Records*
Model architecture selection
Customization of the configuration file of the architecture, setting hyperparameters
Training and evaluation
Visualization of metrics
(fine-tune if necessary)

* This step is not necessary when training with Roboflow.

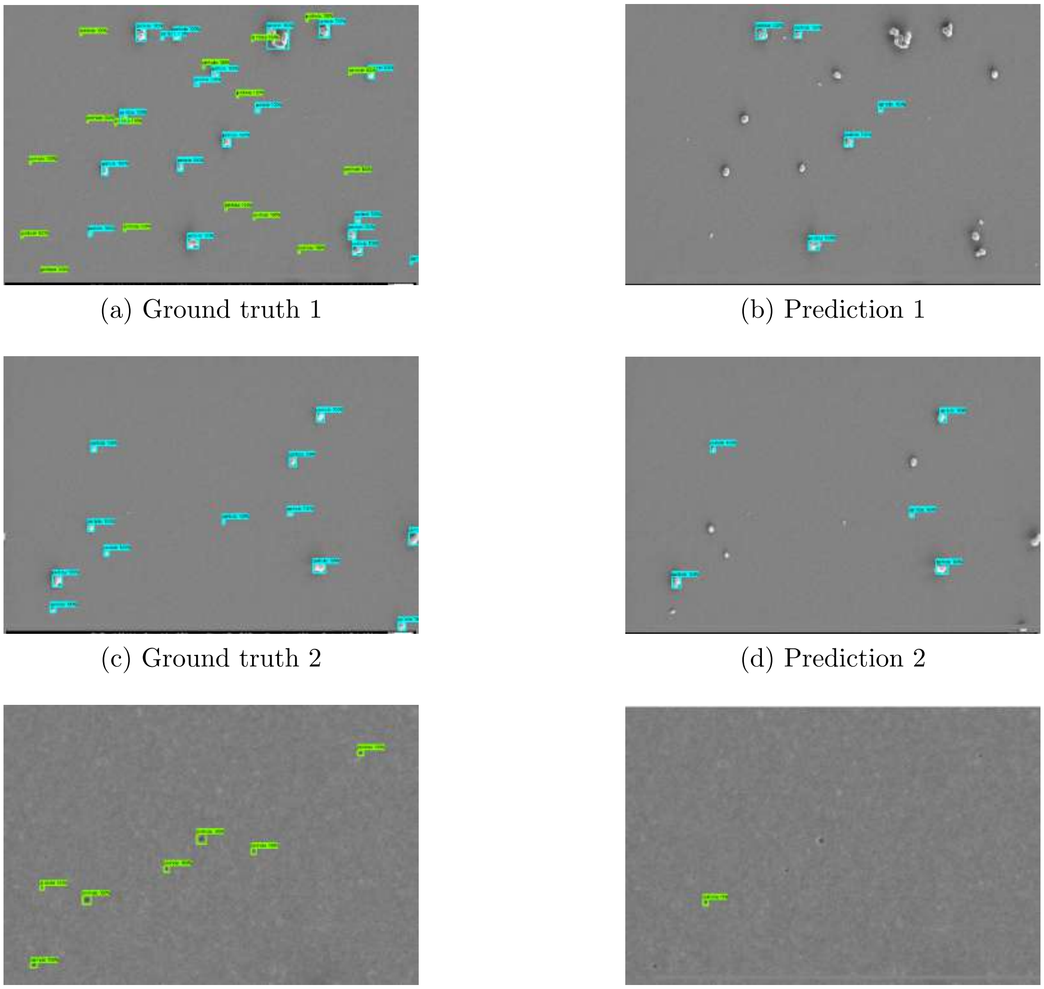
Testing SAHI+YOLOv8

Data collection
Model architecture selection
Configuration of SAHI parameters
Object detection
Evaluation*
(fine-tune if necessary)

* Evaluation is possible by providing COCO annotations in JSON format. Unlike the previous method, here the annotations were solely used for the purpose of evaluation, not training.

Figure 1: Workflow for object detection

Figure 2 contains images used on the evaluation of the architecture Faster R-CNN. Images on the left column contain the ground truth bounding boxes provided for the test set and images on the right column contain the boxes determined by the model after training.



(a) Ground truth 1

(b) Prediction 1

(c) Ground truth 2

(d) Prediction 2

(e) Ground truth 3

(f) Prediction 3

Figure 2: Side by side comparison of the predictions of the ground truth labels provided to the model Faster R-CNN Resnet50 V1 and the obtained predictions.

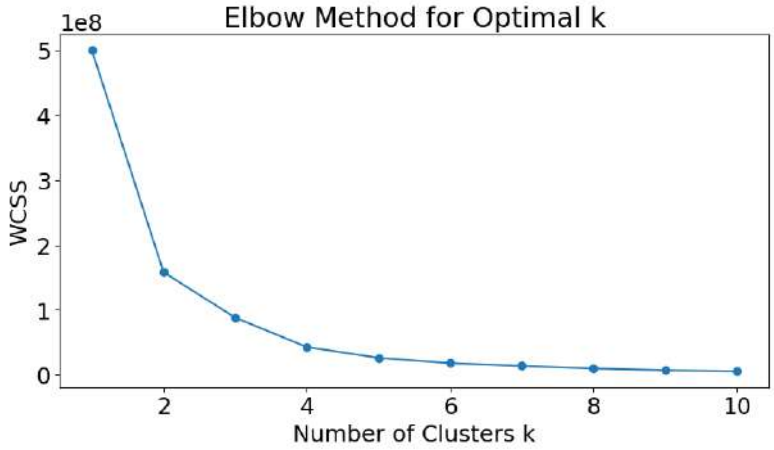


Figure 3: WCSS as a function of the suggested amount of clusters k.

Figure 3 was used when testing the "elbow" method for determining the ideal amount of clusters k for the Small slices model

Bibliography

1. Hanika, M., Langowski, H.-C., Moosheimer, U. & Peukert, W. Inorganic Layers on Polymeric Films – Influence of Defects and Morphology on Barrier Properties. *Chemical Engineering & Technology* **26,** 605–614. https://onlinelibrary.wiley.com/doi/abs/10.1002/ceat.200390093 (2003).

2. With, P. C., Helmstedt, U. & Prager, L. Flexible Transparent Barrier Applications of Oxide Thin Films Prepared by Photochemical Conversion at Low Temperature and Ambient Pressure. *Frontiers in Materials* **7.** https://www.frontiersin.org/journals/materials/articles/10.3389/fmats.2020.00200/full (2020).

3. Da Silva Sobrinho, A. S., Czeremuszkin, G., Latrèche, M. & Wertheimer, M. R. Defect-permeation correlation for ultrathin transparent barrier coatings on polymers. *Journal of Vacuum Science & Technology A* **18,** 149–157. ISSN: 0734-2101. https://doi.org/10.1116/1.582156 (Jan. 2000).

4. Géron, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* ISBN: 978-1-098-12597-4. https://www.oreilly.com/library/view/hands-on-machine-learning/9781098125974/ (O'Reilly Media, Sebastopol, CA, 2023).

5. IBM. *What are convolutional neural networks?* Oct. 2021. https://www.ibm.com/topics/convolutional-neural-networks.

6. Szegedy, C., Toshev, A. & Erhan, D. *Deep Neural Networks for Object Detection* in *Advances in Neural Information Processing Systems* (eds Burges, C., Bottou, L., Welling, M., Ghahramani, Z. & Weinberger, K.) **26** (Curran Associates, Inc., 2013). https://proceedings.neurips.cc/paper_files/paper/2013/file/f7cade80b7cc92b991cf4d2806d6bd78-Paper.pdf.

7. Babs, T. The Mathematics of Neural Networks. *Coinmonks.* https://medium.com/coinmonks/the-mathematics-of-neural-network-60a112dd3e05 (2018).

8. Montesinos López, O. A., Montesinos López, A. & Crossa, J. in *Multivariate Statistical Machine Learning Methods for Genomic Prediction* 379–425 (Springer International Publishing], 2022). ISBN: 978-3-030-89010-0. https://doi.org/10.1007/978-3-030-89010-0_10.

9. Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 779–788 (2016).

10. Abadi, M. *et al. TensorFlow: A System for Large-Scale Machine Learning* in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (2016), 265–283.

11. Lin, T. *et al.* Microsoft COCO: Common Objects in Context. *CoRR* **abs/1405.0312.** arXiv: 1405.0312. http://arxiv.org/abs/1405.0312 (2014).

12. Mingxing, T., Ruoming, P. & Quoc V., L. EfficientDet: Scalable and Efficient Object Detection. *arXiv:1911.09070v7.* https://arxiv.org/pdf/1911.09070 (July 2020).

13. Shaoqing, R., Kaiming, H., Ross, G. & Jian, S. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv preprint arXiv:1506.01497* (2015).

14. Caron, M., Bojanowski, P., Joulin, A. & Douze, M. Deep Clustering for Unsupervised Learning of Visual Features. *CoRR* **abs/1807.05520.** arXiv: 1807.05520. http://arxiv.org/abs/1807.05520 (2018).

15. Caron, M. *et al.* Unsupervised Learning of Visual Features by Contrasting Cluster Assignments. *CoRR* **abs/2006.09882.** arXiv: 2006.09882. https://arxiv.org/abs/2006.09882 (2020).

16. Akyon, F. C., Altinuc, S. O. & Temizel, A. Slicing Aided Hyper Inference and Fine-tuning for Small Object Detection. *2022 IEEE International Conference on Image Processing (ICIP),* 966–970 (2022).

17. Akyon, F. C. *et al. SAHI: A lightweight vision library for performing large scale object detection and instance segmentation* 2021. https://doi.org/10.5281/zenodo.5718950.

18. Padilla, R., Netto, S. L. & da Silva, E. A. B. *A Survey on Performance Metrics for Object-Detection Algorithms* in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)* (2020), 237–242.

19. Padilla, R., Passos, W. L., Dias, T. L. B., Netto, S. L. & da Silva, E. A. B. A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit. *Electronics* **10.** ISSN: 2079-9292. `https://www.mdpi.com/2079-9292/10/3/279` (2021).

20. Lin, T., Goyal, P., Girshick, R. B., He, K. & Dollár, P. Focal Loss for Dense Object Detection. *CoRR* **abs/1708.02002.** arXiv: `1708.02002`. `http://arxiv.org/abs/1708.02002` (2017).

21. Števuliáková, P. & Hurtik, P. *Intersection over Union with smoothing for bounding box regression* 2023. arXiv: `2303.15067 [cs.CV]`. `https://arxiv.org/abs/2303.15067`.

22. Schneider, C. A., Rasband, W. S. & Eliceiri, K. W. NIH Image to ImageJ: 25 years of image analysis. *Nature Methods* **9,** 671–675 (2012).

23. Ferreira, T. & Rasband, W. S. *ImageJ User Guide — IJ 1.46* (2012). `https://imagej.nih.gov/ij/docs/guide/`.

24. Tzutalin. *LabelImg: Data labeling software* 2015. `https://github.com/tzutalin/labelImg`.

25. Yu, H. *et al. TensorFlow Model Garden* `https://github.com/tensorflow/models`. 2020.

26. Dwyer, B., Nelson, J., Hansen, T., *et al. Roboflow (Version 1.0) [Software]* Computer Vision. 2024. `https://roboflow.com`.

27. Gallagher, J. *Announcing Roboflow Train 3.0* Roboflow Blog. 2023. `https://blog.roboflow.com/roboflow-train-3-0/`.

28. Weigert, M., Schmidt, U., Haase, R., Sugawara, K. & Myers, G. *Star-convex Polyhedra for 3D Object Detection and Segmentation in Microscopy* in *The IEEE Winter Conference on Applications of Computer Vision (WACV)* (2020).

29. Weigert, M. & Schmidt, U. *Nuclei Instance Segmentation and Classification in Histopathology Images with Stardist* in *The IEEE International Symposium on Biomedical Imaging Challenges (ISBIC)* (2022).

30. Schmidt, U., Weigert, M., Broaddus, C. & Myers, G. *Cell Detection with Star-Convex Polygons* in *Medical Image Computing and Computer Assisted Intervention - MICCAI 2018 - 21st International Conference, Granada, Spain, September 16-20, 2018, Proceedings, Part II* (2018), 265–273.

31. Thirunavukkarasu, S. *et al.* Methodology for non-destructive assessment of integrity of steam generator shell welds. *International Journal of Structural Integrity* **2,** 145–157 (May 2011).

32. Altmann, M. L., Benthien, T., Ellendt, N. & Toenjes, A. Defect Classification for Additive Manufacturing with Machine Learning. *Materials* **16,** 6242 (2023).

33. Aslam, Y., Santhi, N., Ramasamy, N. & Ramar, K. Defect Detection of Coatings on Metal Surfaces Based on K-Means Clustering Algorithm. *International Journal of Recent Technology and Engineering (IJRTE)* **8,** 5782–5786. ISSN: 2277-3878. `http://www.ijrte.org` (2019).

34. Li, X., Zhu, J., Shi, H. & Cong, Z. Surface Defect Detection of Seals Based on K-Means Clustering Algorithm and Particle Swarm Optimization. *Scientific Programming* **2021,** 3965247. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1155/2021/3965247`. `https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/3965247` (2021).

35. Tan, C. W. *et al.* in, 39–49 (Springer Nature Switzerland, Apr. 2023). ISBN: 978-3-031-30236-7.

36. Wang, K., Teng, Z. & Zou, T. Metal Defect Detection Based on Yolov5. *Journal of Physics: Conference Series* **2218,** 012050. `https://dx.doi.org/10.1088/1742-6596/2218/1/012050` (Mar. 2022).

37. Wang, X., Girdhar, R., Yu, S. X. & Misra, I. *Cut and Learn for Unsupervised Object Detection and Instance Segmentation* 2023. arXiv: `2301.11320 [cs.CV]`. `https://arxiv.org/abs/2301.11320`.