Ernesto Lazaro Garcia

CLASSIFIER OPTIMIZATION VIA DATA AUGMENTATION TECHNIQUES FOR IMBALANCED SEM IMAGE DATASETS

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Biomedical Engineering

submitted to

Graz University of Technology

Supervisor

Pock, Thomas, Univ.-Prof. Dipl.-Ing. Dr.techn.
Institute for Computer Graphics and Vision

Dohr Judith
Infineon Technologies

Graz, Austria, Sep. 2020

Acknowledgments

First and foremost I would like to thank Prof. Pock for his guidance and input throughout this process. Additionally, I would also like to thank Judith Dohr, Corinna Kofler, Anja Zernig and Yury Bodrov for their valuable support during my stay and work at Infineon. I would like to thank my sister, Helena Lazaro Garcia, and her fiance, Carlo Rubino, for their help and advice during my writing. Lastly, I wish to thank all of my friends for their continuous support.

Contents

List of Figures

List of Tables

1 Abstract

Data imbalances consist of an uneven distribution of the dataset in Machine Learning, where one or more classes have a number of instances significantly higher than the others. As a consequence, the algorithm will prioritize learning the majority classes, while the minority will be poorly estimated. This problem affects the quality and reliability of machine learning tasks, and several techniques have been developed in order to manage it. These are, among others, undersampling, oversampling and weight loss balancing. In this work, our main goal is to investigate the effects of these techniques on a set of convolutional neural network models from the EfficientNet architecture in Scanning Electron Microscopy (SEM) image classification during wafer manufacturing. Since current inspection of wafer defects in the production line is performed manually, and has proven to be a tedious and time consuming task, this strategy would constitute a powerful alternative for automated defect detection that would speed up considerably the wafer inspection process. Our work proves that this approach leads to significant headcount reduction in the wafer production line, with accuracy equivalent to that of an actual operator, to whom defect classification not only becomes now faster, but also more manageable.

2 Introduction

2.1 Motivation

Out of all the branches from the semiconductor industry, wafer manufacturing constitutes the core of the business. Since even the smallest speck of dust is capable of harming the microcircuit structure of a wafer, whose features fall within the nanoscale size, this process requires the use of high precision equipment to ensure efficient detection. To further avoid the entrance of any kind of particle within the wafer, the production of such devices is performed in a clean room, that is, a hermetically sealed environment. There's a wide range of defect classes, and they possess a long tail distribution, which constitutes a source for data imbalances.
Silicon has usually been the preferred material for building the microcircuit, which starts out completely blank. First, through a process called photo-masking, the different photoresist patterns are masked onto the wafer's surface in micrometer detail. Afterwards, short-wave ultraviolet light is used in such a manner that the unexposed areas are etched away and cleaned. On the desired zones of the wafer surface, chemical vapors will be deposited in such a way that, as they are baked at high temperatures, they'll heat enough to permeate into the desired zones. Ions can also be implanted at specifics depths and patterns into the wafer through the use of RF-driven ion sources.
The number of times these steps need to be repeated depends on the complexity of the wafer, and are usually repeated hundreds of times. As time goes on, new techniques have been developed in order to improve the resolution with with these steps are completed. As new technologies arise, so does the scale of the circuits, and microcircuit features such as transistors and micro-electro-mechanical systems (MEMS) are packed in a denser manner. This increased density continues the trend of Moore's Law.
Since wafers require high sensitivity technology for surface inspection, devices such as Scanning Electron Microscopes are one of the main tools used for such task, which can be configured to review defects found on the wafer. In short, SEMs can enlarge the defect in such a way that it can be properly reviewed and classified by an operator, for these

high magnification images contain a detailed representation of the surface topography and composition.
Human visual inspection no longer constitutes a feasible approach for defect detection and classification, and computer vision systems such as Convolutional Neural Networks (CNNs) have proven to be the best approach due to both cost reduction and efficiency [41],[43],[44]. Though the current networks used for classification have respectable accuracy, there are available deeper and more powerful architectures developed in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), a project designed for research use in the field of visual object recognition software [52].
The purpose of this thesis is to improve defect classification in the production line of Infineon Technologies Villach through data augmentation techniques (oversampling). Additionally, we plan to replace the current model used for classification with a state-of-the-art architecture, EfficientNet [13],[38]. We faced several problems along the way, such as label noise, hardware limitations, noisy images, bugs, etc. However, we managed to implement the techniques we desired.
We compared the reliability and performance of different models from the EfficientNet family, with different oversampling magnitudes and hyperparameters to investigate its benefits. Preprocessing tuning as well as further inspection of techniques such as class weighted loss, label smoothing and Geometrical Augmentation (GA) allowed us to further optimize our classifier. Both class precision and recall are our main measure of model performance, and visual inspection of heatmaps and saliency maps provide additional insight into the networks.
Chapter 3 provides background information surrounding the main object of this thesis. We introduce the basics of Machine Learning and Deep Learning, followed by the main components of a wafer fabrication plant and SEM defects. In **Chapter 4**, we discuss the different preprocessing techniques used, and justify the reason behind their selection. **Chapter 5** comprehends the implementation and results of these techniques, and constitutes the base upon which we'll determine our final parameters and models, whose performance we'll be included in **Chapter 6**. In **chapter 7** we'll discuss our results and evaluate alternative techniques for model optimization. Finally, we close the paper with a brief conclusion in **Chapter 8**.

2.2 Related Work

Image classification. Our main goals consists on improving as much as possible defect classification, that is, an image classification task. The performance of algorithms related to such field is usually evaluated in datasets such as Imagenet [52]. With the passage of time, improving the architecture of neural networks has been a hot topic in the scientific community, and a lot of effort has been invested into the realization of such goal. For example, the introduction of residual connections in Convolutional Neural Networks has led to the development of competitive architectures [18],[31],[1],[13] widely adopted by

the community.
However, the introduction for residual connections has not been the only direction taken for pushing the development of state-of-the-art networks [24]. Other approaches, such as those based on high input image resolution, a key factor in classification performance [25], led to the development of enormous Convolutional Neural Networks [30]. Other works include the use of weakly labeled datasets for high-capacity CNN development [26]

Knowledge distillation. Also known as Transfer Learning, in this task a high architecture model, teacher, is used in order to train a small one, student. In order to achieve this, the large model is trained using a great amount of labelled samples, for which other tasks do not have enough data available. Therefore, the teacher must select only those instances that are most informative, which therefore leads to a better training of the student network.
Examples of tasks where this technique can be applied include fine-grained classification, where we lack a proper amount of labels, properly process multi-labeled datasets [48], learning adequate curriculums for the training dataset [47], and instances where we face noisy labels, such as for MentorNet [49]. Finally, [27] provides further insight into the different challenges faced in this field.

Data augmentation. This technique has proven successful at improving model performance, be it by estimating the best transformations that can be used for augmentation [33],[36], or by creating artificial samples through the mixture of images [34]. Semi-supervised learning, where new images are added to the main dataset from a supplemental unlabeled one, has also been used for performance improvement [42]. So far, data augmentation has proven essential to optimize image classification tasks [28].
In the field of automated data augmentation, [36] has developed an algorithm capable of automatically searching for the best transformations that can be applied in such field. In this paper, a policy in the search space is composed of many sub-policies, and each sub-policy is composed of a pair of image transformations such as rotation, shearing, etc. as well as their associated probabilities and magnitudes. Each sub-policy will be selected randomly per mini-batch, and the goal of the search algorithm is thus to find the best possible policy, through a separate search phase, that leads to the highest validation accuracy.
However, in [33] a new approach is proposed where no separate search phase is required. To achieve this, the search phase is reduced in such a dramatic manner that a simple grid is sufficient to find data augmentation policies powerful enough to outperform previous studies, see [36], where a separate search phase couldn't be avoided. This low size method of data augmentation is termed *RandAugment*.

One-shot image recognition. Unlike other branches of deep learning, one can assess that one-shot learning has not received as much attention by the machine learning community as with other fields. Thus, one-shot learning still remains somewhat immature, but some works have been done that research the boundaries of this field. In the early 2000's, [50] developed a variational Bayesian framework for one-shot image classification. Their premise was that, given very few examples from a specific class/es, one could use the previously learned classes as leverage in order to help determine future ones.

Another method based its approach on ranking the similarity between inputs [41]. This paper looked to capitalize on the discriminative features between samples thanks to the unique architecture of their network, which allows a generalization of the predictive power of the network not only to new data, but also classes never seen before from completely unknown distributions.

2.3 Theoretical background

2.3.1 Machine learning

Machine learning consists of the study and development of computer algorithms capable of learning and improving automatically through the experience they gather when facing the desired task, without being explicitly programmed to learn such task.

Taking a set of sampled data, known as a training set, these algorithms are capable of constructing a mathematical model precise enough to perform reliable predictions. They have gained a great deal of reputation due to their high performance in applications where using conventional algorithms has proven either difficult or infeasible, this includes game theory, information theory, computer vision, etc.

Machine learning algorithms can be categorized as follows:

1. **Supervised machine learning algorithms:** the algorithm is capable of learning a function that properly maps an input to its corresponding output based on labeled samples. With a known training set as the starting point, the learning algorithm is capable of inferring a function that predicts output values by comparing the generated/predicted outputs with the actual/correct output, finding any errors and modifying the parameters of the model accordingly. After sufficient training is done, the algorithm should be able to efficiently generalize the training data so that it can properly classify new unseen data reasonably.

2. **Unsupervised machine learning algorithms:** unlike supervised learning, the information used for training is neither classified nor labeled. The goal of the algorithm for such task is now different, to explore the given dataset, find out and learn any hidden features contained within it. Therefore, one of the most common tasks associated with this category of machine learning is cluster analysis, which focused

on grouping the data in a feasible manner through measures of similarity between samples, such as probabilistic or Euclidean distance.

3. **Semi-supervised machine learning algorithms:** a middle point between supervised and unsupervised learning, these algorithms make use of partially labeled datasets. These datasets usually contain a high proportion of unlabeled samples while only a small amount of labeled data is available. This technique is usually used when the required labeled data for training must come from a skilled and relevant source. A perfect example for this is text documents, which are predominantly composed of unlabeled data (blogs, scripts, novels, etc.).

4. **Reinforcement machine learning algorithms:** these algorithms have a specific goal in mind, the maximization of their cumulative reward. In order to achieve this, they must take a set of actions in a given environment and automatically determine from these interactions the ideal behavior that must be followed within a specific context in order to maximize performance. Since a balance between exploration and exploitation must be reached in order to optimize performance, features such as trial and error search and the notion of delayed rewards are most relevant in reinforcement learning.

In this work, we'll focus on supervised Machine learning in the field of computer vision (image classification).

2.3.2 Neural Networks

Neural networks are a variety of algorithms modeled after the structure of the human brain, composed of a set of connected input/output nodes (perceptrons) in which each connection has a weight associated with it. The network is capable of learning a task by adjusting automatically its associated weights so that it can reliably predict the actual labels of the given inputs.

Artificial neurons, Figure 2.1, also known as perceptrons, are the basic processing units of the model [43]. They consist of a parametrized function that maps $R^D \mapsto R$, with weights, w_n, bias, b_n, and activation function, f, as parameters.
For a Neural Network to learn a specific task, it requires a training dataset composed of the known inputs and their corresponding result. With these pairs the algorithm is capable of forming probability-weighted associations between the two of them, which will be embedded within the network itself. Given an input sample, determining the actual difference between the processed predicted output of the network and the actual output associated with said input will constitute the learning procedure during training. This difference between predicted and correct is known as the error, and will be used in order to adjust the weights accordingly, w_n, b_n, based on a specific learning rule. As the learning procedure continues, more and more adjustments will take place so that the network will

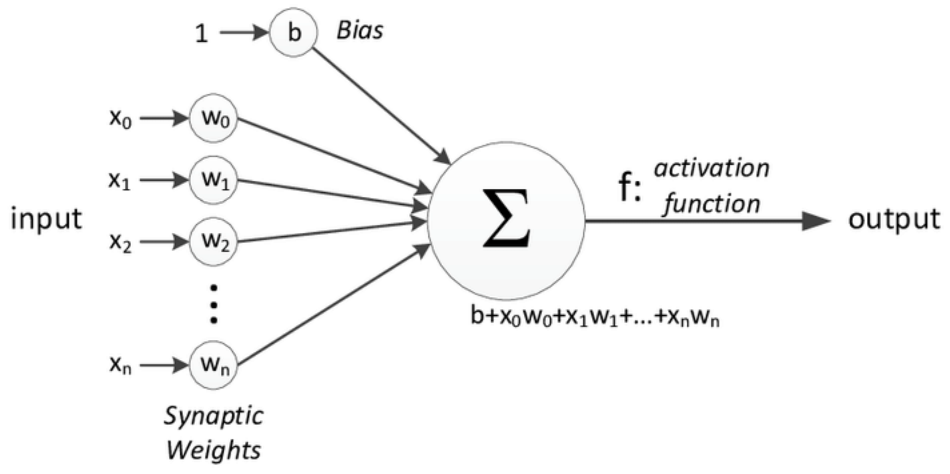


Figure 2.1: Perceptron learning process scheme [2],[3]. The data, x_n, is fed into the input layer, to be later on multiplied by their weight, w_n, add the bias, b_n, and finally feed the sum to the activation function, f, to produce the output.

be capable of producing predictions increasingly similar to the target output. After sufficient training, the learning process will be terminated based upon certain criteria. This process constitutes an example of supervised learning.
While the bias and weights are both adjustable, the output of the neurons can actually range from *-inf* to *+inf*. Through the use of activation functions as our mapping mechanism, we are capable of setting the boundaries of the neurons. The activation function can be understood as an abstraction of the rate of action potential firing in the neuron, but for non-trivial problems, only non-linear activation functions are feasible in solving them. Non-linear activation functions allow the development of a complex mapping between inputs and outputs, which is essential when facing complex datasets (video, audio, etc.) that usually feature high dimensionality, non-linearity, etc. Non-linear activation functions are referred to as *nonlinearities*, and provide the network expressiveness.
There's a wide range of activation functions available for model training, Figure 2.2, among the most commonly used we can find:

- **Linear or Identity function:** the derivative of the function is a constant, and therefore shares no relationship with the input, X, which leads to several disadvantages. For example, it's not possible to optimize performance by determining which weights in the input neurons can provide a better prediction. Additionally, no matter how large the architecture of the network is, given that all layers of the network collapse into one, the last layer will be a linear function of the first one.

- **Heaviside or Binary Step function:** if the value of Y is above a certain threshold, the output is *True*, and if it's less than the threshold, then it's *False*. Due to the fact that the step-function is discontinuous and thus non-differentiable, this function is not done in practice with back-propagation.

- **Binary Sigmoid function:** logistic function where the input values are mapped into probabilities between 0 and 1. Unlike the Heaviside function, this function is differentiable and its output normalized, which usually facilitates learning. However, it suffers from the vanishing gradient problem, that is, given very high/low input values, there is little to no change in the prediction, causing the gradient to vanish.

- **Hyperbolic Tangent function or Tanh:** logistic function where the output value varies from -1 to 1. It facilitates modeling strongly positive, negative and neutral input values, since the function is zero-centered. However, just like the Binary sigmoid function, it also suffers from the vanishing gradient problem, but tends to work better than the sigmoid.

- **Ramp function:** this function maps a range of inputs to outputs over the range from 0 to 1, so it looks like a more definitive version of the sigmoid function. However, it imposes definitive cut off points T1 and T2 in the x-axis, so it can also be understood as a truncated version of the linear function. It allows some level of uncertainty in the lower regions, while also possessing the ability to fire the node very definitively above the threshold.

- **Rectified Linear Unit (ReLU):** this function maps inputs to proportional output values, and outputs zero for negative input values. It can be seen as a special case of the ramp function, and though it might look linear, in reality it's not and therefore possibilitates backpropagation. However, it also has its disadvantages, such as suffering from the dying ReLU problem, that is, when the inputs are negative, the gradient of the function becomes zero and back-propagation cannot be performed, which means that the network is unable to learn. It's one of the most used activation functions worldwide in a wide variety of powerful Convolutional Neural Network (CNN) architectures.

Neural Networks are usually structured into layers, rows of artificial neurons/nodes, but there are three different kinds of layers present in almost all Neural Network architectures. These are the input, hidden and output layer.
The initial data is brought into the system through the input layer of the Neural Network, this data will be later on processed by the subsequent layers of the network. It is important to note that input layer nodes are addressed as 'passive' due to their lack of weights and biases, since they constitute the very beginning of the workflow of the network and therefore do not take information from any previous layers.
Between the input and the output layers one can find the so-called hidden layers. In these layers the function applies weights to the inputs and directs them through an activation function as the output. They are nothing more than mathematical functions designed to produce an output specific to an intended result, and vary significantly depending on

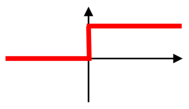
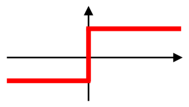
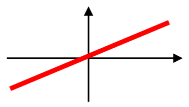
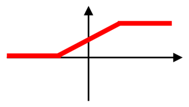
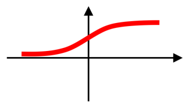
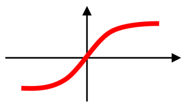
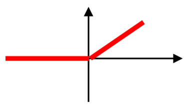
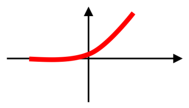
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \dfrac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

Figure 2.2: Activation functions table [4]. In order to ensure proper learning, all activation functions must fulfill two requirements: ensure non-linearity as well as large gradients through the hidden units.

the function of the neural network and their associated weights. Each layer's output is simultaneously the subsequent layer's input, and the further you advance into the network, the more complex the features your nodes are able to recognize.
At the very end of the neural network one can find the output layer, responsible for producing predictions for the network given the input. Their neurons can be seen as identical to that of the rest of the layers, but given that they are the last "actor" nodes on the network, they are addressed in a rather different view. Thus, the output layer is responsible for merging and producing the concrete end result of the model.

The variety of Neural Networks is quite broad, but the most well-known architectures can be counted into one of the following categories:

- **Feed-Forward Neural Networks:** where the first layer constitutes the input and the last layer the output of the model, they are referred to as 'deep' neural networks as well, so long as they have several hidden layers in their architecture. Each layer's neuron activities are a non-linear function of the activities in the layer below, and the network overall computes a series of transformations that change the similarities between cases. They are by far the most common type of neural network for practical applications.

- **Recurrent Networks:** characterized by containing directed cycles, loops, in their architecture. Despite the fact that they are biologically more realistic, these networks are very difficult to train due to their complicated dynamics. These loops in their connection graph, the presence of distributed hidden states and complex updates of these states thanks to their non-linear dynamics grant these networks the ability to store information about the past efficiently, making them capable of remembering information in their hidden state for long periods of time

- **Symmetrically Connected Networks:** unlike recurrent neural networks, with which they share similar architectures, the connections between nodes for these Symmetrically Connected Networks have the same weight in both directions (symmetry). Therefore, they are more restricted in what they can learn, but at the same time are significantly easier to analyze. Those SCNs that lack any hidden units are known as "Hopfield Networks", and those connected with hidden units are called "Boltzmann machines."

We seek to optimize wafer defect image classification. Therefore, Convolutional Neural Networks, an architecture from the Feed-Forward Neural Network category, will prove the best model for such task.

2.3.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are algorithms in Deep learning capable of differentiating various objects within an image, assigning them their associated weights and biases during training, and ultimately telling one apart from the other. Previous methods required the different filters used for such tasks to be hand-engineered, while with CNNs, these very filters are automatically learned by the network itself given sufficient training. These algorithms are mainly used in computer vision tasks such as image classification, segmentation and object detection.

The connectivity pattern encountered in the organization of the animal visual cortex served as inspiration [19],[20],[21],[23] for the development of Convolutional Neural Networks, whose architectures resemble such structure. Two main features of the visual cortex can be highlighted as key for the later construction of CNNs: first, individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. Second, the receptive fields of different neurons partially overlap such that they cover the entire visual field.
When designing a CNN, it is important to make it in such a way that the architecture efficiently learns the sample's features and can also be scalable to massive datasets. By reducing the format of an image to a new easy-to-process format in such a manner that critical features are not lost/distorted, CNNs are capable of generating good predictions. Each neuron analyzes a small region of the image, and in order to do so these neurons have to be split into 3-D structures, Figure 2.3. The output layer of the CNN will finally generate a probability score vector that indicates the likelihood of a specific sample belonging to one of the known classes.

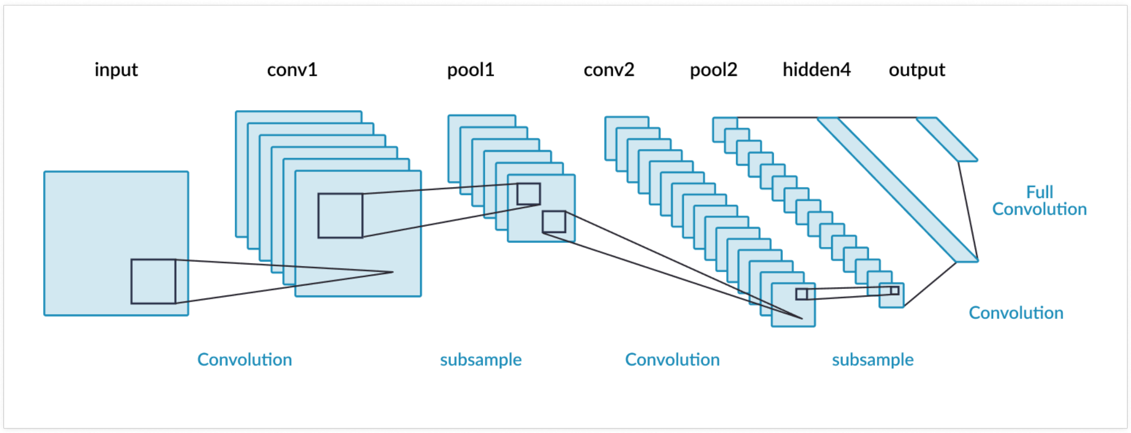


Figure 2.3: Architecture of standard CNN [5]. After the network identifies the main features of the original input by means of convolution, the pooling layers will downsample the results of these to identify further sub-features from smaller parts of the image. This kind of structural logic is used repeatedly in CNNs.

CNNs are composed of the following types of layers:

- **Convolutional layer:** responsible for abstracting the images to a feature map through the use of filters that scan the whole image, reading a few pixels at a time. Convolutional layers can be mathematically understood as a cross-correlation or sliding dot product. Mimicking the biological architecture, each convolutional neuron processes only the data of its receptive field and passes its result to the next layer. Among the different attributes a convolutional layer should have within a network, one can name:

- The convolutional kernels, defined by their a width and height.
- The number of input and output channels.
- The depth of the Convolution filter (the input channels) must be equal to the number channels (depth) of the input feature map.

- **Pooling layer:** through a combination of the outputs of the neuron clusters into a single neuron in the next layer, pooling layers are capable of reducing the dimensions of the input data. One can distinguish local pooling layers, responsible for combining small clusters, from global pooling, which acts on all the neurons of the convolutional layer. In addition, pooling may compute the maximum value from each cluster of neurons at the prior layer, max. pooling, or the average value from each cluster of neurons at the prior layer, average pooling. The most common approach used is max pooling.

- **Fully connected layer:** also known as flatten layers, they are responsible for a full connection of every neuron from one layer to every neuron on the subsequent layer. This way, they turn the outputs into a single vector, which can be used later on as input for the next layer. Usually it is this flattened matrix what is used in order to finally perform image classification.

Many CNN architectures have gained recognition, [18],[29],[31],[40] [1],[13], by achieving state-of-the-art results at the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This annual competition fosters the development of top algorithms in the field of computer vision thanks to a very large collection of human annotated photographs designed by academics. It is thanks to the ILSVRC project that the fields of deep learning and computer vision have actually experienced the birth of true milestone architetures and techniques. Nowadays, the ImageNet dataset counts with more than 14 million images specifically created for the purpose of training CNNs in the field of object detection.

2.3.2.1.1 EfficientNet Architecture

Usually, there's a limited resource budget when designing Convolutional Neural Networks, and as soon as more budget is available, the network is scaled up. There are three different features one can tune in order to scale-up a CNN, that is, their depth [14], width [15] or image resolution [16]. Unfortunately this process requires manual tuning, and though one can scale-up up to two or three dimensions, it tends to lead to sub-optimal performance.
In opposition to these conventional approaches where one arbitrarily scales network dimensions, [13] proposes the so-called *compound scaling method*, that uniformly scales each dimension with a set of fixed scaling coefficients. Through this method, the authors

were capable of generating a lightweight convolutional neural network architecture known as EfficientNet, Figure 2.4, that achieves state-of-the-art performance with up to six commonly used datasets, such as ImageNet, CIFAR, etc.
In [13], the width, depth and resolution of the architecture were uniformly scaled using a compound coefficient ϕ in the following manner:

$$\begin{aligned} depth : d &= \alpha^{\phi} \\ width : w &= \beta^{\phi} \\ resolution : r &= \lambda^{\phi} \\ s.t.\ \alpha.\beta^2.\lambda^2 &= 2 \\ \alpha \geq 1,\ \beta^2 \geq 1,\ \lambda^2 &\geq 1 \end{aligned} \tag{2.1}$$

A small grid search suffices in order to determine the constants α, β and λ, which specify how to assign these extra resources to network width, depth, and resolution respectively. ϕ is user-specified and controls how many more resources are available for model scaling.

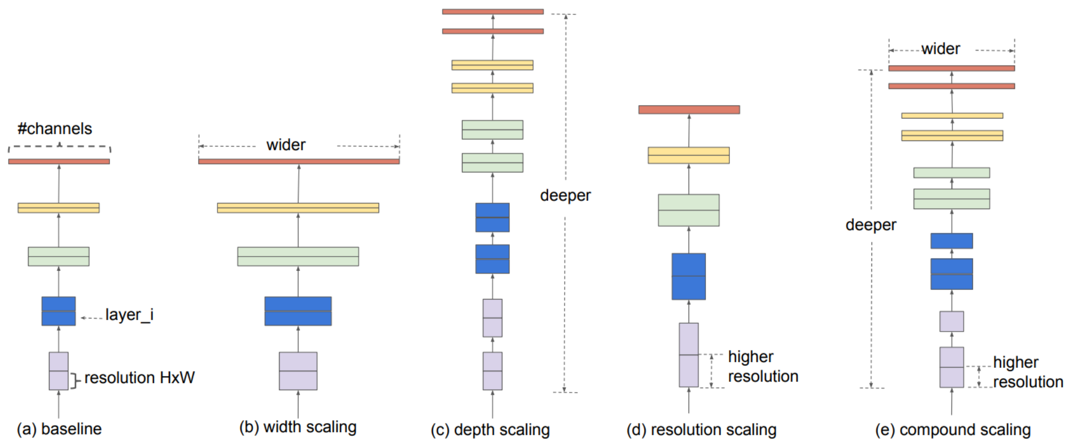


Figure 2.4: EfficientNet model scaling [13]. As stated in the paper: (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) Proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

It is important to note that in [13], a new baseline network was developed by using neural architecture search [17] and subsequently scaled-up in order to obtain the family of models that constitute EfficientNet, which significantly outperform other CNNs such as GPipe [30], ResNet-50 [29], etc. The development of such baseline was done due to

the fact that the effectiveness of the *compound scaling method* heavily depends on the baseline network.
There are currently available 8 different architectures from the EfficientNet family (EficientNet B0-B7), whose computation and build consisted on: for EfficientNet B0, assuming twice more resources available and a small grid search of α, β, λ based on Equation 2.1. They found $\alpha = 1.2\,, \beta = 1.1\,, \lambda = 1.15$ as the best values for this architecture, under constraint of $\alpha\,\beta^2\,\lambda^2 = 2$. For EfficientNet-B1 to B7, they fixed α, β, λ as constants and scaled up baseline network with different ϕ using Equation 2.1, obtaining the remaining architectures.
Given the state-of-the-art performance of the EfficientNet family, we'll investigate this line of models for defect classification. However, due to hardware limitations, we'll limit our scope to the first 5 architectures (EficientNet B0-B4).

2.3.2.2 Loss function

The loss function constitutes a comparison between the prediction performed by the model and the actual label the output should generate. In other words, it computes how poorly our networks performs, being therefore one of the most important components in Neural Network training. Having prediction y_{pred}, if this one differs too much from the actual value y_{true}, then the loss will be consequently very high. On the other hand, as long as both values are close to each other, the loss will become very low.
Hence, the loss function is used to calculate the gradients, which are used to update the weights of the NN. High losses cause the weights to change significantly, while low ones will lead only to small changes. We need to keep a loss which can penalize a model effectively while it is training on a given dataset. This is how Neural Networks learn.
The most essential loss functions, which could be used for most objectives are:

- Mean Squared Error (MSE): calculated by taking the mean of squared differences between the actual and predicted values:

$$MSE = \tfrac{1}{N}\textstyle\sum_{i=1}^{N}(y_{pred} - y_{true})^2$$

- Binary Cross-entropy (BCE): used for the binary classification tasks, where data can only be divided into two classes. The output value should be passed through a sigmoid activation function and the range of the output will consequently lie between 0 and 1:

$$BCE = -\sum_{i=1}^{C=2} y_{true}^{i} log(y_{pred}) = -y_{true} log(y_{pred}) - (1 - y_{true}) log(1 - y_{pred})$$

- Categorical Cross-entropy (CC): for multi-class classification tasks. The number of output nodes must be equal to the number of classes. The output layer should pass through a softmax activation to obtain a probability vector with values between 0 and 1.

$$CC = -\sum_{i=1}^{C=N} y_{true}^{i} log(y_{pred})$$

Since or goal consists in the development of a model capable of classifying 11 different classes, categorical cross-entropy will be our loss function of choice.

2.3.2.3 Backpropagation

In machine learning, backpropagation consists on the computation of the gradient of the loss function one layer at a time with respect to each weight through the chain rule. In order to avoid redundant calculations of intermediate terms in the chain rule, this technique is performed backwards from the last layer of the network, and it is this flow of the error information what allows an efficient computation per layer of the gradient, in opposition to the naive approach of calculating the gradient of each layer separately. Backpropagation is therefore a widely used algorithm in supervised training that tunes a neural network's weights to improve the prediction accuracy.
This technique addresses not only the algorithm for computing the gradient, but also the adjustments of the network's weights during the entirety of the learning process, Figure 2.5, including how the gradient is used in order to update the parameters of the network. A wide range of variants for backpropagation algorithms have been developed, some more complex than others. Among the most well-known are: Root Mean Square backpropagation (RMSProp), Stochastic Gradient Descent (SGD), Nesterov Momentum, and algorithms with adaptive learning rates such as AdaGrad, Adam, RAdam, etc.

- *We want:* $\frac{\delta E_n}{\delta W_{ij}^{(i)}}$

- *Error of neuron 1 in layer L:*

$$\delta_1^{(L)} = y(x_n) - t^{(n)}$$

- *Error of neuron in layer l?*

Credit assignment

$$\delta_i^l = g'(a_i^{(l)}) \sum_j w_{ij}^{(l+1)} \delta_j^{(l+1)}$$

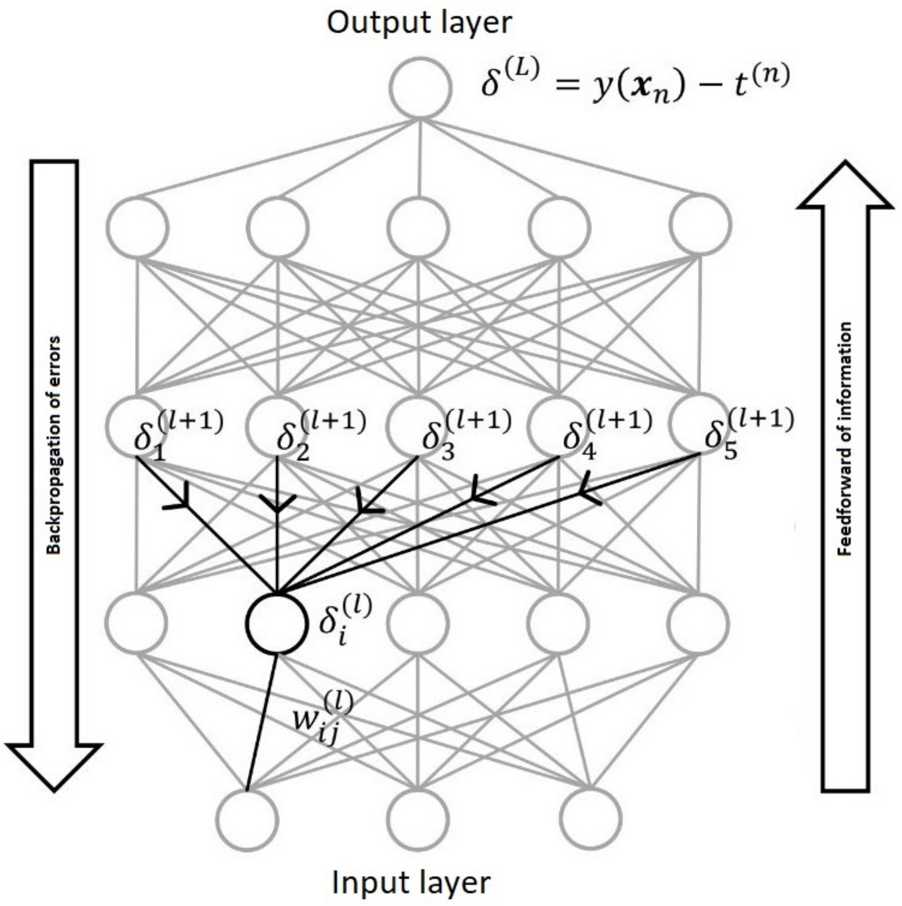


Figure 2.5: Schematic of backpropagation training algorithm [9]**.** Calculate the output for every neuron from the input layer to the output layer. Calculate the error in the outputs and travel back from the output to the input layer to adjust the weights such that the error is decreased.

2.3.2.3.1 Stochastic Gradient Descent

Stochastic gradient descent is based, as its names suggests, on the iterative algorithm known as Gradient Descent, which consists on localizing and reaching the lowest point of a function by traveling down its slope in steps with a randomly selected starting point. Unlike Gradient Descent, SGD replaces the actual gradient, computed from the entire dataset, by an estimate consisting of a randomly selected subset of the dataset or batch. In this manner, SGD can be better understood as an stochastic estimation of Gradient Descent, and it is this estimation what allows faster iterations at the cost of a lower

convergence rate.
In this way, 'mini-batch' stochastic gradient descent, which computes the gradient against more than one training example, can actually make use of vectorization libraries rather than computing each step separately, which translates into a performance that can be significantly better than through the 'true' stochastic gradient descent. Not only that, but it can lead to smoother results due to the fact that an average of the gradient computed per step over the training set is performed. Therefore, 'mini-batch' stochastic gradient descent results from a compromise between computing the gradient at a single example and the actual true gradient.
The learning rate in stochastic gradient descent constitutes a flexible and highly influential parameter in the convergence of the algorithm. Only a single learning rate is maintained, also termed as 'alpha', and it does not change during training. The larger the learning rate is, the larger the will steps taken down the slope be. This could lead to a jump across the minimum point, thereby missing it. Consequently, it is always advised to stick to low learning rates, such as 0.01. However, if the value is too small, it will lead to slow convergence to the optimal point
Theories such as convex minimization and stochastic approximation provided a tool for analysis of the convergence of the stochastic gradient descent. From these theories the following conclusion can be inferred: stochastic gradient descent converges to a global minimum when the objective function is convex or pseudoconvex, and otherwise converges to a local minimum as long as an appropriate decrease of the learning rate is ensured, along with a few relatively mild assumptions
Machine learning algorithms such as graphical models, support vector machines (SVMs) and logistic regression use stochastic gradient descent as their default algorithm for training, and it is the combination of this technique with backpropagation what constitutes the *de facto* standard algorithm for training artificial neural networks. Its formulation follows:

Let ϵ be the learning rate for update k.
Sufficient condition to guarantee convergence of SGD.

$\sum_{k=0}^{\infty} \epsilon_k = \infty$, *and* $\sum_{k=0}^{\infty} \epsilon_k^2 < \infty$

Commonly used in practice:

- *Start with initial rate* ϵ_0
- *Decrease linearly until update* τ*:*
  $\epsilon_k = (\frac{1-k}{\tau})\,\epsilon_0 + \frac{k}{\tau}\epsilon_\tau$
- *Then keep constant rate* ϵ_τ

Choosing meta parameters:

- τ *...A few 10 passes through the training examples*
- ϵ_τ *... about 1% of* ϵ_0
- ϵ_0...
  - *Monitor first several iterations*
  - *Choose learning rate higher than the best performing one*
  - *But not so high that it causes severe instability*

2.3.2.3.2 Adam

Building on top of the stochastic gradient descent algorithms, the Adam or Adaptive moment estimation [46] optimization algorithm performs an estimation of both the first and second moments of the gradients in order to compute individual adaptive learning rates for different parameters
The authors describe Adam as combining the advantages of two other extensions of stochastic gradient descent. Specifically:

1. **Adaptive Gradient Algorithm (AdaGrad):** This algorithm is based on a per-parameter learning rate approach that leads to improvements in model performance when facing problems with sparse gradients, such as natural language and computer vision problems. In fact, when the scaling of the weights is unequal, this algorithm proves better than SGD since it converges both faster and more reliably.

2. **Root Mean Square Propagation (RMSProp):** based on computing the average of the magnitudes of recent gradients for a specific weight, to later on divide the learning rate of that weight by its corresponding average. This algorithm is usually used for online and non-stationary problems.

While in RMSProp only the average of the first moments are used in order to adapt the parameter learning rates, in Adam one makes use of the average of both the first (mean) and second (uncentered variance) moments of the gradients.
Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters ϵ_1 and ϵ_2 control the decay rates of these moving averages.
When facing non-convex optimization problems, Adam counts with a wide array of benefits. The algorithm shows computational efficiency, the memory requirements for such method are small, little tuning is required for the hyper-parameters, which one can as well interpret intuitively, the algorithm proves a great tool for the resolution of large size dataset/parameter problems, etc.
Through an estimation of first and second moment of the gradients, the algorithm is capable of computing individual adaptive learning rates for different parameters, while remaining computationally efficient. The algorithm's formulation is as follows :

```
    Required:
    • $\alpha$: step size
    • $\epsilon_1,\ \epsilon_2 \in [0,1)$: exponential decay rates for the moment estimates
    • $f_t(\theta)$: stochastic objective function with parameters $\theta$
    • $\theta_0$: initial parameter vector.
Initialize:
    • $m_0 \longleftarrow 0$: $1^{st}$ moment vector
    • $v_0 \longleftarrow 0$: $2^{nd}$ moment vector
    • $t \longleftarrow 0$: timestep
while $\theta_t$ not converged do
    • $t \longleftarrow t+1$
    • $g_t \longleftarrow \bigtriangledown_\theta f_t(\theta_{t-1})$: get gradient w.r.t stochastic objective at timestep $t$
    • $m_t \longleftarrow \epsilon_1 m_{t-1} + (1-\epsilon_1) g_t$: update biased first moment estimate
    • $v_t \longleftarrow \epsilon_2 v_{t-1} + (1-\epsilon_2) g_t^2$: update biased second raw moment estimate
    • $\hat{m_t} \longleftarrow m_t/(1-\epsilon_1^t)$: compute bias-corrected first moment estimate
    • $\hat{v_t} \longleftarrow v_t/(1-\epsilon_2^t)$: compute bias-corrected second raw moment estimate
    • $\theta_t \longleftarrow \theta_{t-1} - \alpha \hat{m_t}/(\sqrt{\hat{v_t}} + \epsilon)$: update parameters
end while
return $\theta_t$: resulting parameters
```

2.3.2.3.3 Rectified Adam

In the same manner that Adam was developed using stochastic gradient descent as the building base, so does RAdam emerge. This rectification of the Adam optimizer achieves better precision, generalization, and fewer number of epochs during training.
Converging to local optima is one of the many challenges faced in Deep Learning when making use of adaptive learning rates, and in order to battle such obstacle researchers have resorted to a 'heuristic warmup', that is, the use of a small learning rate during the first epochs of training in the hopes of mitigating such problem. However, to determine the settings of such warm-up phase constitutes a tedious and time-consuming task, since each application demands different settings, which forced the operator to a trial-and-error approach.
In [45], authors claim to have found the root cause of such convergence issue through an empirical and theoretical analysis. The origin of such problem seems to lie in the fact that, given the limited amount of training samples being used in the early stage of model training, the adaptive learning rate experiences undesirably large variance. Thus, through the use of smaller learning rates in the first few epochs of training one should be able to reduce such variance, consequently optimizing the warmup heuristic phase.
These findings led Liu et al. [45] to rectify the Adam optimizer, and this explicit rectification of the variance of the adaptive learning rate produced as a result an algorithm that surpassed Adam in regards to accuracy, generalization, and the number of epochs employed during training. This new proposal received the name of Rectified Adam (RAdam).
The fact that adaptive learning rate optimizers have difficulties generalizing during the first batch updates during training, as well as the undesirably high variance of the learning rate in these stages, were the pivoting points used by the authors to develop their method. They implemented two changes: first, training was initialized with a small learning rate, and second, the momentum term for the first few input train batches was turned off. By doing this, the aforementioned issues could be actually rectified.
The authors evaluated their experiments on language modeling, image classification, and neural machine translation and found that their Rectified Adam implementation brings consistent improvement over the vanilla Adam.

Its formula follows:

```
Input: $\{\alpha_t\}_{t=1}^{T}$: step size, $\{\epsilon_1, \epsilon_2\}$: decay rate to calculate moving average and moving $2_{nd}$ moment, $\theta_0$: initial parameter, $f_t(\theta)$: stochastic objective function

Output: $\theta_t$: resulting parameters

Initialize:
    • $m_0, v_0 \longleftarrow 0, 0$: moving $1_{st}$ and $2_{nd}$moment
    • $\rho_\infty \longleftarrow 2/(1-\epsilon_2) - 1$: compute the maximum length of the approximated SMA
while $\theta_t$ not converged do
    • $t \longleftarrow t + 1$
    • $g_t \longleftarrow \bigtriangledown_\theta f_t(\theta_{t-1})$: get gradient w.r.t stochastic objective at timestep $t$
    • $v_t \longleftarrow \epsilon_2 v_{t-1} + (1-\epsilon_2)g_t^2$: update biased second raw moment estimate
    • $m_t \longleftarrow \epsilon_1 m_{t-1} + (1-\epsilon_1)g_t$: update biased first moment estimate
    • $\hat{m_t} \longleftarrow m_t/(1-\epsilon_1^t)$: compute bias-corrected first moment estimate
    • $\rho_t \longleftarrow \rho_\infty - 2t\theta_2^t/(1-\theta_2^t))$: compute the length of the approximated SMA
    • if the variance is tractable, i.e, $\rho_t > 4$ then:
        – $\hat{v_t} \longleftarrow \sqrt{v_t/(1-\epsilon_2^t)}$: compute bias-corrected moving $2_{nd}$ moment
        – $r_t \longleftarrow \sqrt{\frac{(\rho_t-4)(\rho_t-2)\rho_\infty}{(\rho_\infty-4)(\rho_\infty-2)\rho_t}}$: compute the variance rectification term
        – $\theta_t \longleftarrow \theta_{t-1} - \alpha_t r_t \hat{m_t}/\hat{v_t}$: update parameters with adaptive momentum
    • else
        – $\theta_t \longleftarrow \theta_{t-1} - \alpha_t \hat{m_t}$: update parameters with un-adapted momentum
end while
return $\theta_t$: resulting parameters
```

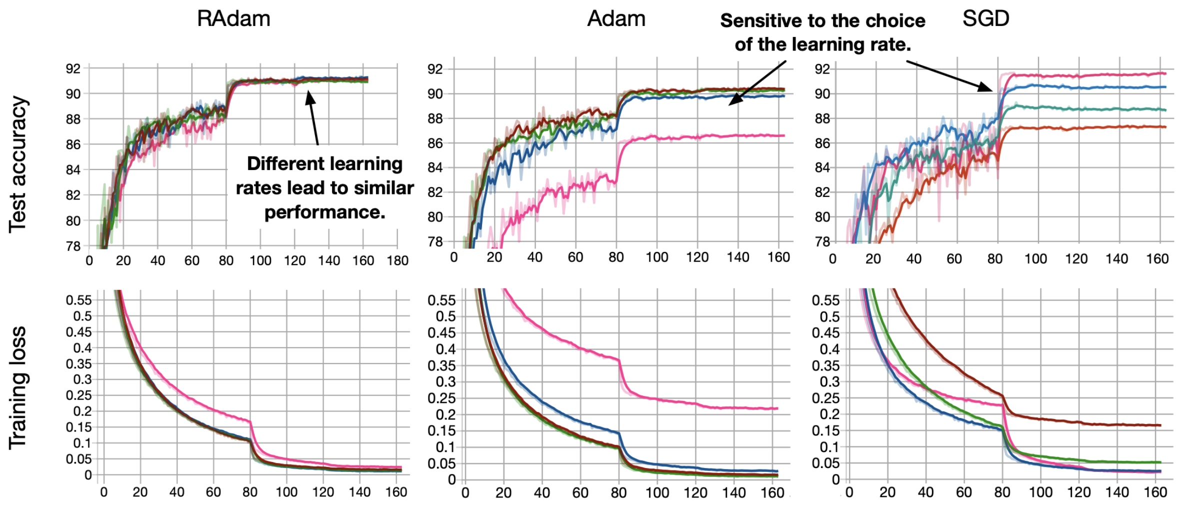


Figure 2.6: Optimization algorithms comparison [45]. As stated in the paper: performance of RAdam, Adam and SGD with different learning rates on *CIFAR10*.

In the development of this work, we'll use RAdam as our default optimizer.

2.3.3 Manufacturing Process of Integrated Circuits

The manufacture of semiconductor devices constitutes the core process of the Semiconductor industry. They usually constitute an integral part of the circuitry of chips that are currently present in all kinds of electrical devices.
Starting out with a wafer made of pure semiconducting material [51], the different electronic circuits are gradually built through a multiple-step sequence of chemical and photolithographic processes onto the surface. Silicon Carbide/Gallium Nitride are becoming the most popular materials for integrated circuits.
Due to the sensitivity of the nanoscale structure of these circuits, the manufacturing process requires the use of hermetically sealed environments in order to reduce the appearance of defects and increase the proportion of properly functional microchips in a wafer (yield). To achieve this goal, semiconductor fabrication plants tend to be highly specialized and completely automated.

2.3.3.1 Wafer processing

Wafer manufacturing facilities compose the heart of the production line for integrated circuits. Microelectronic devices require a semiconductor substrate material where they can be built in and upon, these substrates consists of thin slices of semiconductor known as wafers. One wafer contains several microcircuits, which are later on separated and packaged as integrated circuits through a process known as wafer dicing.

In order to obtain these wafers, it is first required to produce a seed crystal, a perfect single crystal from which they can be sliced. Two processes can be followed in order to obtain these seeds, Figure 2.7.

- The Czochralski process, where a seed crystal on a rotating rod is brought to the surface of the silicon melt. An immediate solidification of the seed crystal is possible due to the fact that the crucible temperature remains just slightly above the melting point. It is important to note that only when the melt enters in contact with the seed crystal, its architecture is overtaken by the melt.
  The seed grows as it is slowly pulled upwards with constant rotation, while also remaining continuously in contact with the melt. In order to ensure a steady growth of the crystal the temperature must remain constant. The drawing speed determines the diameter of the crystal, and the faster the drawing speed is, the thinner the crystal becomes. Finally, to avoid silicon oxidation, the entirety of the device must be placed under a controlled atmosphere.

- The Float-zone silicon, where only a small polysilicon area of a few millimeters is molten. Once again, a seed crystal introduced at the end of the polycrystalline silicon rod sets the crystal structure. The heated region is slowly guided along the rod, and as the polycrystalline crystal enters in contact with the seed, it overtakes the seedling's structure and slowly transforms into a single crystal.

With the passage of time, the manufacture of integrated circuits has developed considerably. Starting with a wafer diameter of only 25 mm, this size has reached up to 150-300 mm diameter today, and research is already being done in order to efficiently generate 450 mm diameter wafers, Figure 2.8. In fact, the wafer surface has increased by more than 300-fold from the tiny 1-inch original wafer, whereas the disk diameter has in comparison only increased by a factor of 18.
This increase in wafer diameter not only leads to significant increases in the production rate of microchips in the manufacturing process, but also to considerable cost reductions. Two advantages are particularly important: first, more than twice as many chips can be produced on a 300 mm wafer as on a 200 mm wafer, and second, since the wafer's edge is less curved as the diameter increases, the cut-off, due to the rectangular shape of the chips, is consequently minimized.
The cut-off step is performed due the existing conflict between the wafer and microcircuit shape. Despite the fact that microchips are rectangular, wafers are fabricated in a round shape. This translates into some areas, or blend, on the wafer where no entire chips can be placed and which has to be consequently discarded at the end of the manufacturing process.
The round wafers have several advantages over an angular shape, even though it is possible to cut the round single crystal into rectangular shape. These are:

- The quality of the silicon has proven to be compromised if we try to straighten the

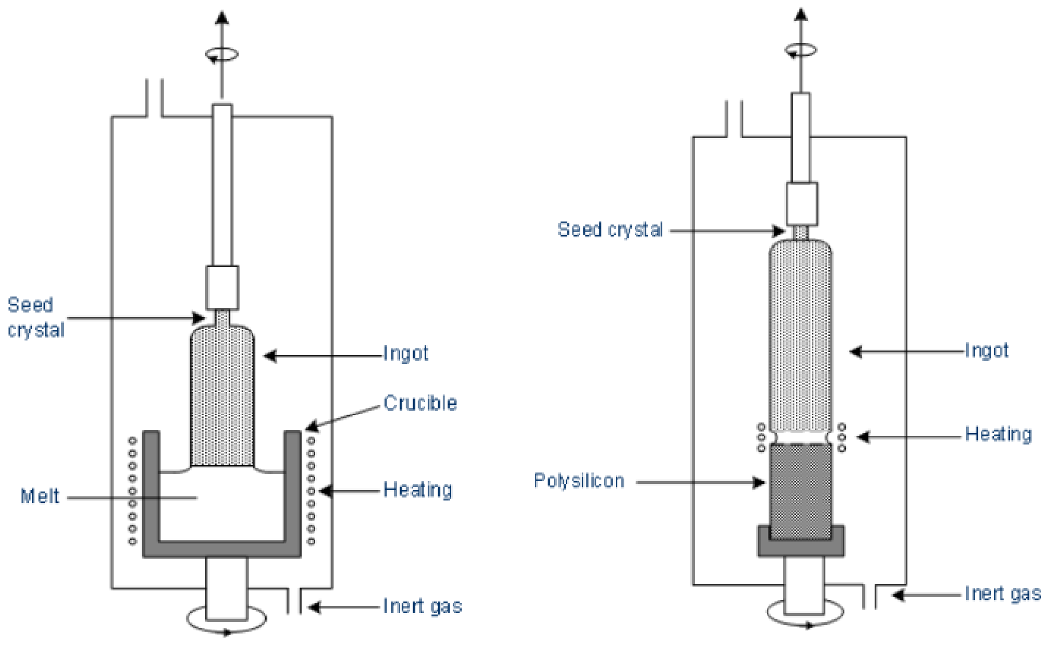


Figure 2.7: Methods of crystal growth in semiconductor manufacturing [51]**.** Left, the Czochralski method. Right, the Float-zone silicon method.

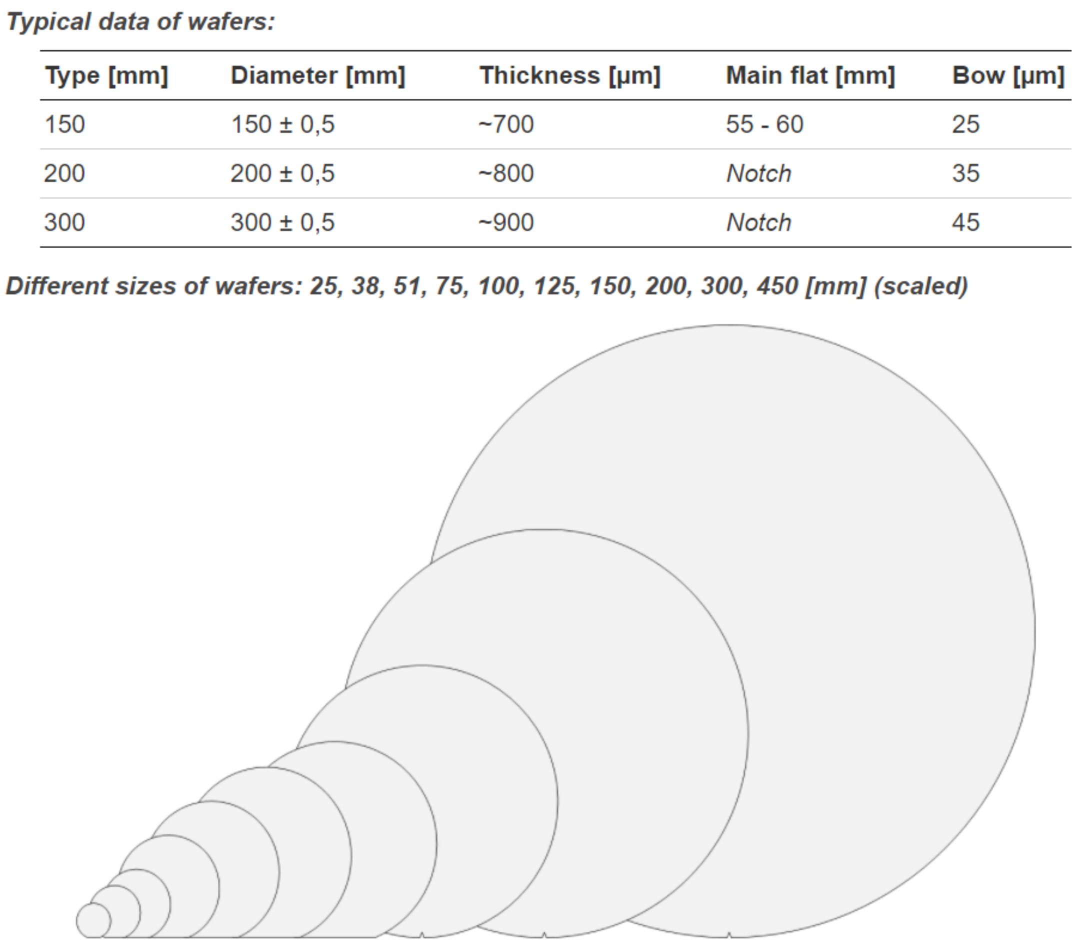
round silicon, leading to defects and dislocations. This is caused by the additional stress the crystal is subject to when applying this process.

- Round wafers have proven not only to be more stable, but also more robust during transportation, since angular wafers could hardly be transported and processed without damage.

- Radially symmetrical processes are much easier than angular ones for homogen processing during microchip manufacturing.

- Angular wafers are not without cut-off, since it is impossible to process to the outermost edge, which means the extreme edges must be discarded. In order to avoid layers spalling off of the edge, the wafer must be clamped during transport.

Since even a speck of dust is capable of ruining a microcircuit due to its nanoscale architecture, we require the use of highly specialized plants known as clean rooms, Figure 2.9, in order to control the amount of intruding agents present in the manufacturing process. Additionally, clean rooms must be warded off against other possible sources of perturbations such as static electricity, which is controlled through narrow bands of temperature and humidity, or damping the room against vibrations.



Typical data of wafers:

Type [mm]	Diameter [mm]	Thickness [µm]	Main flat [mm]	Bow [µm]
150	150 ± 0,5	~700	55 - 60	25
200	200 ± 0,5	~800	*Notch*	35
300	300 ± 0,5	~900	*Notch*	45

Different sizes of wafers: 25, 38, 51, 75, 100, 125, 150, 200, 300, 450 [mm] (scaled)

Figure 2.8: Development of the wafer size [51]**.** In the solar cell manufacturing process wafers can be poured in a rectangular form and their production is easier in comparison to microchip fabrication.

All high precision equipment required for semiconductor manufacturing can be found within the clear room, these include all machinery used for etching, doping, dicing and cleaning. In order to minimize the amount of contamination brought by the operators to the room, these are required to wear clean room suit, also known as bunny suits, that cover the totality of their body, and must step into air showers to blow everything clear before entering the actual room. As explained above, the running trend of producing larger diameter wafers has led to a significant increase in the amount of chips produced at once in the production line.
The structure of a fabrication plant often follows the following design:

- The ground floor: which contains all electrical equipment.

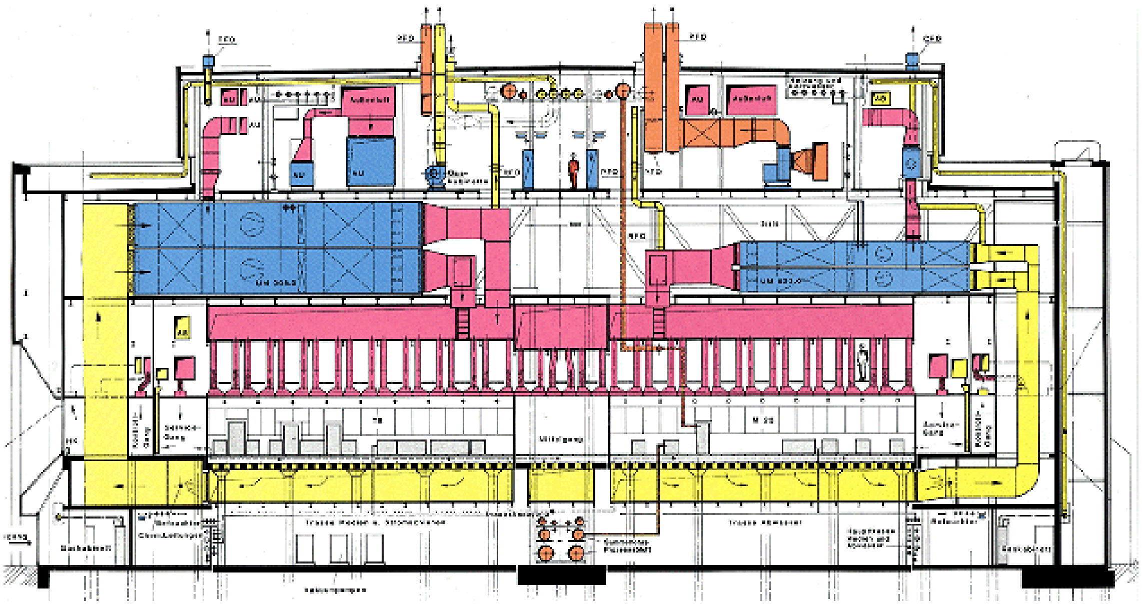


Figure 2.9: Illustration of a clean room [51]. The production line is only the region between the light-red colored and the yellow colored area. At the bottom one can find the basement, equipped with supply units, and on the top the air filters.

- The clean sub-fabrication plant: that contains chemical delivery, purification, and destruction systems.
- The clean room: which can be composed of several floors.
- The roof: composed of air handling equipment that draws, purifies and cools the outside air.

2.3.3.2 Defect Detection in production line

Despite the quality of the facility and its equipment, wafer defects remain unavoidable, and surface inspection tools are required for evaluation. Visual inspection is one the most important steps in semiconductor manufacturing, since it constitutes a source of information relevant for the identification and correction of problems encountered in the production line, consequently improving the quality of the product.
Defects can be divided into two categories according to their cause: random defects, mainly caused by the class 200 defects and that become attached to the surface, and systematic defects, caused by the exposure process as well as the conditions of the mask. The most common wafer defects are small particles and pattern defects. There's a wide variety of defect classes, Figure 2.10, and some appear more often than others. This constitutes a source of data imbalances in machine learning.
Additionally, while some defects are critical to the product, and if present imply the removal of the wafer from the production line, others do not pose a real threat to the

circuits and can be tolerated up to a certain threshold. This further emphasises the importance of defect classification during wafer inspection.

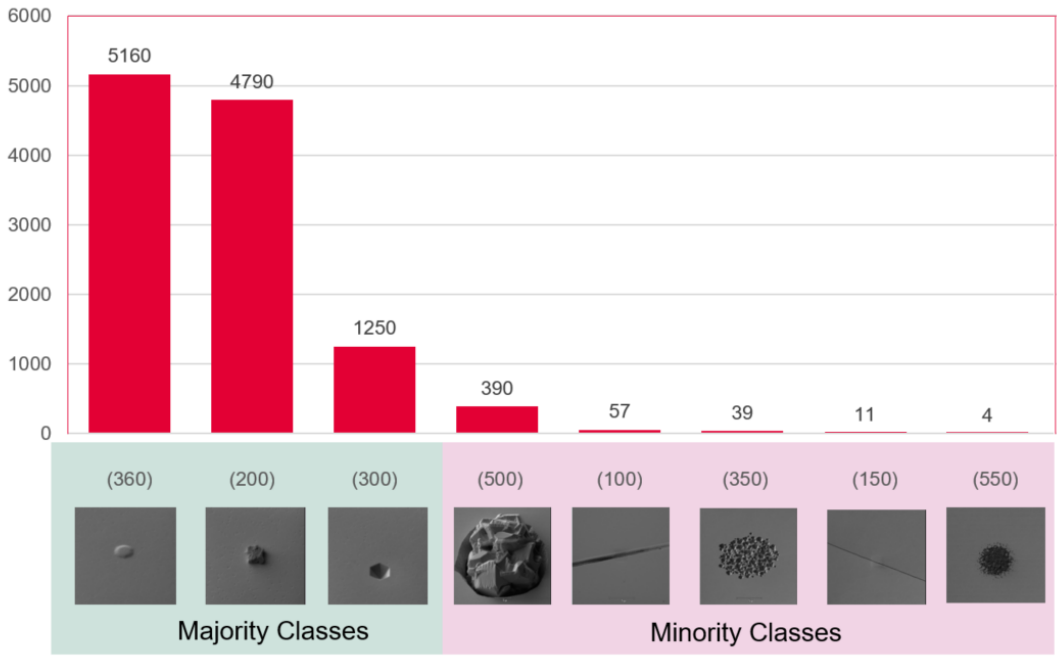


Figure 2.10: Illustration of early wafer defects distribution. Long tail distribution of some of the defects encountered during wafer inspection. Green box contains all those classes with over 1000 samples (majority classes) and light-red box contains those with less than 1000 samples (minority classes).

In this work, we'll focus on the classification of 10 different defects, plus a No-Defect class, Figure 3.6. It is important to note that some classes will be implemented in later stages of the work, see Section 3.1.4.

2.3.3.2.1 Scanning Electron Microscope

The Scanning Electron Microscope (SEM) is a kind of microscope that uses an electron beam as illumination source in order to scan the surface of the sample at hand, in our case the wafer's, and generate images of its topography. The interaction between the electrons of the focused beam and the materials from the sample can produce several kinds of signals that contain different types of information that relate to the sample's topography and composition, and can be processed to generate images that contain specific features from it. Due to the high sensitivity of this device, it is one of the most used technologies for visual inspection in the manufacturing process.
The SEM consists of the following main components: the electron gun, a Tungsten filament

heated up to 24000^{o}C that emits the electrons, an anode, responsible for accelerating the electrons, magnetic lenses, which focus the electrons in a narrow beam, as well as the amount that reaches the sample, scanning coils, which allow the electron beam to move row by row across the sample surface, and the objective lenses, that regularize the focus of the lenses. See Figure 2.12.
The sample is placed upon a holder and fixed in the microscope. All measurements must be done in vacuum in order to ensure a clean surface as well as no interaction between electrons and gases. In this work, datasets consists exclusively of secondary electron images.
The surface point bombarded by the electrons will generate 3 different kinds of signals, which will be detected by three different channels respectively. These signals are composed of:

1. **Secondary electrons:** characterized by having low energy, these electrons are 'kicked out' of the sample by inelastic scattering interactions with beam electrons and can only escape near the surface. They are detected by the collector and due to their low energy, they can only originate from within a few nanometers below the sample surface.

   - The brightness of the signal depends on the number of secondary electrons that reach the detector. For this detector, edges and steep surfaces will appear brighter than other flat structures, for example, protrusions will appear bright in contrast to holes and cracks, which will appear dark. This is caused by the angle of incidence of the beam and its interaction with the sample, that is, as the angle of incidence increases, the interaction volume increases and so does the number of expelled secondary electrons.

2. **Back-scattered electrons:** these electrons managed to penetrate deeper the sample and were 'bounced back' in a similar direction. They originate in the electron beam and their interactions with the sample's materials, known as elastic scattering, reflects/back-scatters them out of the specimen.

   - Number of back-scattered electrons depends on the atomic number of the elements of the sample. Therefore, images will appear brighter for high atomic number elements, while dark for low atomic number elements. These electrons can also be used in order to determine the crystallography of the specimen by means of an electron back-scatter diffraction (EBSD) image.

3. **Characteristic X-rays:** primary beam electrons 'kick out' an inner shell electron from the sample's atom, which consequently leads to an outer shell electron occupying the available spot, emitting characteristic radiation for sample composition measurement.

- Analysis of the x-ray signals are usually used in order to determine the number of elements that constitute the sample, as well as the distribution of these.

While we have at our disposal samples from all three detectors, Figure 2.11, we decided to focus this work on those taken only with the secondary electron detector, considered to represent best the specimen topography.

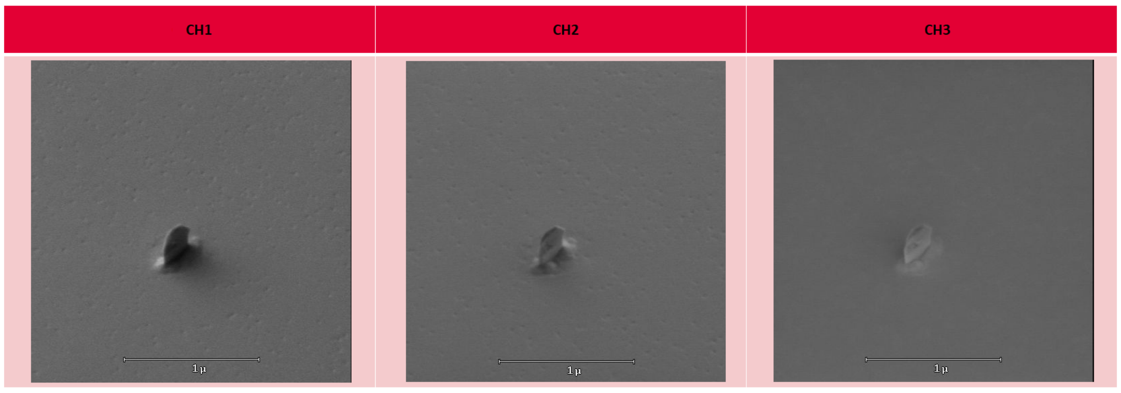


Figure 2.11: Illustration of samples from SEM channels. CH1 corresponds to the secondary electron detector, CH2 to the back-scattered electron detector, and CH3 to the X-ray detector.

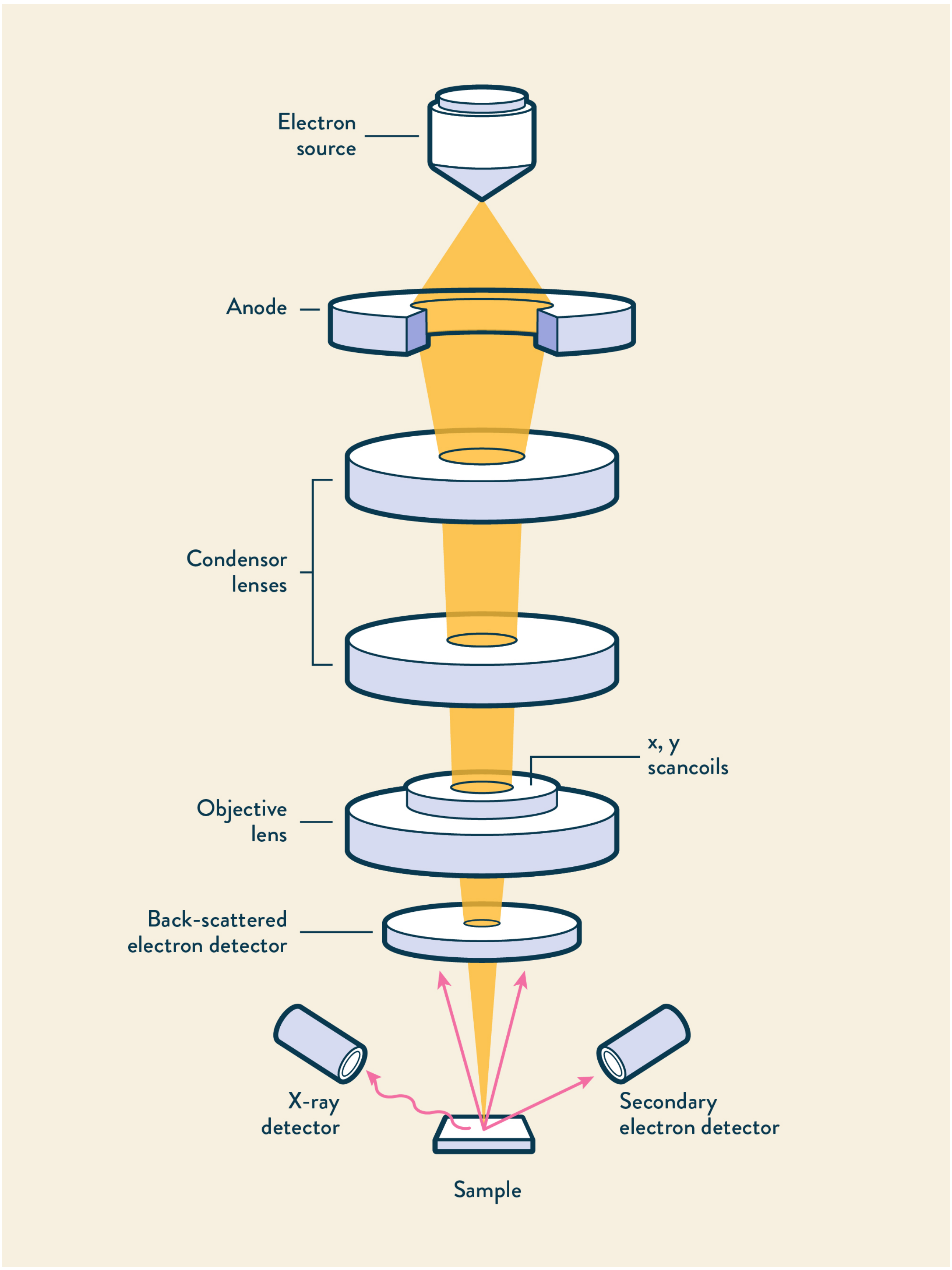


Figure 2.12: Schematic of a SEM [6]. The microscope allows a degree of magnification up to 500,000 times the original scale. In contrast, the best light microscopes provide a magnification limit of 250 times the original size.

3 Methodology

In section 3.1 we perform a theoretical introduction of the different preprocessing and regularization steps considered to improve model performance, followed by a brief explanation of the additional classes that will be implemented, as well as the techniques used for model visualization.
In section 3.2, after initial evaluation of the oversampling effects on minority class prediction, each regularization/preprocesing step will be evaluated independetly in order to determine which ones lead to performance reduction, which will be therefore not considered any further.
On the other hand, all those that led to no significant variations or to actual improvements will constitute part of our final preprocessing process, which will thus be composed of the combination of these, section 3.2.4. Afterwards, we'll proceed to increase the number of classes implemented in the model, improve the consistency of our heatmaps, and evaluate higher architectures, further explained in section 3.2.5. Finally, after addtional data cleaning, class expansion, and hyperparameter tuning, we'll reach the realization of our final models, section 3.2.6.

3.1 Experiments

3.1.1 Preprocessing

Feature scaling. Originally, our dataset's intensity values ranged between 0 and 255. On this account, we scaled down the range so that the features lie on a similar scale that can help the gradient descent converge faster towards the minimum, since all the features would contribute equally to the result. Therefore, we normalized our data.
When the distribution of your dataset differs from a Gaussian distribution, its is recommended to use Min-Max scaling (normalization). This technique shifts the values of your data in such a manner that they end up ranging between 0 and 1.

Standardization. Another scaling technique where the values are centered around the mean with unit standard deviation. The new distribution of the dataset behaves in such a way that the mean becomes zero and the resultant standard deviation equals one. Our original approach was to use standardization in combination with normalization, but after inspection of the images generated after such preprocessing, we decided to discard standardization due to its extreme effects on the images' contrast, Figure 3.1.
This is caused due the fact that, in standardization, both mean and standard deviation are sensitive to outliers, and thus does not guarantee a common numerical range for the normalized dataset. Moreover, as long as the input scores are not Gaussian distributed, this technique is not able to retain the input distribution at the output.

Label removal. The samples generated by the SEM contain by default labels which indicate the size of the defect. In order to ensure that our samples contain only crucial information, we removed these labels. Though it must be noted that, given the size and location of these labels, as long as we count with a large enough set of samples per class, no significant variation should be noted.
We formulated two hypothesis: on one hand, these labels could prove beneficial for learning, since they provide additional information about defect features. On the other hand, they could pose a source of noise for our model, which could focus on them as the actual feature, rather than the defect itself. This applies largely to the minority classes, due to their small size.

Image background. Geometrical transformations such as rotation and shearing produce images were the background is, by default, completely white. Several options [35] were evaluated to properly cover the background, and due to the features of the surface topography surrounding the defect, reflecting the original sample proved the fittest alternative, Figure 3.3.

Reduction to grayscale. All SEM samples are by default in RGB (Red-Green-Blue) format, but it's clear defect images consist only of black and white features, Figure 3.1. Therefore, we reduced our dataset to grayscale, Figure 3.4.
Nonetheless, it's worth mentioning that all models evaluated in this paper were built with the RGB architecture in mind. The fact that grayscale reduction may lead to relevant information loss remains to be seen.

Geometrical Augmentation. For the sake of model generalization improvement, a set of simple transformations was applied randomly to our dataset [33],[36], among these operations we count with rotation, shearing and solarizing. Though solarizing, inversion of pixel values, is not considered a geometrical transformation, it was included in this stage in order to further improve generalization.
There are three main differences between our algorithms for Random Augmentation

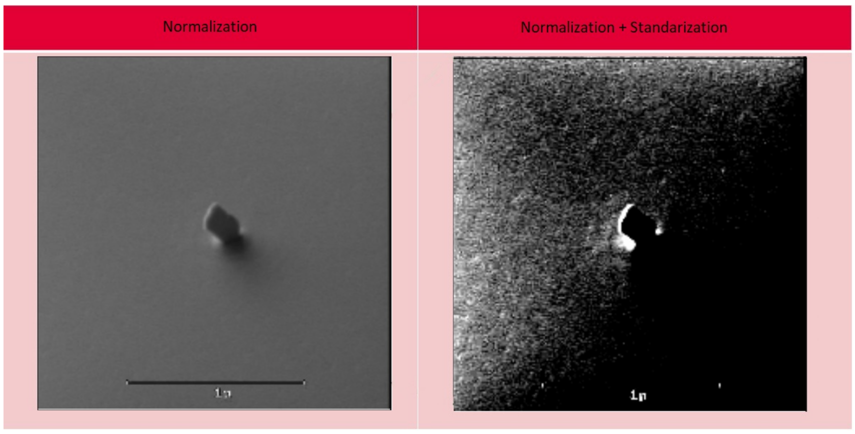


Figure 3.1: Illustration of image scaling techniques. Normalization of sample (left) led to no changes in defect features, while combination of normalization and standardization (right) darkened considerably the image.

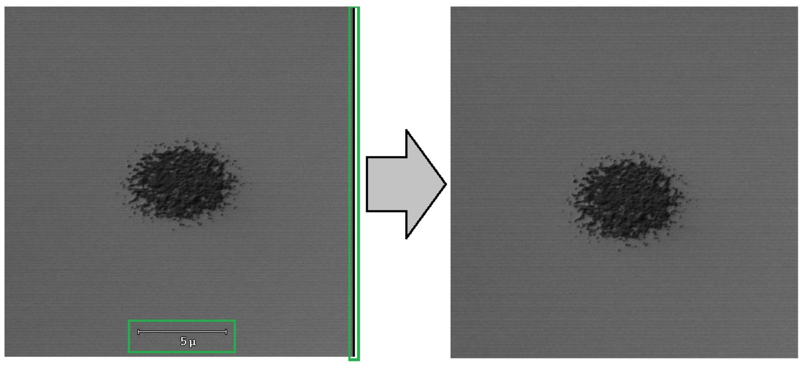


Figure 3.2: Example of label and right border cropping. Right border black line and bottom label cause conflict during preprocessing, see Section 3.1.2. After cropping, all images were resized to their original shape (480x480).

(RA) and Geometrical augmentation (GA). First, GA applies only one transformation per sample, while RA applies one pair instead, second, the magnitude of these transformation is larger in RA, and third, RA counts with additional transformations not available in GA, Figure 3.5. This is done to guarantee that artificial samples differ as much as possible from the originals, while also allowing some level of base generalization.

3.1.2 Data Augmentation

Oversampling. One of the most reasonable approaches in order to compensate data imbalances. This procedure is based on the creation of artificial samples by applying basic image transformations such as rotation, shearing, cropping, noising, blurring, etc. to the original dataset.

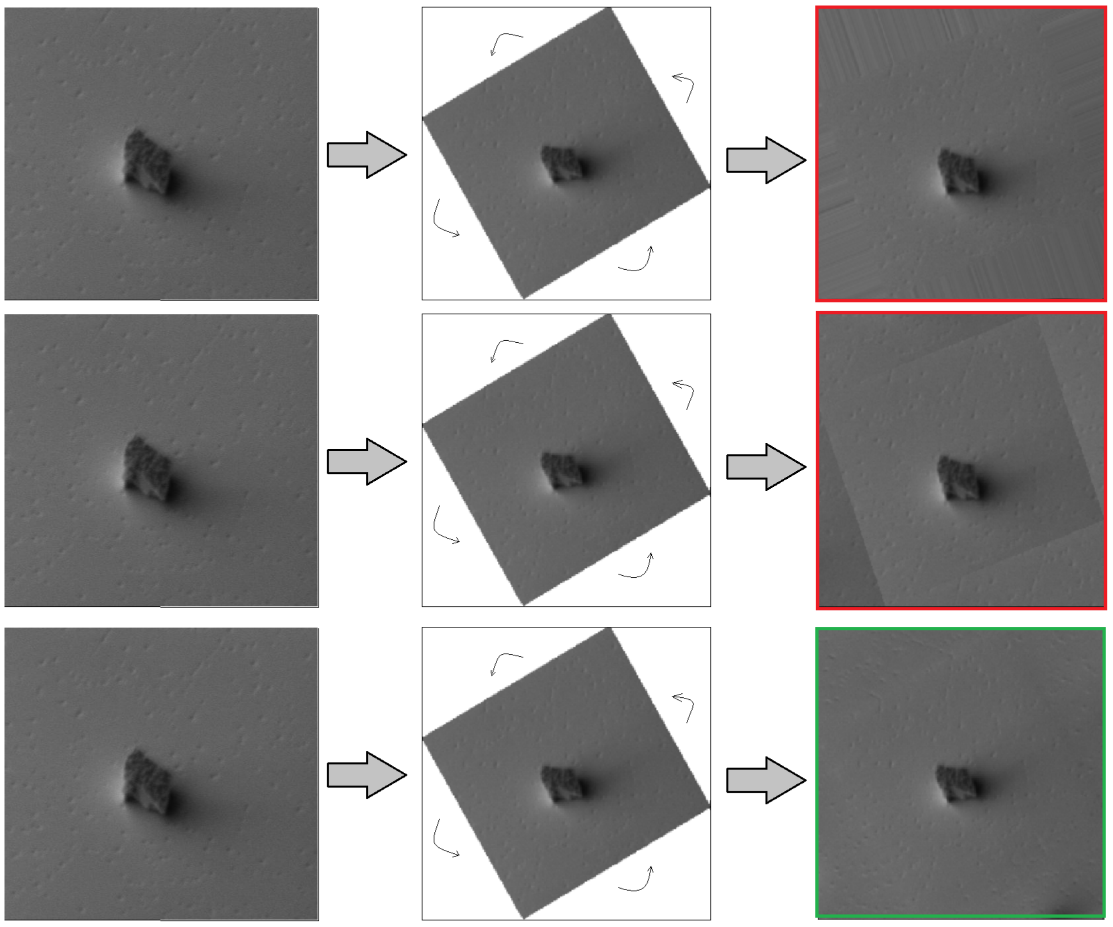


Figure 3.3: Illustration of image borders/padding for geometrical transformations. From top to bottom: border replication, border wrapping and border reflection. Reflection generated the best results.

In this work, we used EfficientNet Auto Augmentation algorithm [36] as a reference for the development of our own data augmentation space. It is important to note that we do automatically search for improved data augmentation policies [36],[33], but directly defined the operations we wish to apply and investigate its effects on a trial-an-error setting.
The following operations were considered for augmentation: rotation, shearing, cropping, flipping, noising, blurring, solarizing and contrasting, Figure 3.5. In accordance to [36], we designed a augmentation space where a policy consists of many sub-policies, one of which is randomly chosen for each image. A sub-policy consists of two operations, with customizable probabilities and magnitudes with which the functions are applied.

Undersampling. A simple technique based on the random removal of samples from

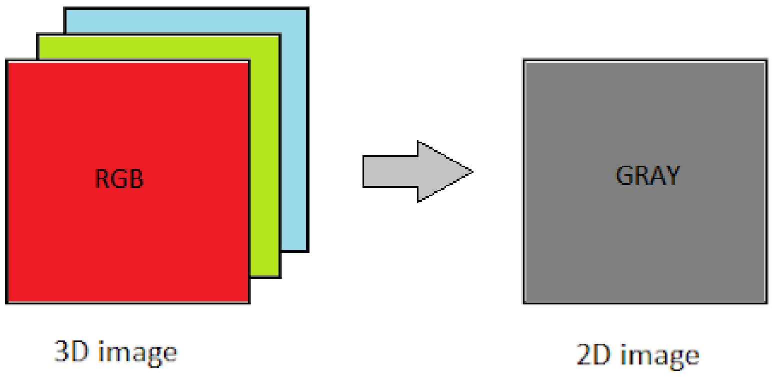


Figure 3.4: Grayscale reduction. RGB images from the SEM actually consist of the same grayscale image copied three times in a row. This process is done by default by the SEM.

the majority classes. Though data imbalances are compensated by such method, its main pitfall is the loss of information (samples) from the majority classes in order to facilitate minority learning.
Our main interest lies on oversampling as our sole data augmentation technique. However, both of these procedures were tested independently, as well as combined, to investigate the variability of minority class prediction.

3.1.3 Regularization

Class Balanced Loss. When facing a long-tailed dataset, Figure 2.10, balancing the loss is counted among the procedures most commonly used for data balancing. Weight loss is a regularization technique based on re-weighting the losses of the different classes based on the number of samples available for each class. For our cases, the weights are set to the inverse of the number of images per class.
Re-sampling or re-weighting based on the number of images per class are some of the currently existing re-balancing strategies followed for imbalanced datasets. However, as stated in [10], in these strategies the additional benefit of a newly added image will decrease as the number of samples increases.

Label smoothing. The use of one-hot encoded labels during model training gives rise to the appearance of large logit gaps in the softmax function from the output layer. Combining this with the bounded gradient leads to the generation of a model that is not only too confident about its predictions, but also less adaptive.
This overconfidence will lead to predictions where the probabilities are consistently higher than the accuracy. For example, for inputs with an actual accuracy of 75%, it may generate predictions of 95%. This indicates that the model is not properly calibrated.

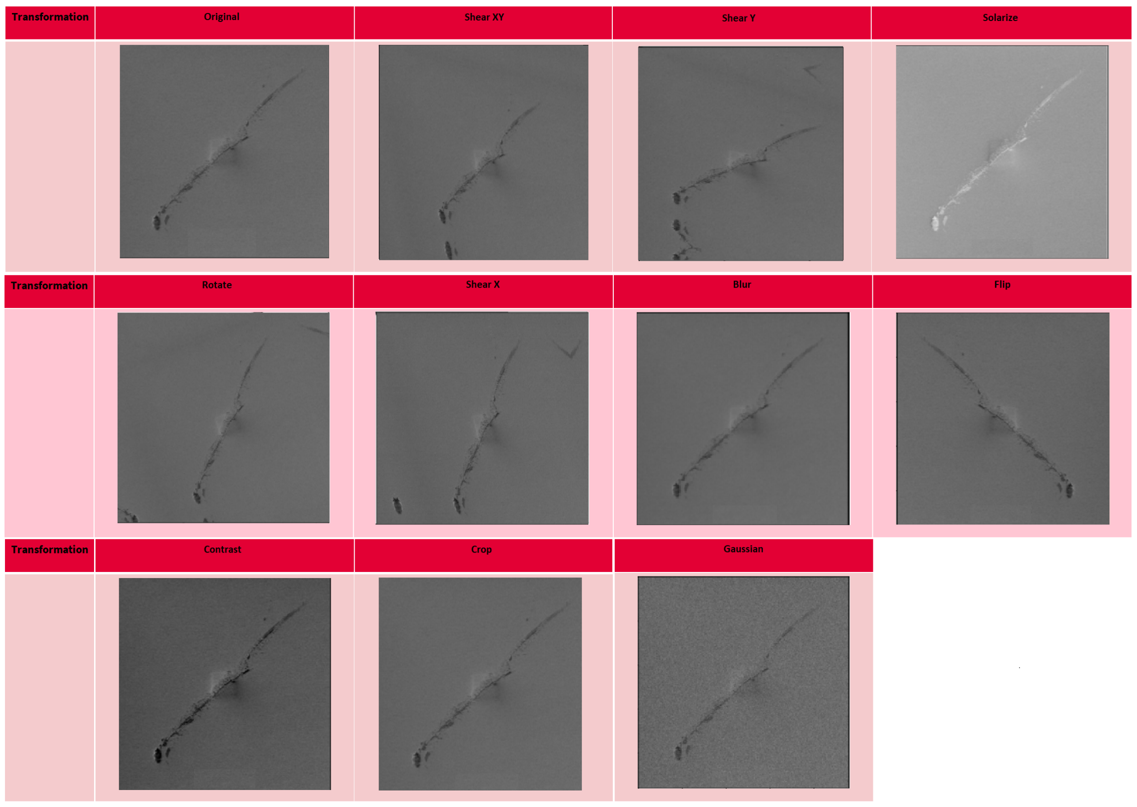


Figure 3.5: Table of image transformations. Border reflection leads to duplication of the defect for those samples where the defect is close to the border. Wether this has or not a negative effect on model performance remains to be seen.

Label smoothing [37] mixes the one-hot encoded target vector, y_{hot}, with the uniform distribution, effectively replacing the original target label. This modification will encourage the appearance of small logit gaps, preventing model overconfidence and ensuring proper calibration.

The formula for label smoothing is:

$$y_{ls} = (1 - \alpha)y_{hot} + \alpha/K$$

Where α is computed as follows:

$$\alpha = N_{classes}(1 - soft_{th})/(N_{classes} - 1) \tag{3.1}$$

where K is the number of known classes, α the degree of smoothing, and $soft_{th}$ the minimum confidence our model's prediction must assign to one of the label classes. If $\alpha = 0$, we obtain the original one-hot encoded y_{hot}. If $\alpha = 1$ we get the uniform distribution.

3.1.4 Class expansion.

Even though our dataset originally counted with only eight different defect classes, Figure 2.10, our end goal was the successful classification of eleven different classes, 2 additional defects and a No-Defect class.
Therefore, after initial training, two additional classes were included in the dataset, class 575 and 301. 575 consists of particles whose main feature lies in their sharp shape. Class 301 is similar to 360, but as an additional feature their main bump is surrounded by a 'staircase' structure, Figure 3.22.
Finally, the No-Defect class only contains samples from the intact surface of the wafer. Since the generation of these samples cannot be avoided in production, we implemented them as an additional class in our models for simplicity's sake.

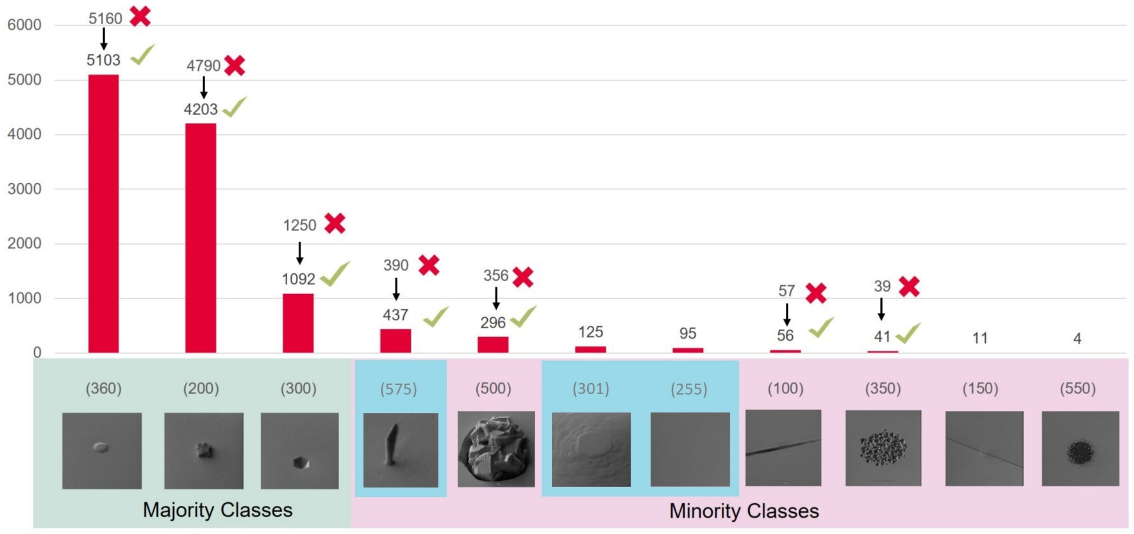


Figure 3.6: Illustration of full wafer defects distribution. Long tail distribution of the defects encountered during wafer inspection after data cleaning. The cleaning process led to changes in class size for some defects classes (check-marked). Light-green box contains all those classes with over 1000 samples (majority classes) and light-red box contains those with less than 1000 samples (minority classes). Additional minority classes contained within light-blue boxes. The No-defect class is required due to SEM shortages during sampling.

3.1.5 Visual inspection.

In order to determine if our model is properly focusing its attention on the relevant features of the image, the use of visualizing tools have proven crucial to help ensure the network is not actually cheating. Example of CNNs making wrong decision in classification include: classifying an image as a train when the actual focus of the model falls into the train tracks (since both of them appear often together), or the identification within an x-ray image of a patient of a disease that in reality is not there, since the

model does not actually focus on the appearance of such illness, but on perturbations like metallic tokens placed upon the patients' shoulder during the measurement (which would be understood as a continuous feature by the network), etc.
Thus, we would like to highlight the regions in an image that the CNN focuses on in order to ensure the model is learning the actual features we are interested in, and to estimate its robustness to new examples.

Class Activation Mapping (CAM) or Heatmaps. In order to generate these kind of images, we must ensure and take advantage of the following structures from within the CNN architecture: the last convolutional layer, from which the heatmap will be computed, and its subsequent Global Average Pooling layer (GAP), responsible for averaging all values within the feature map, effectively turning them into a single number. For example, if we had K feature maps, we would end up with K numbers after Global Average Pooling.
The different output classes are therefore estimated by multiplying each of these numbers by their corresponding weights, Figure 3.7. Class Activation Mapping follows a similar logic, instead of multiplying each weight by their corresponding GAP averages, it multiplies them by their corresponding feature maps obtained from the last convolutional layer. The resulting grid of numbers is what we actually refer to as heatmap.

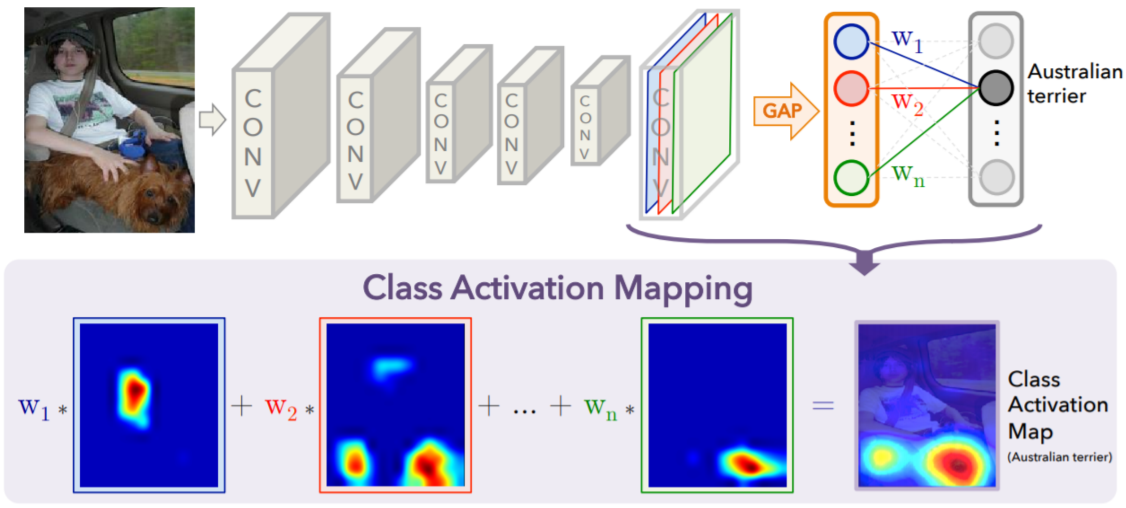


Figure 3.7: Example of Class Activation Mapping generation [7]. One needs to select the class of interest, e.g. Australian terrier, in order to compute its corresponding heatmap.

Saliency maps. This technique makes use of the most alluring regions within the image, which are estimated by evaluating its unique features such as pixels, resolution, etc. ('saliency' in visual processing) in order to compute a topographical representation that processes this data in such a manner that a clear differentiation of the different

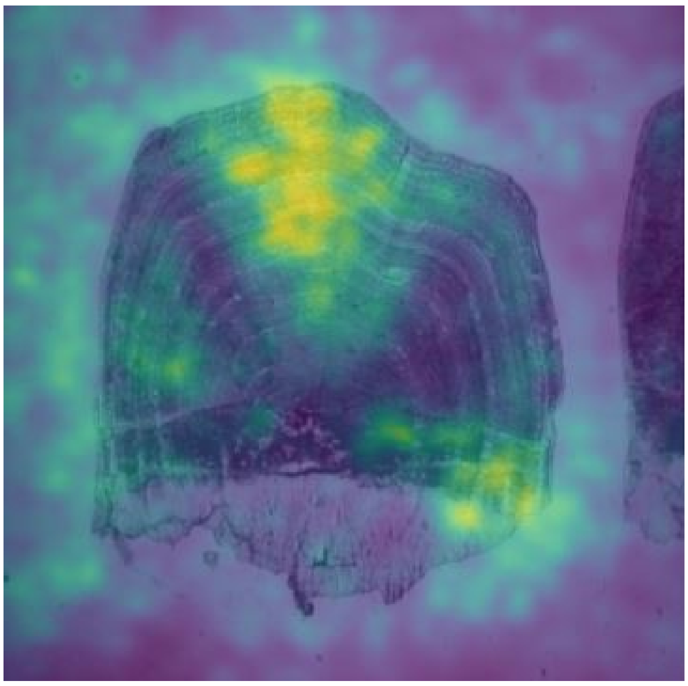


Figure 3.8: Example of a raw saliency map [8]. Pixels with a high gradient show up in yellow versus those with a low gradient in blue.

features is possible.

By applying refine adjustments to the values of the pixels across the whole input image, we are capable of estimating the relative importance each one of them has in the ultimate prediction of the network. In short, saliency maps plot the gradient of the predicted output respect to the input's pixels, Figure 3.8.
In this work, we produce our heatmaps as described in [22], where it's stated that, despite the fact that the class score $S_c(I)$ is a highly non-linear function of the input image in deep CNNs, it is possible to approximate $S_c(I)$ with a linear function close to I_0 by computing the first-order Taylor expansion:

$$S_C(I) \approx w_c^T I + b_c$$

where b is the model bias and w is the derivative of S_c with respect to the image I at the point (image) I_0:

$$w = \frac{\delta S_c}{\delta I} |_{I_0} \tag{3.2}$$

Given an image I_0 (with m rows and n columns) and a class c, the class saliency map $M \in R_{m \times n}$ is computed as follows for RGB images. First, the derivative of w (3.2) is computed by back-propagation. Afterwards, a rearrangement of the elements of w will yield the saliency map. In order to derive a single class saliency value for each pixel (i, j), [22],[11] takes the maximum magnitude of w across all colour channels: $M_{ij} = max_c|w_{h(i,j,c)}|$.
It is important to note that, to produce a saliency map, it is required but a single back-propagation pass, and since these are extracted using a CNN trained on a labeled dataset (supervised learning) no additional annotation is required.

3.2 Evaluation

3.2.1 Under-/Oversampling.

As stated in Section 3.1.2, RA consists of a search space where a policy is composed of many sub-policies, one of which is randomly chosen for each image. Different sets of oversampling magnitudes (from lower to higher) were developed in order to determine the border between generalization improvement, overfitting and noise. These are Oversampling 1 (OV1), Oversampling 2 (OV2), etc.
Respect to underfitting, three different maximum sizes were imposed upon all classes. These are 100, 1000 and 5103 samples, where the last of these actually comprises all samples, that is, the maximum size of the largest majority class (360), Figure 2.10.
In those cases were oversampling was performed, 100 as maximum class size implied 600 and 128 samples for training and validation set respectively. For 1000 class size, 6000 and 852 for training and validation set respectively. Finally, for 5103 class size, 30618 and 2702 for training and validation set respectively.
On the other hand, for the reference, were the data remained imbalanced, 100 maximum class size implied 384 and 128 samples for training and validation set respectively. For 1000 class size, 2556 and 852 for training and validation set respectively. Finally, for 5103 class size, 8104 and 2702 for training and validation set respectively.
We generated a reference model, where only underfitting took place, in addition to four more models where we performed a combination of both underfitting and four different hyperparameter sets for oversampling.
As maximum class size increases, the precision and recall of the minority classes tends to decrease, while the majority classes remain almost unchanged (Figure 3.11). In fact, when analyzing the overall base accuracy of the models, size increase will lead to better accuracy (Figure 3.10), but only majority classes will actually benefit from such population growth. In those models were oversampling was applied, precision, recall and overall accuracy of minority classes improved. So far, increasing the magnitude and probability of the policies did not seem to have any negative effects on model predictability. Therefore, it would stand to reason to select the most pushing hyperparameter set, referred to as Oversampling 4 (OV4), to reinforce generalization, since it provided the best performance.
It's worth mentioning that for all models evaluated in this section, weight loss was enabled, even for those where oversampling took place, and consequently assigned a weight of 1 to all classes.

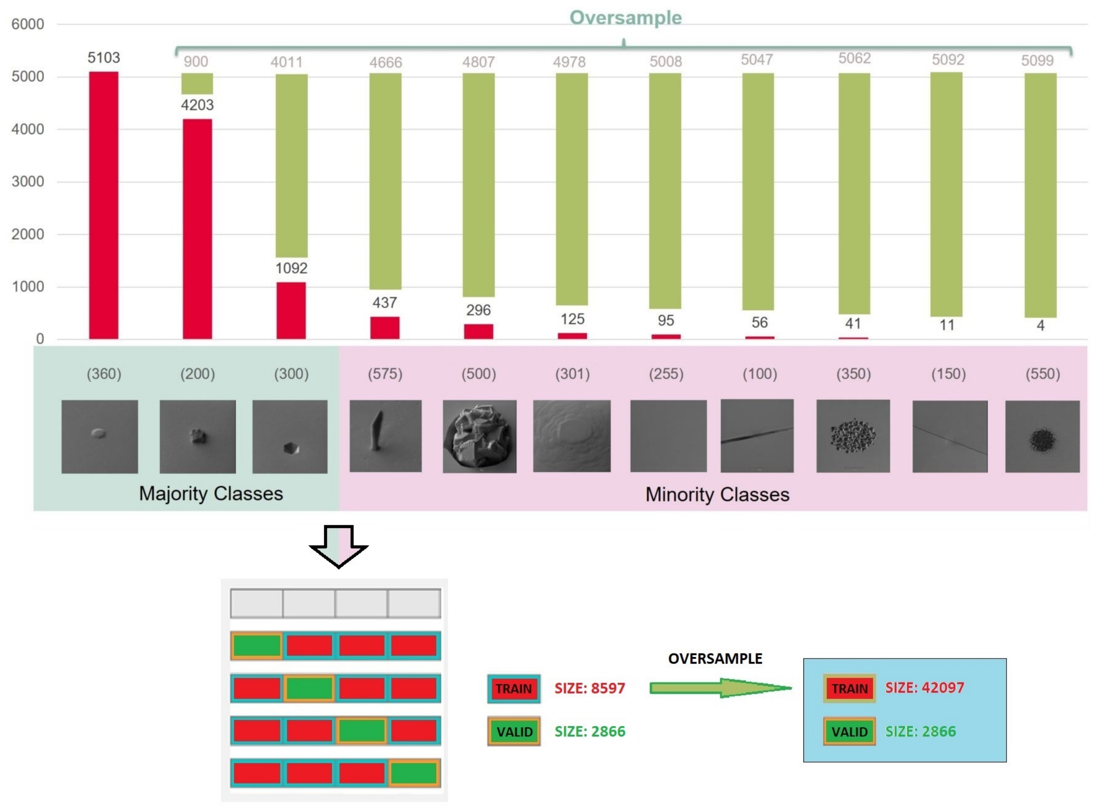


Figure 3.9: Illustration of balanced defect distribution after oversampling. Light-green box contains all those classes with over 1000 samples (majority classes) and light-red box contains those with less than 1000 samples (minority classes). After oversampling, all defect classes posses the same size as that of the largest majority class. Number of new artificial samples highlighted in green boxes. Therefore, defect dataset will be first split into 4 folds, and afterwards the training set will be oversampled so that all classes posses the same size as the maximum majority class. Validation set remains untouched.

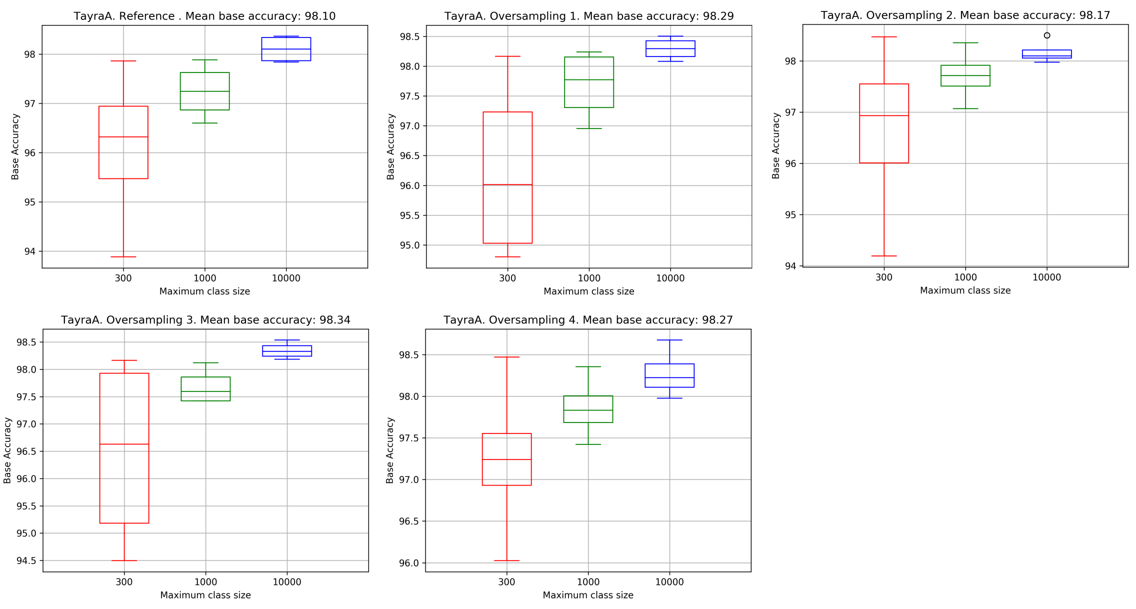


Figure 3.10: Test accuracy boxplot over class maximum size. 1-time 4-fold cross validation per hyperparameter set. Larger datasets lead to higher test accuracy. However, minority classes do not benefit from this growth, as it further accentuates imbalances.

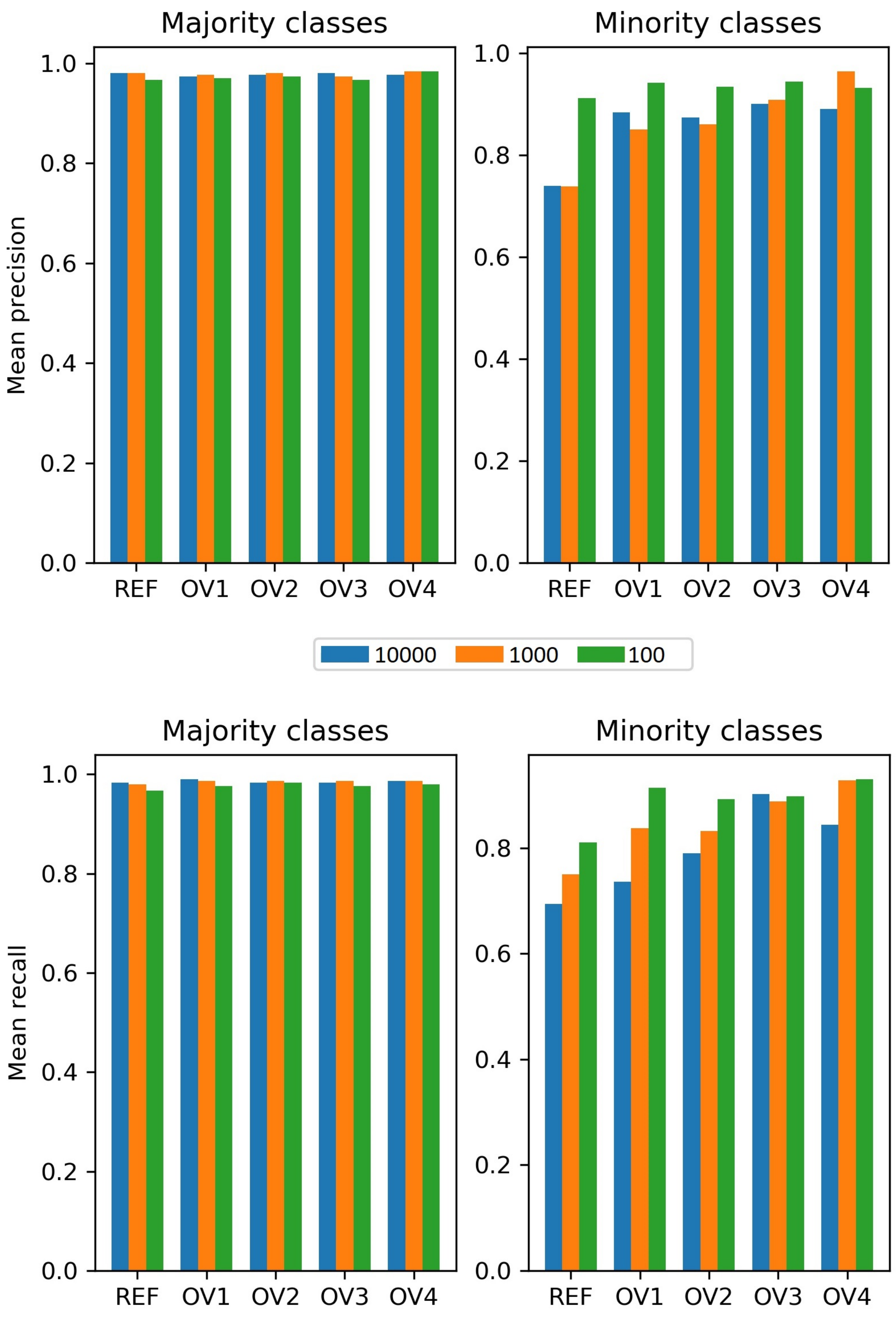


Figure 3.11: Model performance over class maximum size. 1-time 4-fold cross validation per hyperparameter set. Mean precision, first row, and recall, second row, over majority and minority classes, see Figure 3.9, per maximum class size. It is important to note that class 575, 301 and 255 have yet to be implemented. See Figures 6.1 & 6.2 for further details.

3.2.2 Hyperparameters test

The following scenarios were investigated independent from one another, in order to determine if the individual implementation/removal of the following techniques had beneficial effects on performance, to later combine all those that had it for the next stage of the experiment. In all cases, maximum class size was set to 1000, that is, for oversampling models 6000 and 852 samples, while for reference models 2556 and 852 samples for training and validation set respectively.

Weight loss removal. Class balanced loss proves a helpful tool for model regularization, but it isn't without pitfalls [10]. Its main problem lies in the majority classes, because as the number of samples increases, the additional benefit of new samples for these classes diminishes. To investigate further the effects of class balanced loss, we trained all previous models in Section 3.2.1 with balanced loss disabled.
Results indicate that weight loss removal had no significant effect on model performance, Figure 3.12, remaining fairly similar to previous runs, Figure 3.11. In fact, minority class 550 seems to benefit greatly from this change, which suggests disabling weight loss may actually have a positive effect. However, since class size was reduced, this improvement might be caused by undersampling.
We decided that, for future models, balancing the class loss would no longer be implemented, in order to avoid the pitfalls related to such technique [10].

Label removal. As stated in Section 3.1.1, we held two different hypothesis concerning the presence of size labels within defect images: either it would facilitate image classification or prove a source of noise.
Model performance remained mostly unchanged, Figure 3.13, which suggests that label inclusion did not provide any useful information for classification, nor did it introduce any significant noise. However, in order to ensure our samples remain as clean as possible, these labels were cropped out.

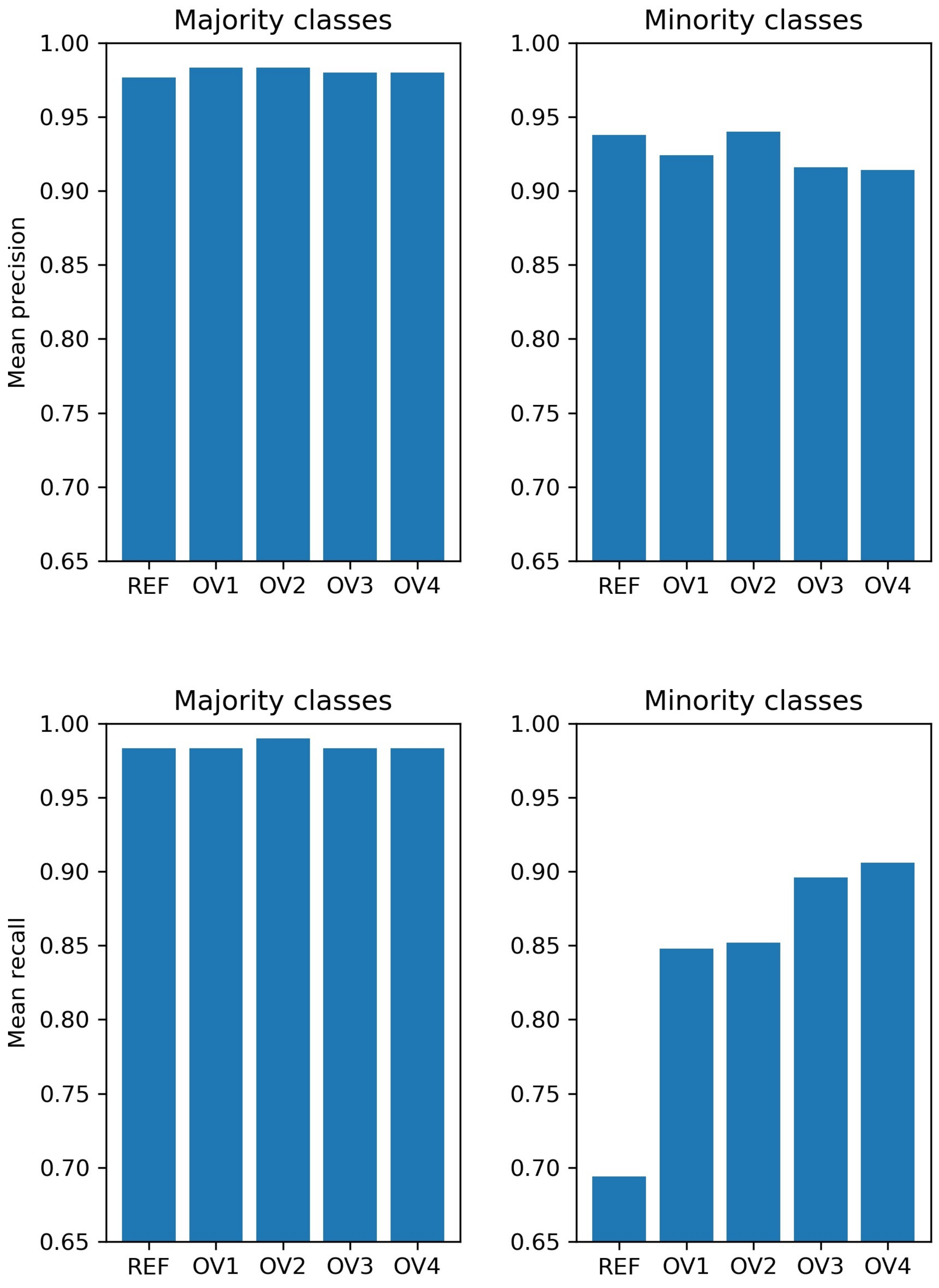


Figure 3.12: Model performance without weight loss. 1-time 4-fold cross-validation per hyperparameter set. Mean precision, first row, and recall, second row, over majority and minority classes, see Figure 3.9. It is important to note that class 575, 301 and 255 have yet to be implemented. See Figure 6.3 for further details.

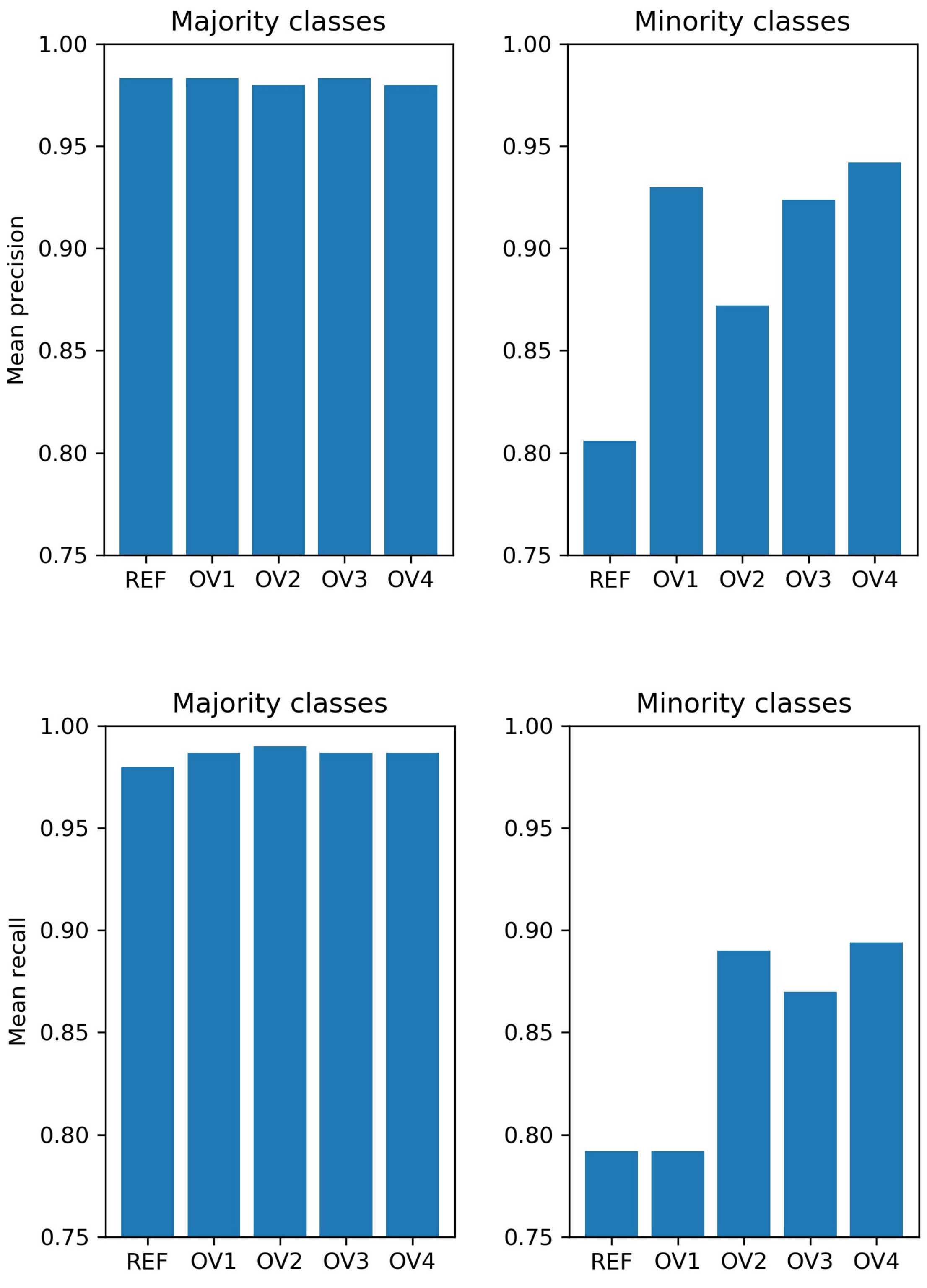


Figure 3.13: Model performance after label removal. 1-time 4-fold cross-validation per hyperparameter set. Mean precision, first row, and recall, second row, over majority and minority classes, see Figure 3.9. It is important to note that class 575, 301 and 255 have yet to be implemented. See Figure 6.4 for further details.

Background reflection. Rotation and shearing generate white background spaces that contain no relevant information, Figure 3.3. That being the case, filling these spaces with an appropriate had to be implemented to ensure proper sample format.
It is worth mentioning the presence of labels within images during this stage. This means that image reflection would lead not only to the reflection of the background, but of the label as well. Additionally, the presence of a black line border on the left size of the image would also be reflected as well. Given these two sources of irrelevant information, applying reflection alone (without cropping first) was expected to decrease performance and increase class confusion.
However, as results show (Figure 3.14), there were no significant changes in model performance, matching previous results. Therefore, for future stages, where cropping will be applied first as part of preprocessing, reflection was enabled for background in geometrical transformations.

Reduction to grayscale. Despite the RGB format of the SEM samples we work with, only one channel is actually required in order to represent all defect features. In fact, even though the microscope generates images in RGB format, these are in truth composed of the same channel cloned three times in a row (Figure images). Therefore, it stands to reason to reduce our dataset to grayscale.
Since all architectures we worked so far have been developed for three channels (RGB), this reduction must be followed by decreasing the dimensionality of the convolutional layers of the models tested down to one channel as well.
The results suggest, Figure 3.15, a loss of information that leads to consistent significant reductions in minority class recall, up to over 10% compared to previous cases (Figure 3.12, 3.13, 3.14). As expected, majority classes' prediction remains essentially unchanged. However, for the minority classes, especially 100 and 150, see Appendix, both precision and recall start to decrease. It stands to reason then, to discard grayscale from the preprocessing stage.
However, despite its negative effects on performance, we decided to implement it as well for future models in the hopes that, by combining it with all previous changes, this loss could be compensated.

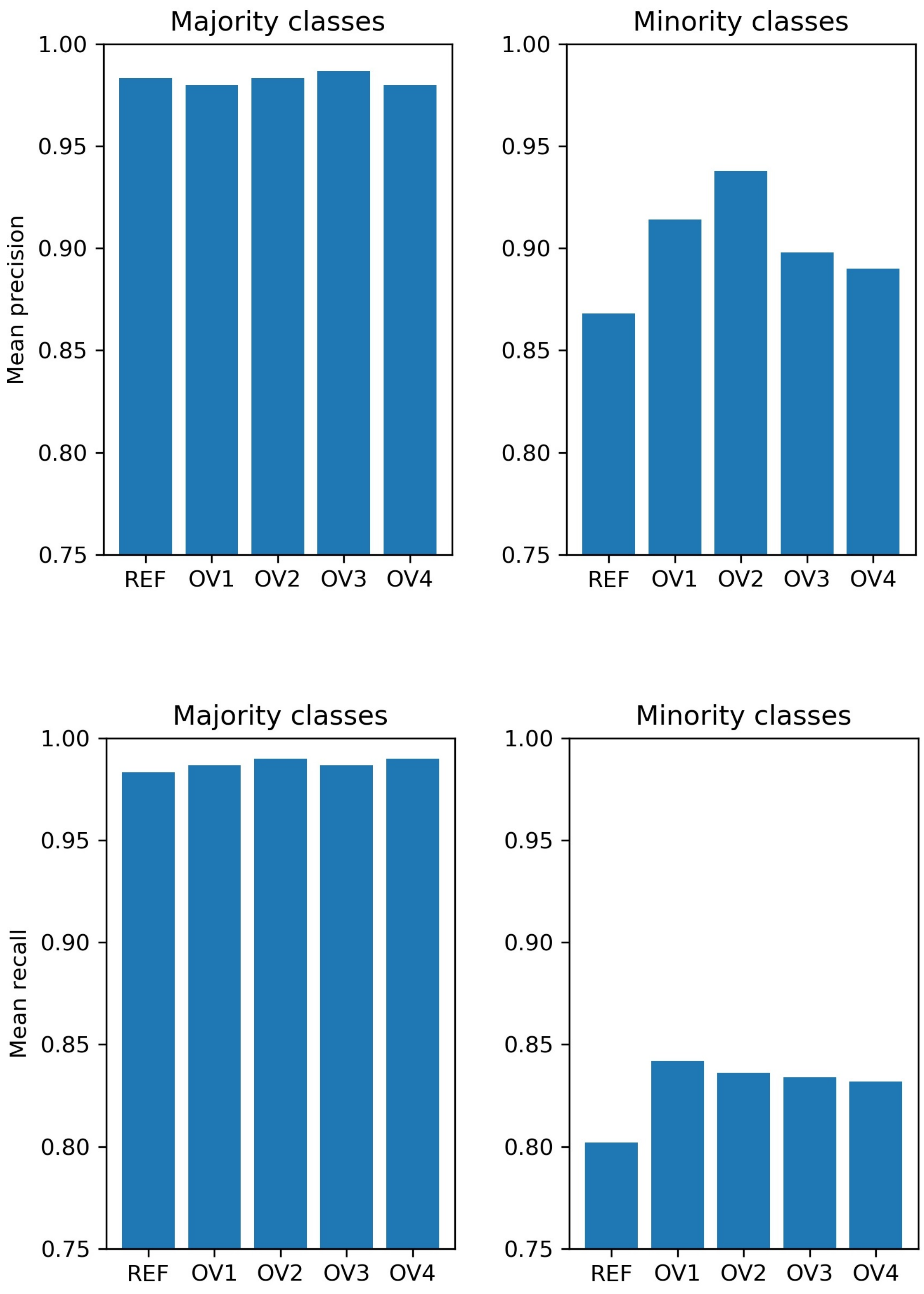


Figure 3.14: Model performance with background reflection. 1-time 4-fold cross-validation per hyperparameter set. Mean precision, first row, and recall, second row, over majority and minority classes, see Figure 3.9. It is important to note that class 575, 301 and 255 have yet to be implemented. See Figure 6.5 for further details.

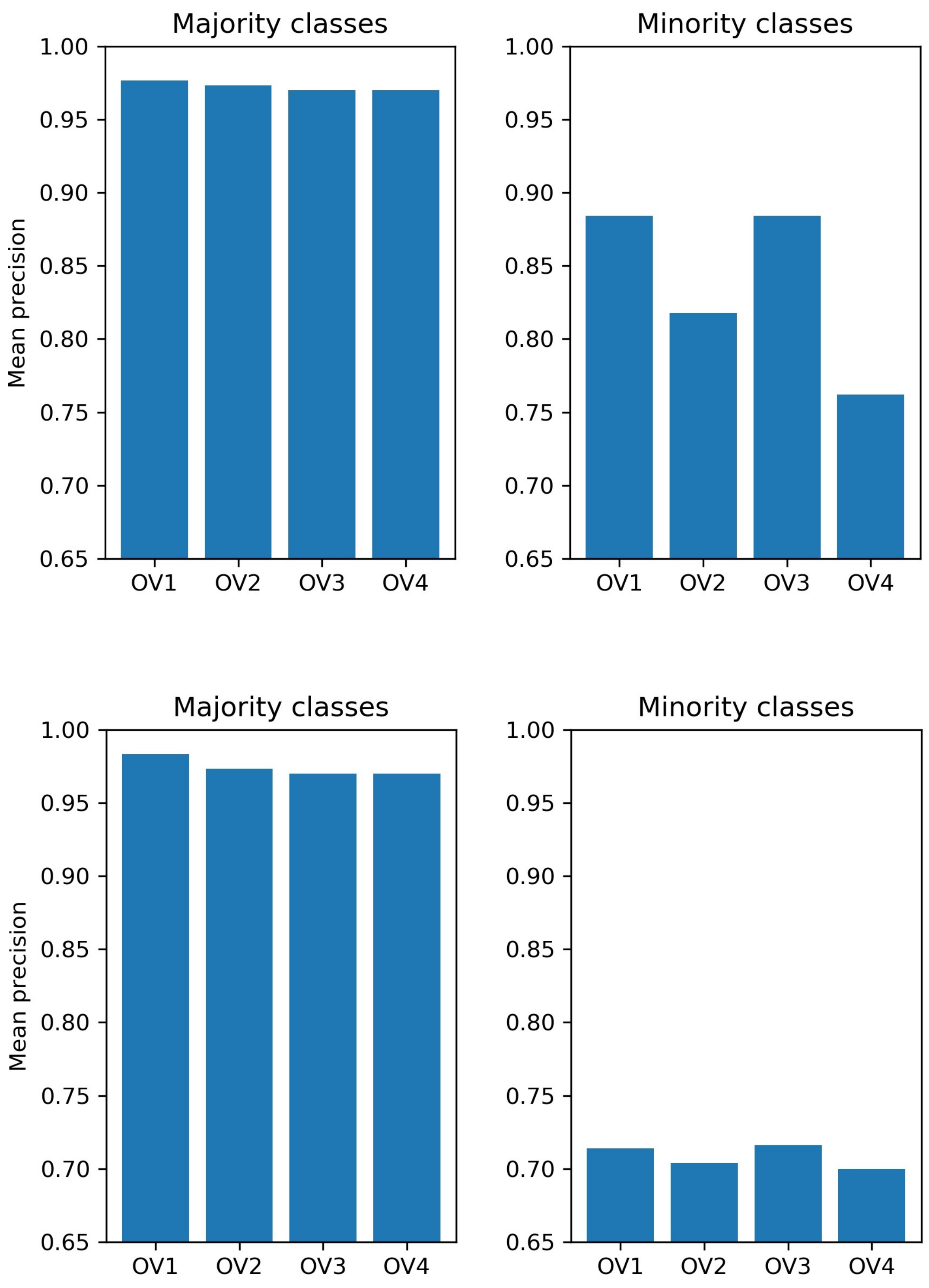


Figure 3.15: Model performance after grayscale reduction. 1-time 4-fold cross-validation per hyperparameter set. Mean precision, first row, and recall, second row, over majority and minority classes, see Figure 3.9. It is important to note that class 575, 301 and 255 have yet to be implemented. See Figure 6.6 for further details.

3.2.3 Full preprocessing test

After indepndently evaluating the effects of the different preprocessing hyperparameters in the Section 3.2.2, we combined all these parameters to determine the fittest preprocessing technique for the models.

Since individually all hyperparameters led to no significant losses in model performance, except for grayscale, two different preprocessing pipelines were implemented and tested. Both of them included size label removal, normalization and background reflection during Geometrical and Random Augmentation. Additionally, one of them implemented grayscale as well, to finally determine if its previously observed performance reduction (Figure 3.15) could be compensated. Finally, maximum class size was set to 1000, that is, 2556 and 852 samples for training and validation set respectively, to speed up training.

Results indicate that, despite the inclusion of all previous parameters during preprocessing, grayscale loss in overall performance remains while RGB format provides consistently better models, Figure 3.16. Therefore, grayscale was discarded from the final preprocessing stage.

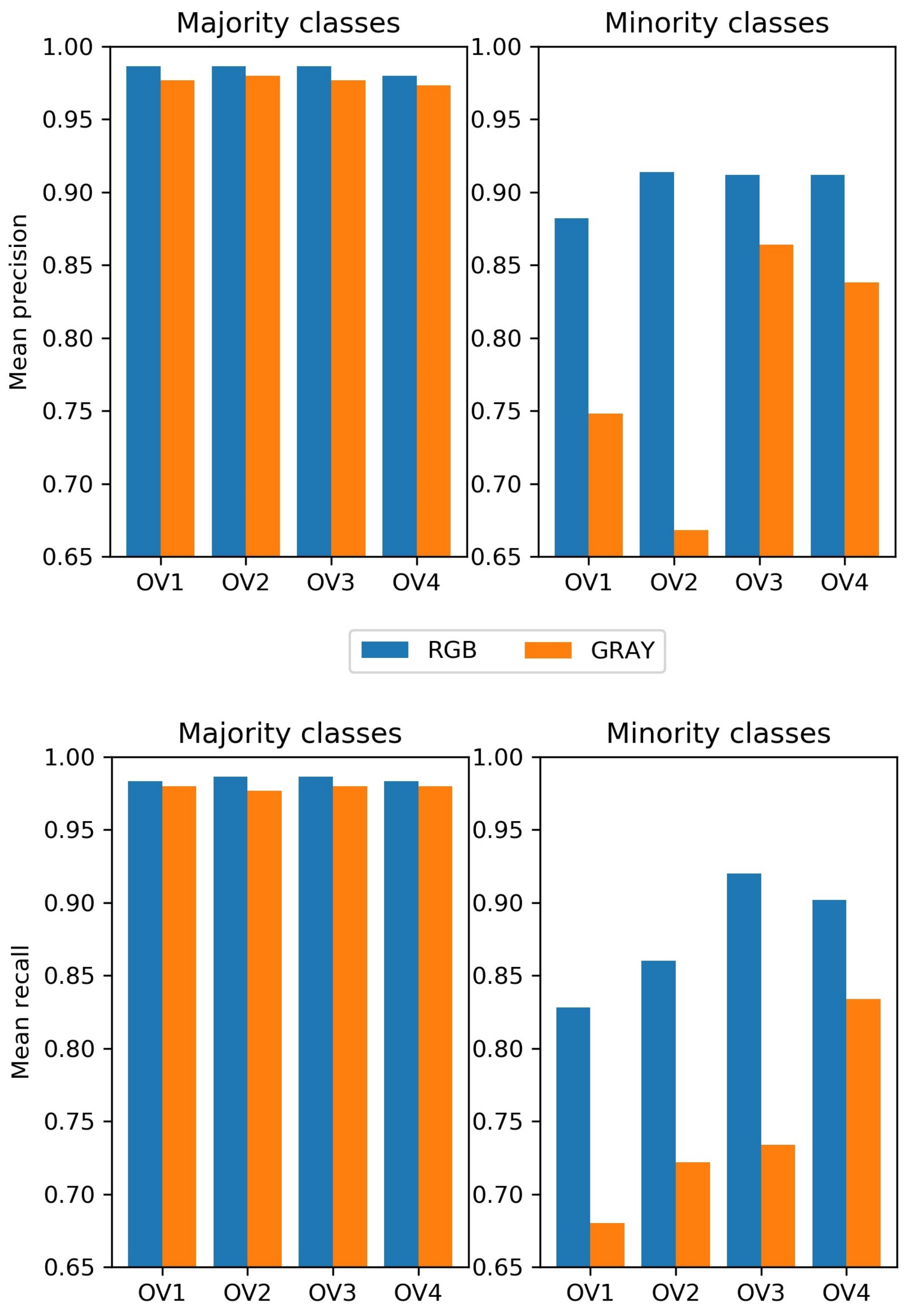


Figure 3.16: Model performance with/-out grayscale. 1-time 4-fold cross-validation per hyperparameter set. Mean precision, first row, and recall, second row, over majority and minority classes, see Figure 3.9, per image format. It is important to note that class 575, 301 and 255 have yet to be implemented. See Figure 6.7 & 6.8 for further details.

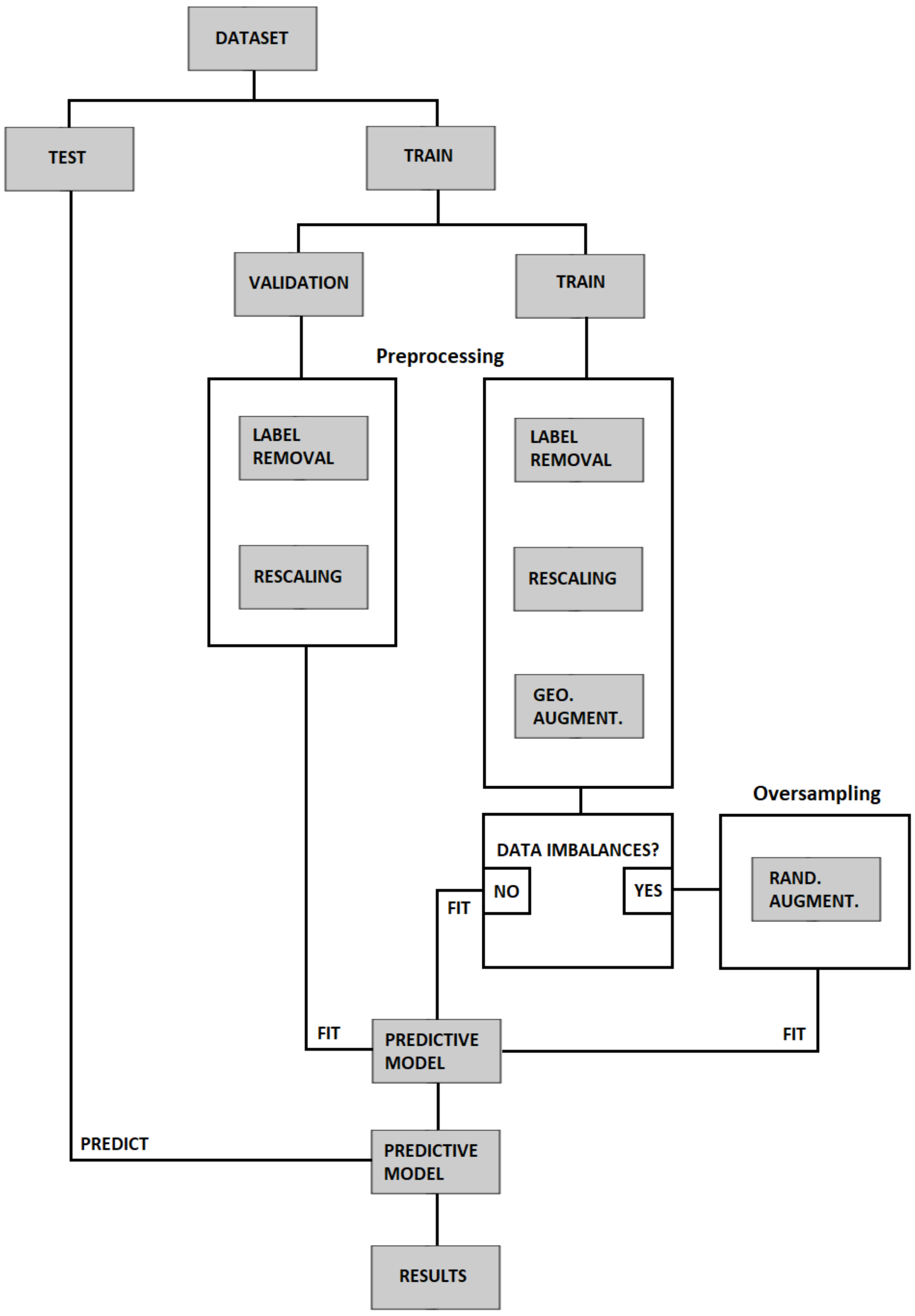


Figure 3.17: Illustration of dataset final preprocessing pipeline. Images in RGB format. Background reflection enabled for all geometrical augmentations.

3.2.4 First stage model

Final Oversampling. Results so far, Section 3.2.1, suggest that increasing oversampling's magnitudes does not lead to performance reduction, and potentially increases generalization since new artificial samples are more likely to be more distinguishable from one another, we decided to investigate even larger magnitudes and probabilities for the policies that compose RA.
Taking OV4 as a reference, we expect to reach some threshold over which the magnitudes would be too large, and the new artificial defects would end up deformed by the transformations applied, thus losing information and decreasing performance. Finally, maximum class size was set to 1000, that is, 6000 and 852 samples for training and validation set respectively, to speed up training.
However, our results suggest greater flexibility for oversampling than originally thought, Figure 3.18, since increasing the magnitudes does not seems to have a consistent tendency towards lower performance. Therefore, the probabilities and magnitudes for RA policies could be selected with relative freedom, and we chose a set of values we considered would work best on our dataset for later models. We'll refer to this set as Oversampling Final (OVF)

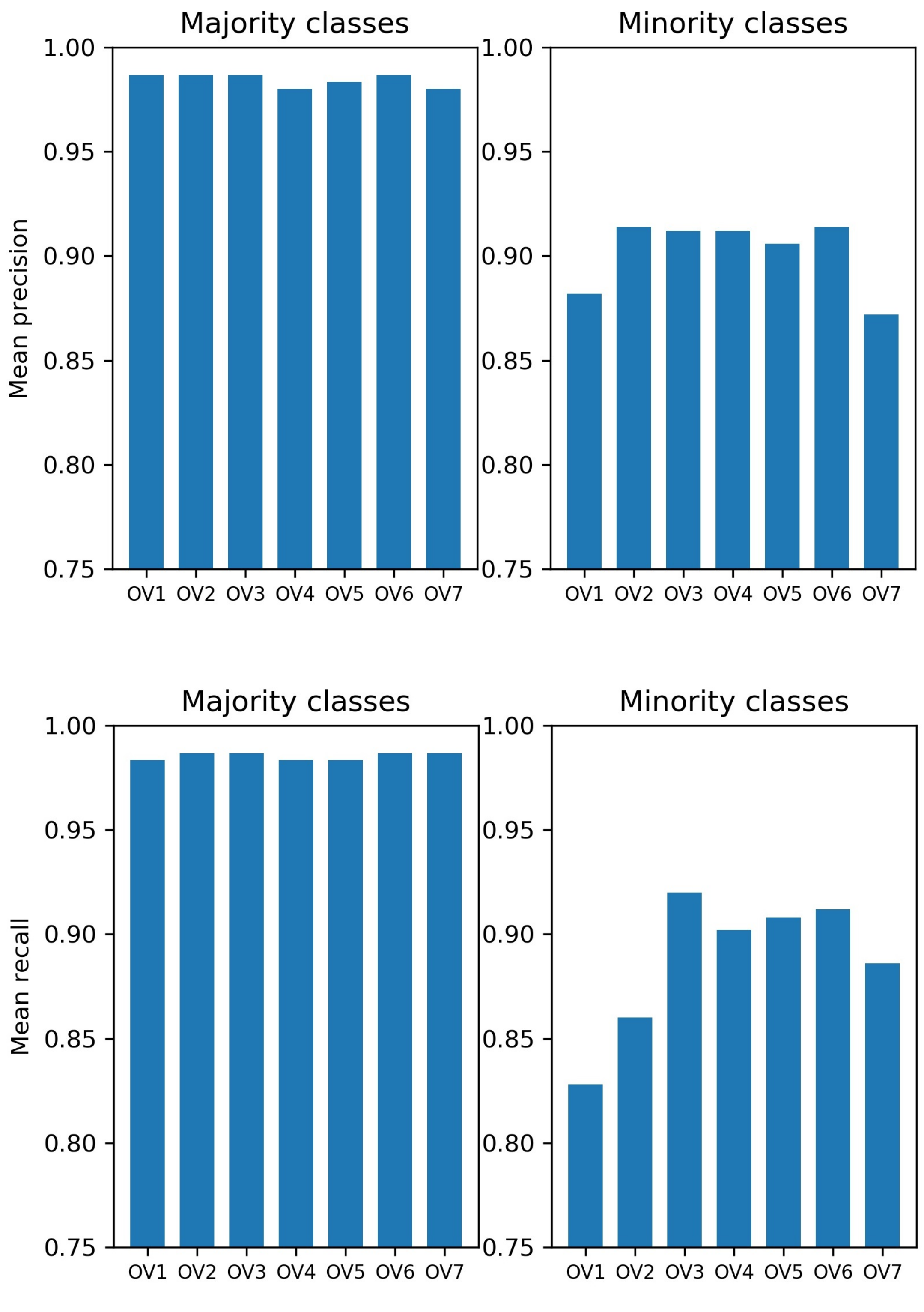


Figure 3.18: Model performance with additional oversampling. 1-time 4-fold cross-validation per hyperparameter set. Mean precision, first row, and recall, second row, over majority and minority classes, see Figure 3.9. It is important to note that class 575, 301 and 255 have yet to be implemented. See Figure 6.9 for further details.

Data cleaning and full size dataset. Once determined the final set of magnitudes during oversampling, we removed the threshold imposed upon the maximum class size so that all samples could be considered. Therefore, all minority classes would be oversampled in order to reach the same size as the largest majority class, that is, class 360 (5103 samples). Thus, for oversampling models 30618 and 2702, while for reference models 8104 and 2702 samples for training and validation set respectively.
Additionally, we inspected and cleaned the dataset, since we realized some of the samples were either located in the wrong class (mislabelled), or contained no defect at all, appearing simply as a gray background (Figure 3.27).
Results, Figure 3.19, indicate that: first, models with oversampling surpass those without it for minority class recall, while in precision they fall short. Therefore, overall accuracy in both cases is almost identical.
Analysis of the models' heatmaps indicated the presence of more noise within the dataset, despite previous cleaning, so further visual inspection and data cleaning was still required. Additionally, the models sometimes focused their attention on the background rather than the defect itself, which seems to have no relation to oversampling itself, since it happens for the reference as well, Figure 3.20. We considered label noise the reason behind such behavior, and expect data cleaning to improve heatmap visualization.
On an additional note, we further investigated model robustness through its saliency maps, Figure 3.21. Unlike the heatmaps, which despite their arbitrary activation on the background for some samples, keeps most of its focus on the features of interest, saliency maps seem to lack this consistency.
We attributed this behavior to a faulty implementation of the code, but further experimentation with different models indicated otherwise. Since heatmaps already provided a clear representation of the model robustness, we decided to remove saliency maps from the visual inspection stage and focus solely on CAM.
After inspecting the dataset, more mislabelled samples were located, within these classes. Therefore, further data cleaning was required.

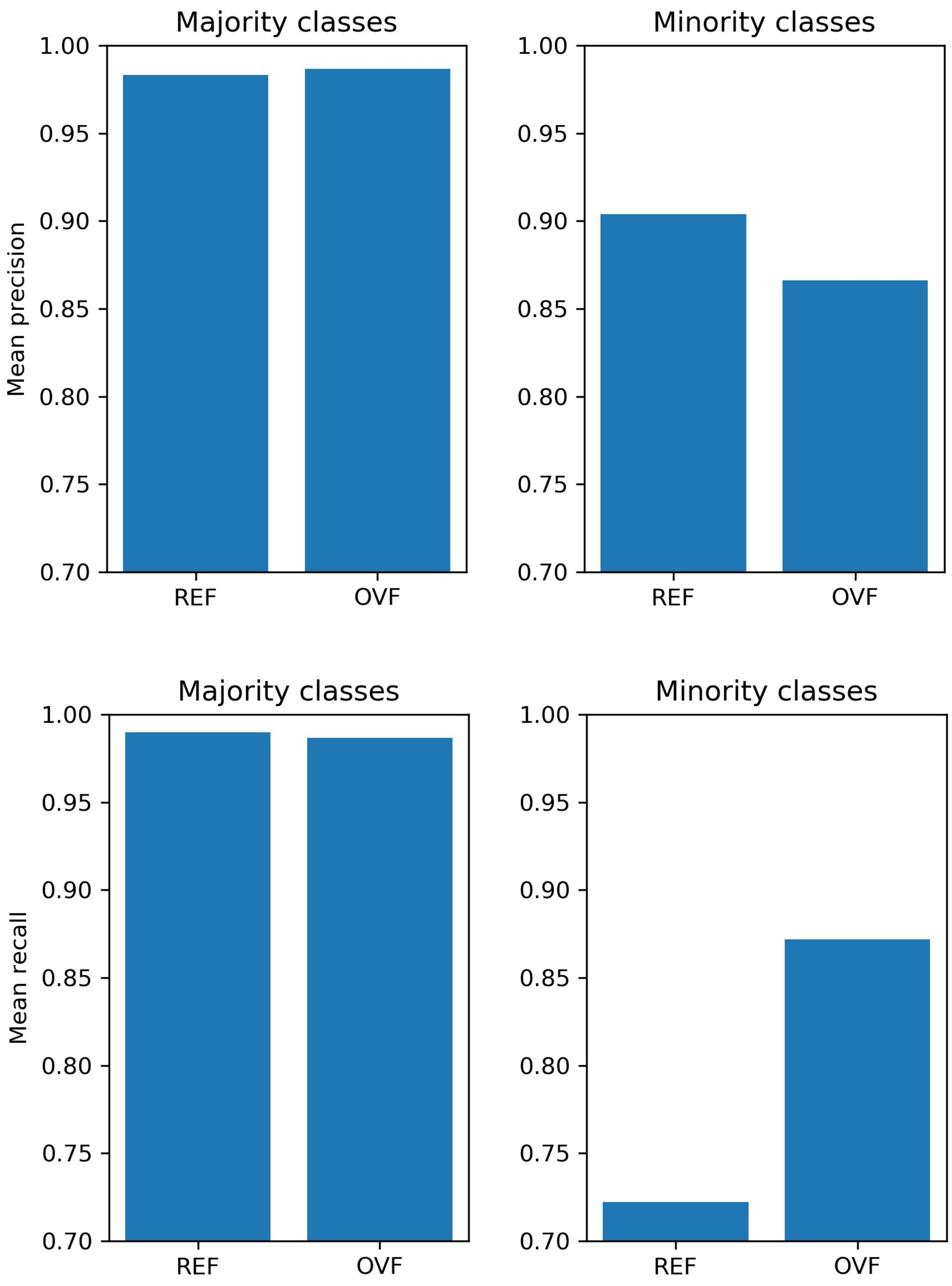


Figure 3.19: First stage model performance. 5-times 4-fold cross-validation per hyperparameter set. Mean precision, first row, and recall, second row, over majority and minority classes, see Figure 3.9. It is important to note that class 575, 301 and 255 have yet to be implemented. See Figure 6.10 for further details.

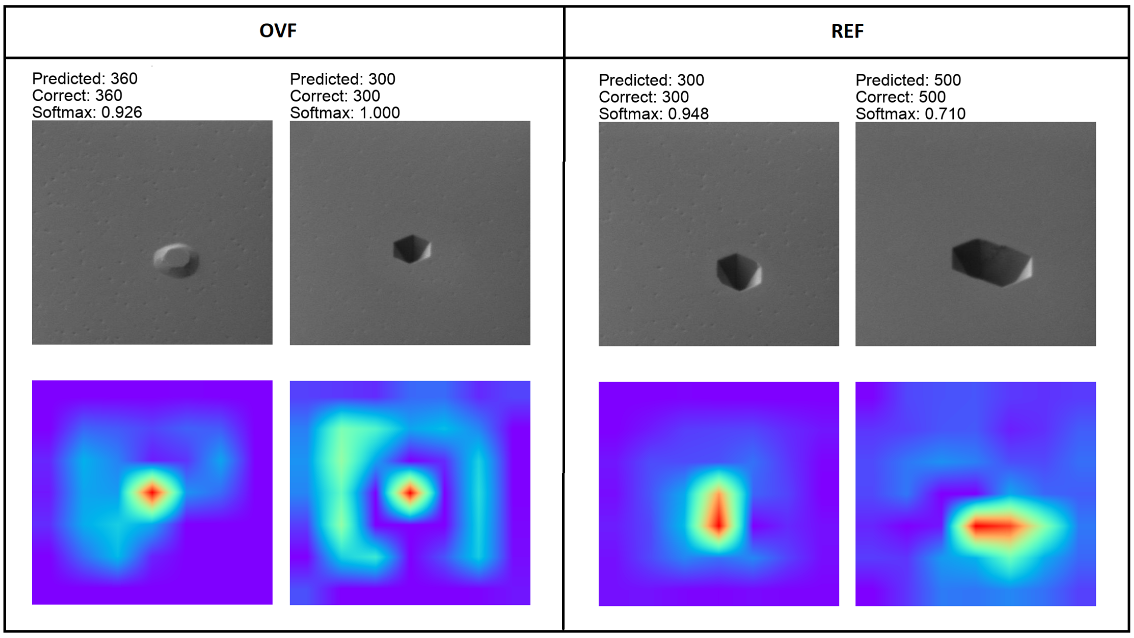


Figure 3.20: First stage model heatmap examples. Overactivation on the background seems more clearly visible in models with oversampling (left), though they are also present in the reference model (right). Pixels with a high activation show up in red/yellow versus those with a low activation in purple/blue.

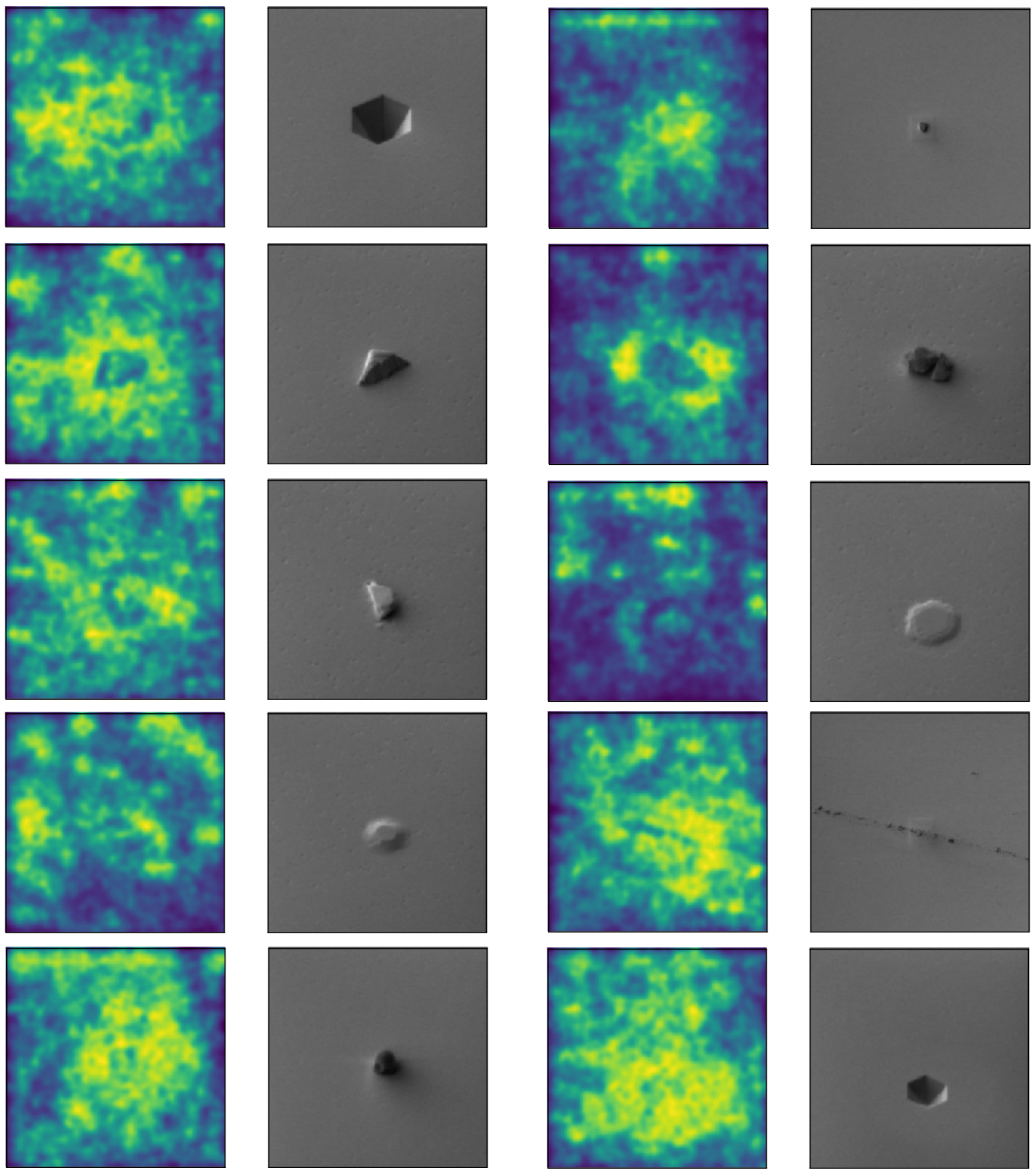


Figure 3.21: First stage model saliency map examples. Overactivation on the background takes place for all samples, independent of the class. Pixels with a high gradient show up in yellow versus those with a low gradient in blue.

3.2.5 Second stage model

Class expansion. Now that the model training for our imbalanced dataset has been completed, which achieved great performance for all eight classes so far (Figure 3.19), we proceed to the next step of our work. This consists on the inclusion of two additional classes to the dataset, 575 and 301. See Section 3.1.4.

Further inspection and cleaning of the dataset was performed after introduction of both classes. Therefore, despite similarities between class 575 and 200, as well as class 301 and 360, Figure 3.22, we expect to have minimized the confusion between them.

In order to investigate in greater detail the individual effect that each class inclusion had, we developed three different dataset configurations: the first contained only the original eight classes; the second contained nine classes, where we included class 575 and 301, but 575 were labeled as class 200; and the third had the same structure as the second, but with the class 575 now labelled as an independent class. They are referred to as *C8*, *C9* and *C10* respectively.

According to the previous statement, we therefore counted with three different datasets. Maximum class size was set to 1000, which led to 6000 and 852 samples for *C8*, and 7500 and 992 for *C9* and *C10* for the training and validation set respectively.

Results (Figure 3.23) indicate that data cleaning improved base accuracy for all cases, while the inclusion of class 575 and 301 didn't decrease model performance. In fact, precision and recall of both new minority classes is considerably high, which increased mean performance.

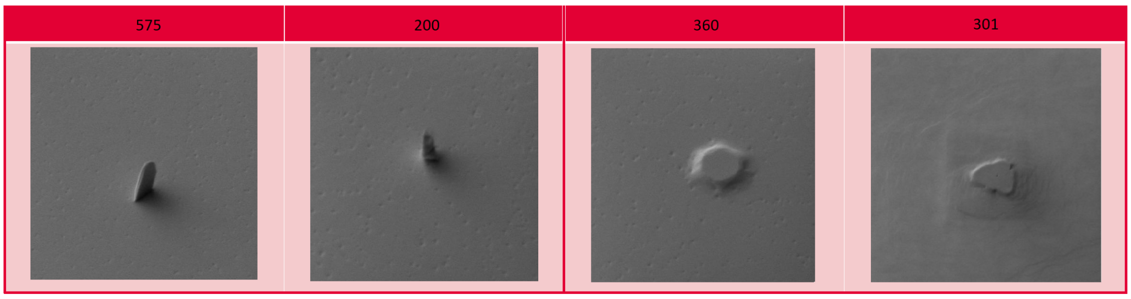


Figure 3.22: Illustration of between-class similarities. Class 575 differentiating feature is their sharp outline. For class 301, their differentiating feature consists on the 'staircase' effect surrounding the main bump.

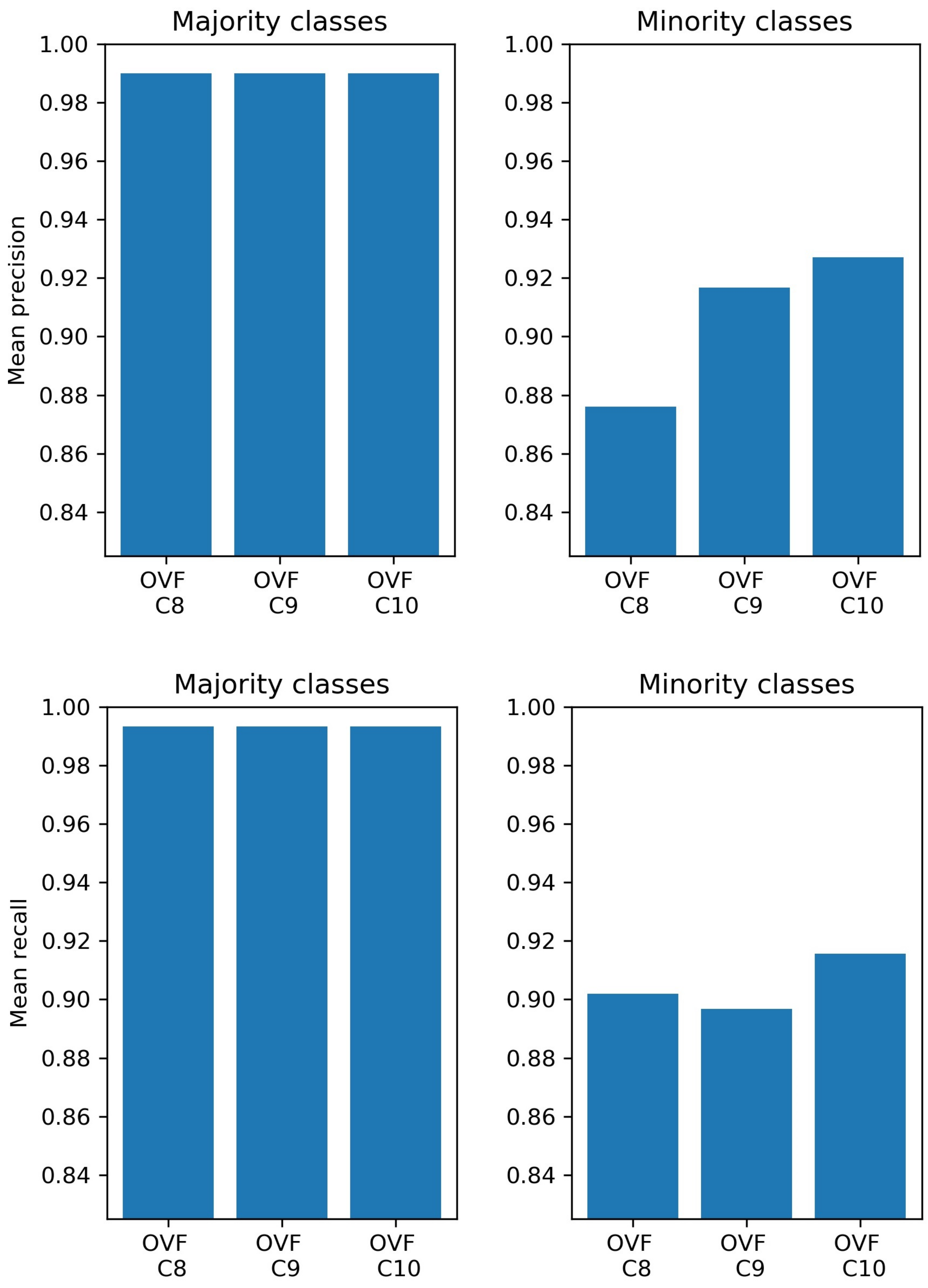


Figure 3.23: Second stage model performance over class expansion. 1-time 4-fold cross-validation per number of classes implemented (*C8*, *C9*, *C10*). Mean precision, first row, and recall, second row, over majority and minority classes, see Figure 3.9. It is important to note that class 255 has yet to be implemented. See Figure 6.11 for further details.

Label smoothing. The presence of noisy images within the dataset has been a constant issue during the project. Despite the several sessions of data cleaning performed, noise might still remain within the dataset. Naturally, after subsequent sessions this noise has decreased, but in order to guard off further against this, we proceeded to the implementation of label smoothing into our models. Maximum class size was set to 1000, which implies 7500 and 992 samples for the training and validation set respectively.
It's worth noting that before implementing label smoothing, another session of data cleaning was performed. In addition, Label smoothing is already implemented in Tensorflow within the cross entropy loss functions, and since no common convention or formula exists regarding the value of alpha, its selection had to be done through trial and error using our own estimation. See Section 3.1.
Results (Figure 3.24) suggest that label smoothing does not significantly change model performance. However, inspection of the heatmaps showed that small values of label smoothing improved activation distribution, which is now more focused on the defect and less scarce (Figure 3.25), with $\alpha = 0.025$ providing the best heatmaps.
The reason behind such behavior lies in the interaction between the target labels and the backpropagation algorithm [37],[45]. Since label smoothing softens the targets, all output neurons' values will be superior to zero. Thus, all hidden units will be updated during backpropagation, reducing scarcity, since target values equal to zero do not modify hidden weights.
Given these results, we decided to implement label smoothing with an alpha of 0.025, which according to Eq. 3.1, corresponds to 97.7% prediction confidence.

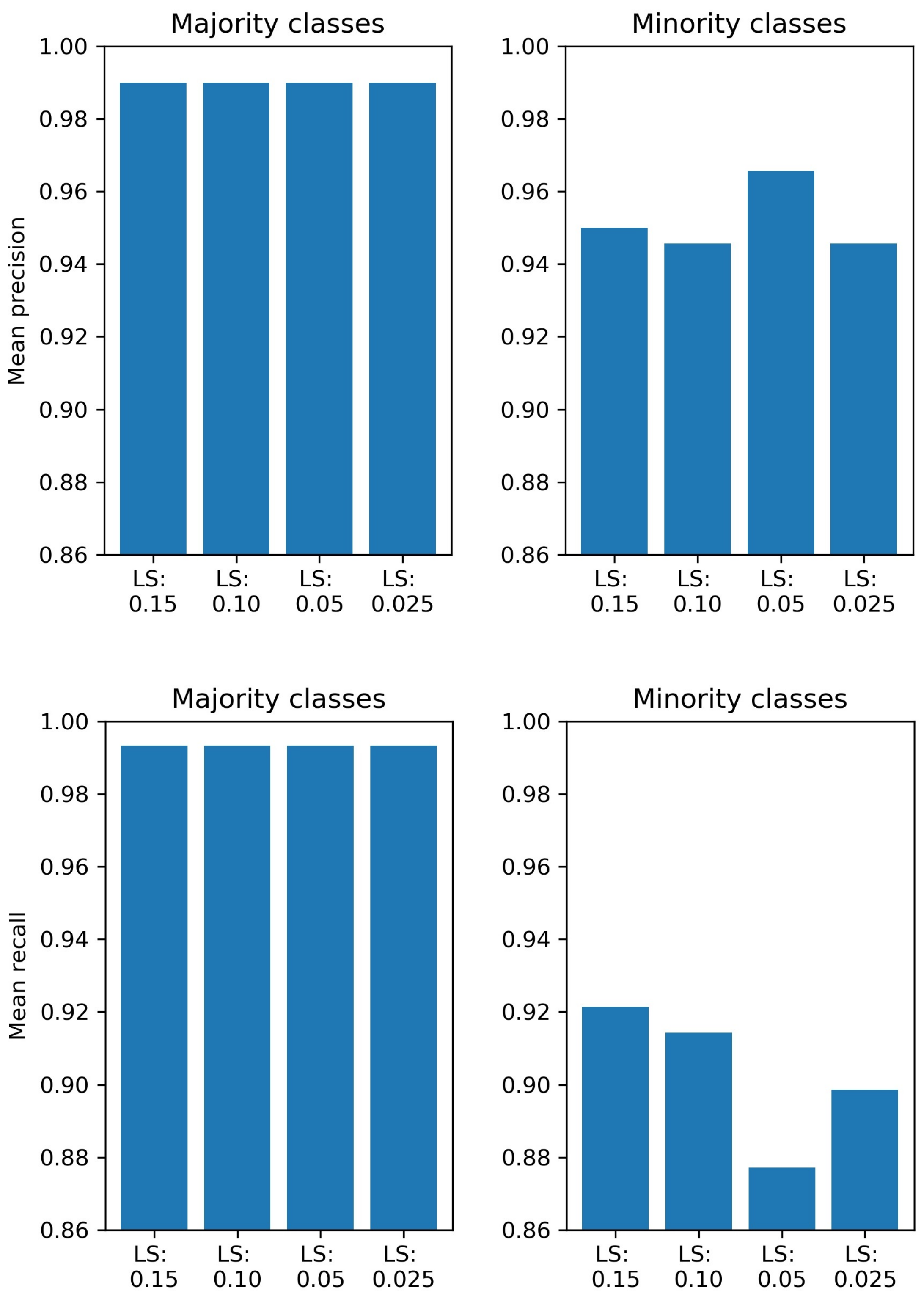


Figure 3.24: Second stage model performance over label smoothing. 1-time 4-fold cross-validation per alpha value. Mean precision, first row, and recall, second row, over majority and minority classes, see Figure 3.9. It is important to note that class 255 has yet to be implemented. See Figure 6.12 for further details.

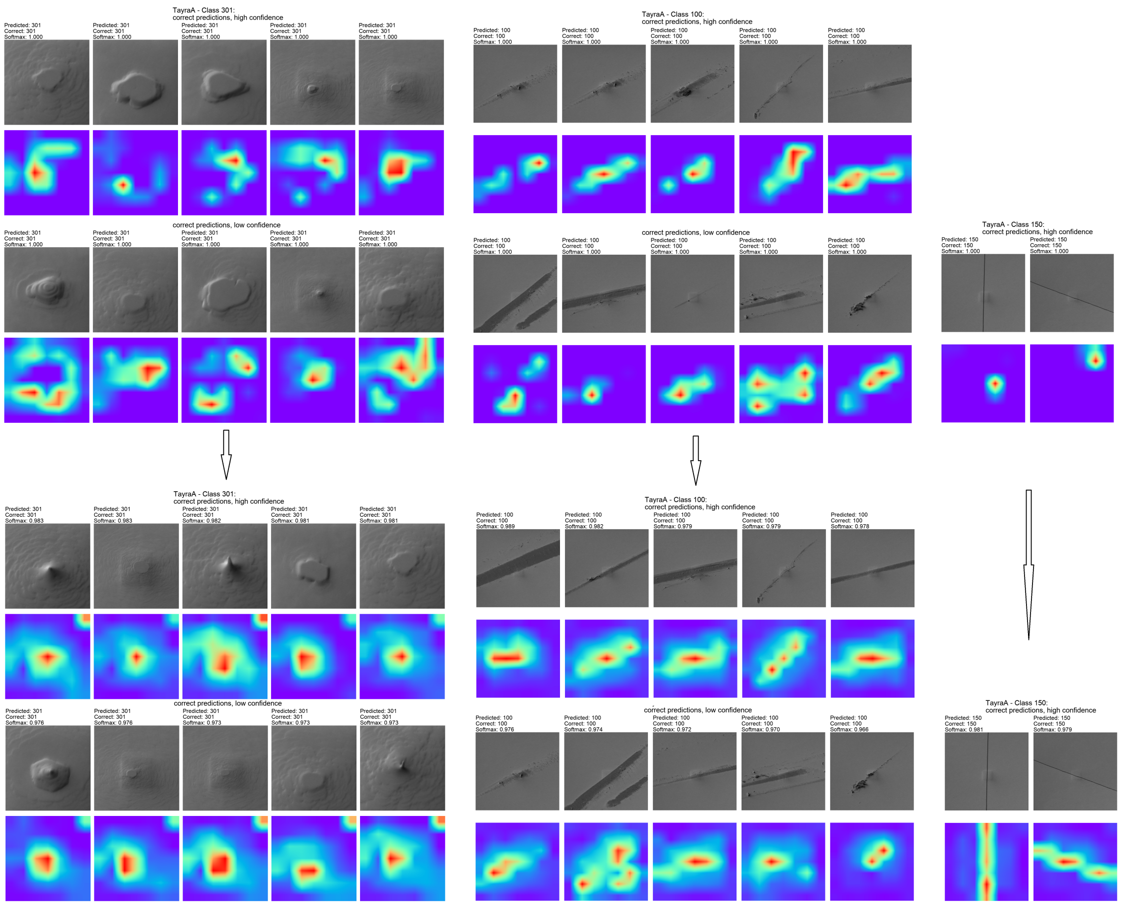

TayraA - Class 301:
correct predictions, high confidence
Predicted: 301 Correct: 301 Softmax: 1.000
correct predictions, low confidence
Predicted: 301 Correct: 301 Softmax: 1.000
TayraA - Class 100:
correct predictions, high confidence
Predicted: 100 Correct: 100 Softmax: 1.000
correct predictions, low confidence
Predicted: 100 Correct: 100 Softmax: 1.000
TayraA - Class 150:
correct predictions, high confidence
Predicted: 150 Correct: 150 Softmax: 1.000
TayraA - Class 301:
correct predictions, high confidence
Predicted: 301 Correct: 301 Softmax: 0.983
Predicted: 301 Correct: 301 Softmax: 0.983
Predicted: 301 Correct: 301 Softmax: 0.982
Predicted: 301 Correct: 301 Softmax: 0.981
Predicted: 301 Correct: 301 Softmax: 0.981
correct predictions, low confidence
Predicted: 301 Correct: 301 Softmax: 0.976
Predicted: 301 Correct: 301 Softmax: 0.976
Predicted: 301 Correct: 301 Softmax: 0.973
Predicted: 301 Correct: 301 Softmax: 0.973
Predicted: 301 Correct: 301 Softmax: 0.973
TayraA - Class 100:
correct predictions, high confidence
Predicted: 100 Correct: 100 Softmax: 0.989
Predicted: 100 Correct: 100 Softmax: 0.982
Predicted: 100 Correct: 100 Softmax: 0.979
Predicted: 100 Correct: 100 Softmax: 0.979
Predicted: 100 Correct: 100 Softmax: 0.978
correct predictions, low confidence
Predicted: 100 Correct: 100 Softmax: 0.976
Predicted: 100 Correct: 100 Softmax: 0.974
Predicted: 100 Correct: 100 Softmax: 0.972
Predicted: 100 Correct: 100 Softmax: 0.970
Predicted: 100 Correct: 100 Softmax: 0.966
TayraA - Class 150:
correct predictions, high confidence
Predicted: 150 Correct: 150 Softmax: 0.981
Predicted: 150 Correct: 150 Softmax: 0.979

Figure 3.25: Second stage model heatmap improvement. Implementation of label smoothing led to more evenly distributed activation maps in all defects since backpropagation scarcity was reduced.

Evaluation of larger architectures. Now that the second stage model for defect classification using the smallest architecture, EfficientNet B0, available is completed, we'll investigate the effects of higher complexity structures, EfficientNet B1, B2, B3 and B4, on performance. We expect to reach maximum performance with EfficientNet B1, since the architecture's size is almost doubled from B0 to B1, while for higher versions this change is considerably smaller, which would lead to more diificult to notice improvements.
Due to hardware limitations, as network architectures increased, we had to reduce both the maximum size of the classes, 1000 samples per class, and the size of our batches so that the models could train without interruptions. Starting from EfficientNet B0 up until B4, we used batches of size 64, 32, 24, 14 and 6 samples respectively. Finally, the maximum class size was set to 1000, that is, 7500 and 992 for the training and validation set respectively. Using EfficientNet B1 as the reference (Figure 3.26), we observe no significant changes in model performance for architectures above B1, with B3 and B4 providing similar results, while B2 led to the weakest improvements. It's clear B1 shows better results in precision and recall when compared to EfficientNet B0, as expected.
Therefore, for the remaining of the thesis we focused our efforts on the two smallest architectures available from EfficientNet, B0 and B1.

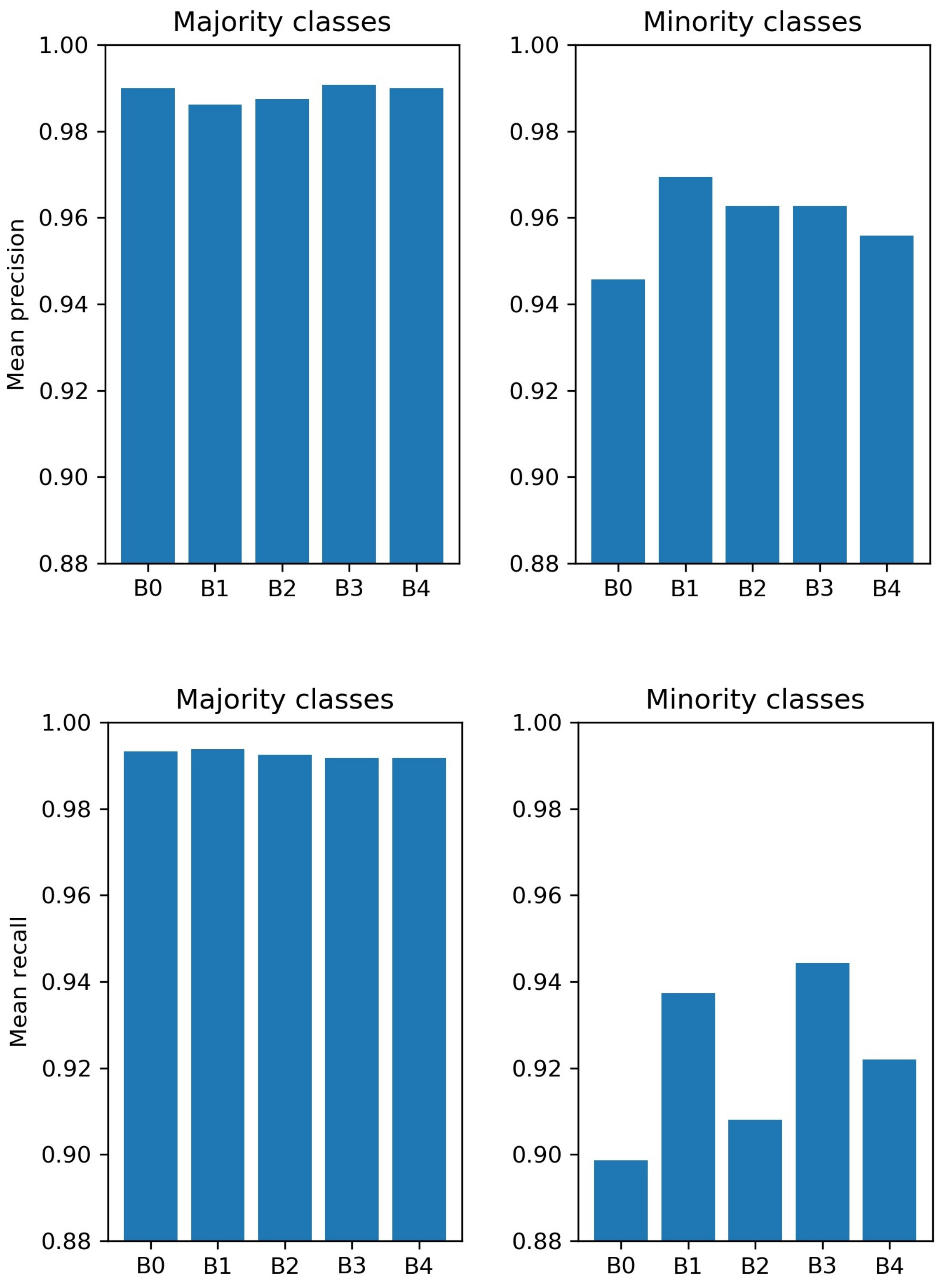


Figure 3.26: EfficicentNet architectures performance. 1-time 4-fold cross-validation per architecture. Mean precision, first row, and recall, second row, over majority and minority classes, see Figure 3.9. It is important to note that class 255 has yet to be implemented. See Figure 6.13 for further details.

3.2.6 Third stage model

Class expansion. Once the second stage model was implemented and tested in the production line with new defect samples from the SEM, we found out that considerable amount of these did not contain any actual defects. The presence of these kind of images is related to the lack of precision of the SEM, which samples the wafer surface automatically. Since the generation of No-Defect samples cannot be avoided in production, the simplest solution was to introduce this new set as an additional class for our model.
It is important to note the mark left by the secondary electron detector when sampling, Figure 3.27, which appears in the form of a square. Though this mark can be found in all classes learnt so far, our concern is that it might be mistaken by an actual feature for the No-defect class.
Therefore, we'll count with two different datasets, one without and another one with No-Defect class implemented, C10 and C11 respectively. Maximum class size was set to 5103, which means 38272 and 2842 in C10, and 42099 and 2866 samples in C11 for the training and validation set respectively.
Results indicate that introduction of No-Defect class had small effects on model performance, Figure 3.28, remaining fairly similar to the second stage model. Increases in precision and recall are caused by the high accuracy of the No-Defect class, which could be easily predicted by our model. Confusion graphs have now been replaced by matrices in order to analyze in greater detail the degree of missclassification. Confusion matrix, Figure 3.29, further proves there's no conflict between No-Defect and any of the other classes, being completely differentiable from one another.

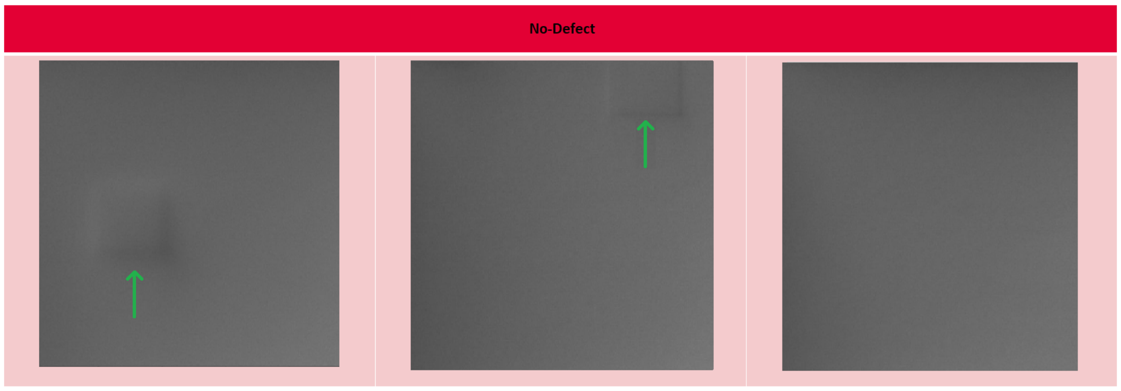


Figure 3.27: Illustration of No-Defect class. Green arrow points out the presence of the square mark left by the electron beam on wafer surface.

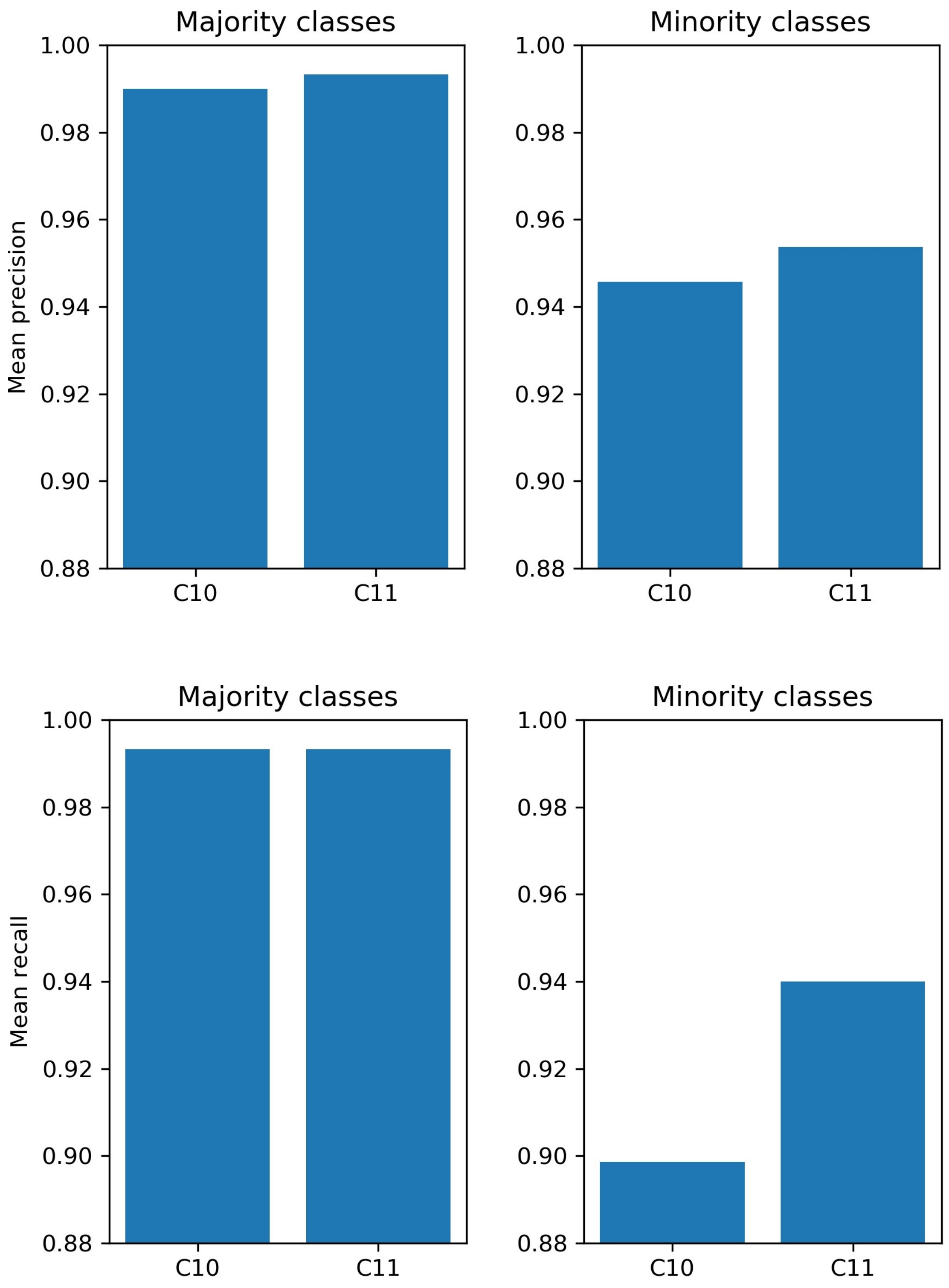


Figure 3.28: Third stage model performance after No-Defect inclusion. 1-time 4-fold cross-validation per number of classes on EfficientNet B0 architecture. Mean precision, first row, and recall, second row, over majority and minority classes, Figure 3.9. See Figure 6.14 for further details.

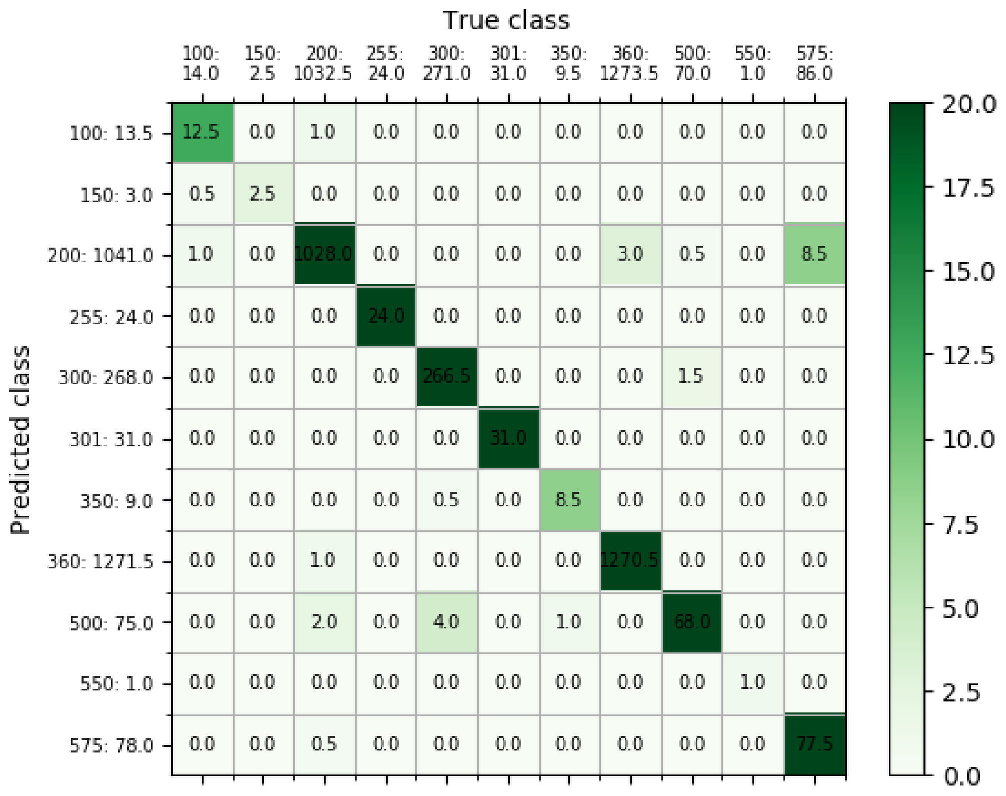


Figure 3.29: Third stage model confusion matrix after No-Defect inclusion. 1-time 4-fold cross-validation on EfficientNet B0 architecture.

	OV1	*OV2*	*OV3*	*OV4*	*OV5*	*OV6*	*OV7*	*OVF*	*OVF/2*
μ	0	0	0	0	0	0	0	0	0
σ	0.5	1	1.5	2	3	4	5	4	2
Crop left	2%	4%	6%	8%	10%	10%	10%	8%	4%
Crop right	2%	4%	6%	8%	10%	10%	10%	8%	4%
Crop top	2%	4%	6%	8%	10%	10%	10%	8%	4%
Crop bottom	2%	4%	6%	8%	10%	10%	10%	8%	4%
Shear X	0.15	0.3	0.45	0.6	0.75	0.9	0.9	0.5	0.4
Shear Y	0.15	0.3	0.45	0.6	0.75	0.9	0.9	0.5	0.4
Shear XY	0.05	0.1	0.2	0.3	0.4	0.5	0.5	0.4	0.2
β_{low}	-15	-30	-45	-60	-75	-90	-90	-70	-50
β_{high}	15	30	45	60	75	90	90	70	50
ϕ	5	10	15	20	25	30	40	25	20
Reflect	True	True	True	True	True	True	True	True	True
Flip	1	1	1	1	1	1	1	1	1
Blur	3	3	3	3	3	3	3	3	3

Table 3.1: Oversampling hyperparameter sets. μ and σ stand for the mean and standard deviation of the Gaussian filter. *Crop left, right, top* and *bottom* are the maximum percentage of the image to crop out. *Shear X, Y* and *XY* contain the angle of shearing. β_{low} and β_{high} stand for the maximum and minimum change in contrast. ϕ is the maximum angle of rotation. *Reflect* determines if image reflection as background is enabled or not (default True). *Flip* value determines if flipping is performed horizontally, -1, vertically, 0, or both, 1 (default 1). *Blur* contains the blurring kernel size (default 3).

Oversampling mitigation. Model performance so far has been considerably high. However, we still wonder if the final hyperparameter set we have chosen for oversampling is in fact the optimal one. This concern arises from the fact that, for geometrical transformations, defect duplication due to background reflection remains an issue.
Though it doesn't seem to affect model performance significantly, Figure 6.9, other instabilities such as class confusion, background over activation in defect heatmaps, especially for class 575, or the lack of consistency in saliency maps might be one of its consequences.
Therefore, we reduced the magnitude of the oversampling parameters down to half of the original values of OVF, and trained a new model to investigate its effects. Table 3.1 contains all hyperparameters sets we experimented upon for data augmentation. Finally, we set maximum class size to 5103, with 42099 and 2866 samples for the training and validation set respectively.
Results on performsance, Figure 3.30, suggest that by reducing oversampling's magnitude, we obtain significant increases on precision/recall for the minority classes, up to +2%, without any significant changes in class confusion, Figure 3.31. However, saliency and heatmaps instabilities still remain, which leads us to the next point.

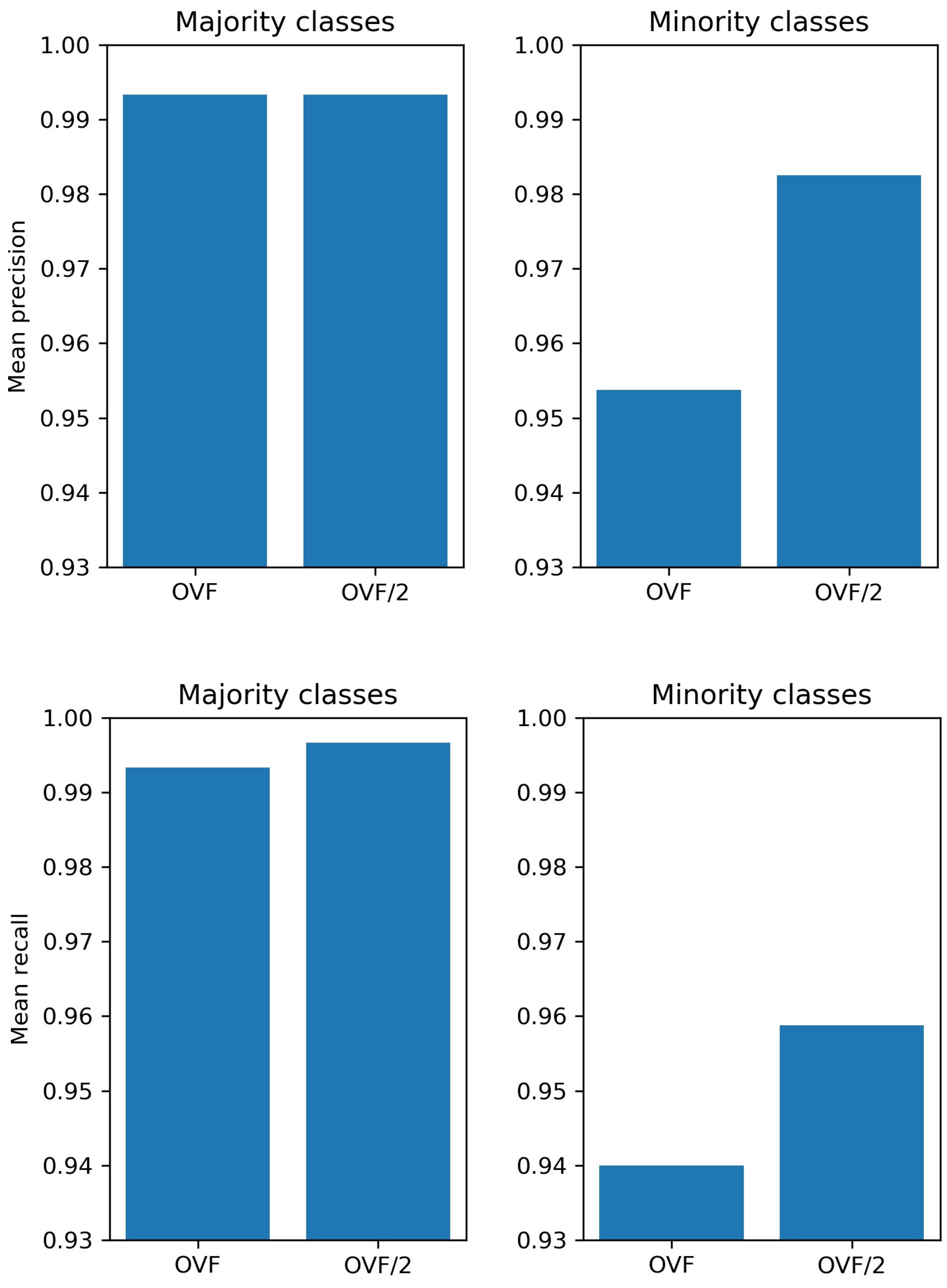
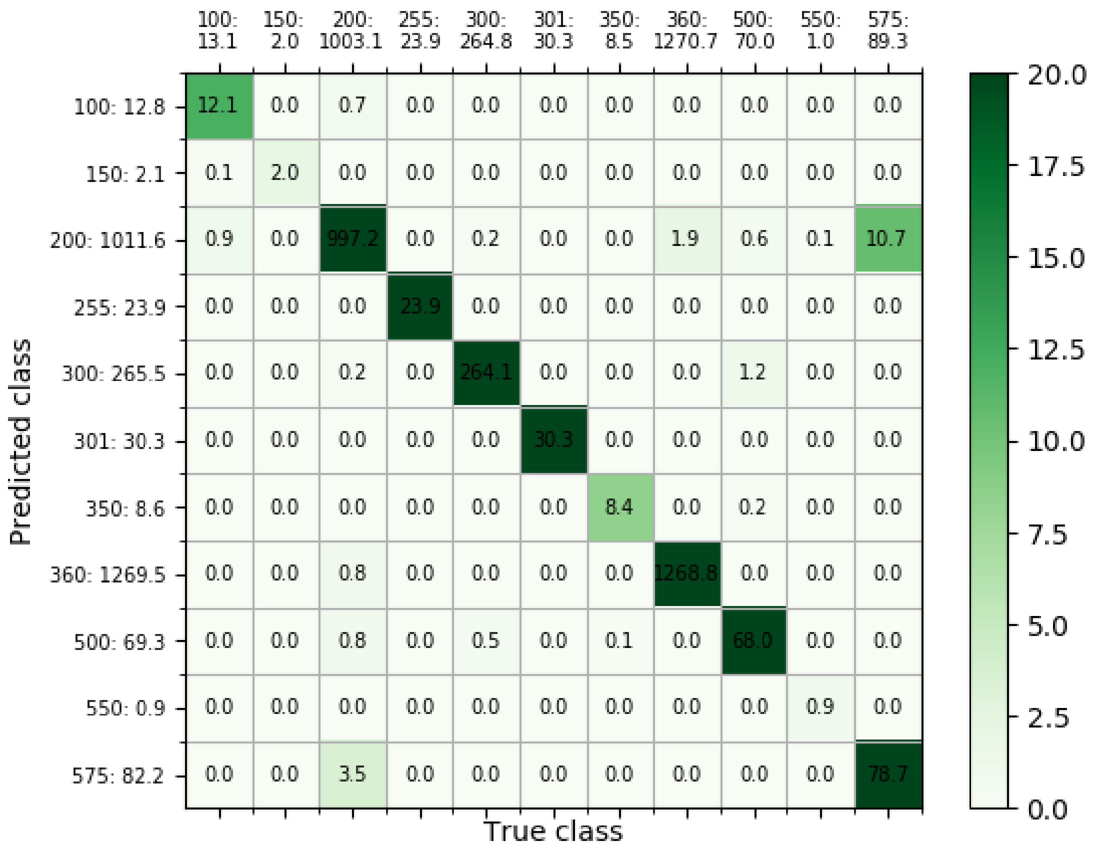


Figure 3.30: Third stage model performance after oversample mitigation. 1-time 4-fold cross-validation per oversampling hyperparameter set on EfficientNet B0 architecture. Mean precision, first row, and recall, second row, over majority and minority classes, Figure 3.9. See Figure 6.15 for further details.



Predicted class \ True class	100: 13.1	150: 2.0	200: 1003.1	255: 23.9	300: 264.8	301: 30.3	350: 8.5	360: 1270.7	500: 70.0	550: 1.0	575: 89.3
100: 12.8	12.1	0.0	0.7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
150: 2.1	0.1	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
200: 1011.6	0.9	0.0	997.2	0.0	0.2	0.0	0.0	1.9	0.6	0.1	10.7
255: 23.9	0.0	0.0	0.0	23.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0
300: 265.5	0.0	0.0	0.2	0.0	264.1	0.0	0.0	0.0	1.2	0.0	0.0
301: 30.3	0.0	0.0	0.0	0.0	0.0	30.3	0.0	0.0	0.0	0.0	0.0
350: 8.6	0.0	0.0	0.0	0.0	0.0	0.0	8.4	0.0	0.2	0.0	0.0
360: 1269.5	0.0	0.0	0.8	0.0	0.0	0.0	0.0	1268.8	0.0	0.0	0.0
500: 69.3	0.0	0.0	0.8	0.0	0.5	0.0	0.1	0.0	68.0	0.0	0.0
550: 0.9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.9	0.0
575: 82.2	0.0	0.0	3.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	78.7

Figure 3.31: Third stage model confusion matrix after oversampling mitigation. 1-time 4-fold cross-validation on EfficientNet B0 architecture.

Class 575 resampling and mixed defects removal. As mentioned before, model heatmaps revealed that activations distribution where more unstable than usual for the class 575. We concluded that this behavior arises from the class 575 itself, whose original samples contained additional labels on the top side of the picture that had to be cropped together with the bottom labels and right black lines, concluding with a re-escalation of the image (Figure 3.33).
This preprocessing on class 575 was significantly more aggressive than for the rest of the classes, since a considerable portion of the sample had to be cropped out. This led to a downgrade of image resolution, which consequently deformed the features present. This specific issue took place before we found out the top labels could be disabled in production.
As a solution, all images from class 575 had to be resampled and properly preprocessed in the same manner as all other classes. Through this change, we hope to fix the activation instabilities observed in the heatmaps, as well as improve generalization.
On an additional note, we observed that for some of our minority classes, such as class 150 and 575, several defects were contained within the same image. Since the model that we wish to obtain is intended for multiclass classification and not multilabelling, we decided to remove and replace this kind of samples from the dataset. Finally, we set maximum class size to 5103, with 42099 and 2866 samples for the training and validation set respectively.
Results indicate that mixed defects removal had little to no effect on the confusion matrix (Figure 3.32). However, class 575 resampling did lead to heatmap improvement, and the activation maps now properly focus on the defect (Figure 3.34). Nonetheless, regarding model performance (Figure 3.35), it seems EfficientNet B1 architecture proved weaker in comparison to B0, despite the former counting with higher complexity.

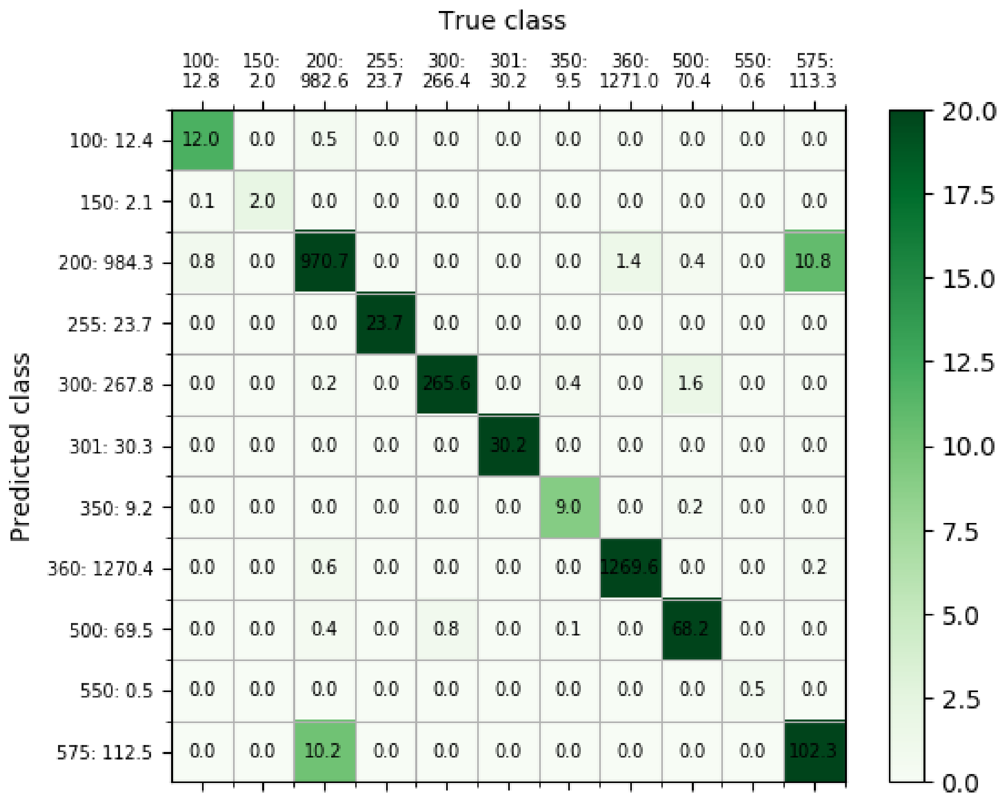


Figure 3.32: Third stage model confusion matrix after class 575 resampling. 1-time 4-fold cross-validation on EfficientNet B0 architecture.

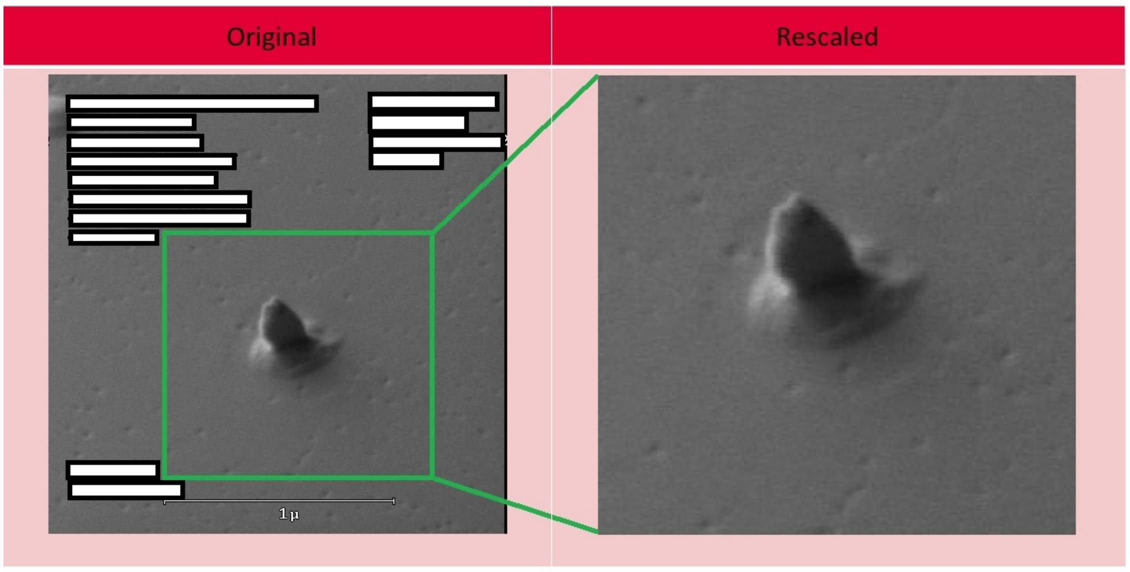


Figure 3.33: Class 575 label cropping and re-scaling. In exchange of label removal, the resolution of the image gets significantly lower. Labels have been concealed to the reader since they contained sensitive information.

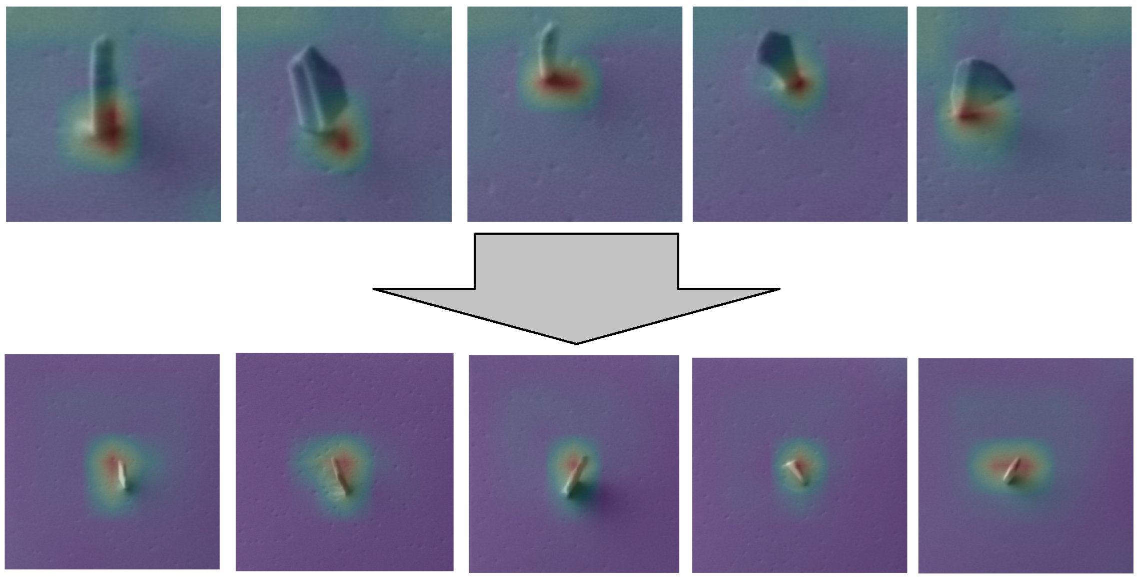


Figure 3.34: Class 575 heatmap improvement. Top row contains activation maps of original low-resolution samples from class 575. Bottom row contains the activation maps of the new higher resolution samples.

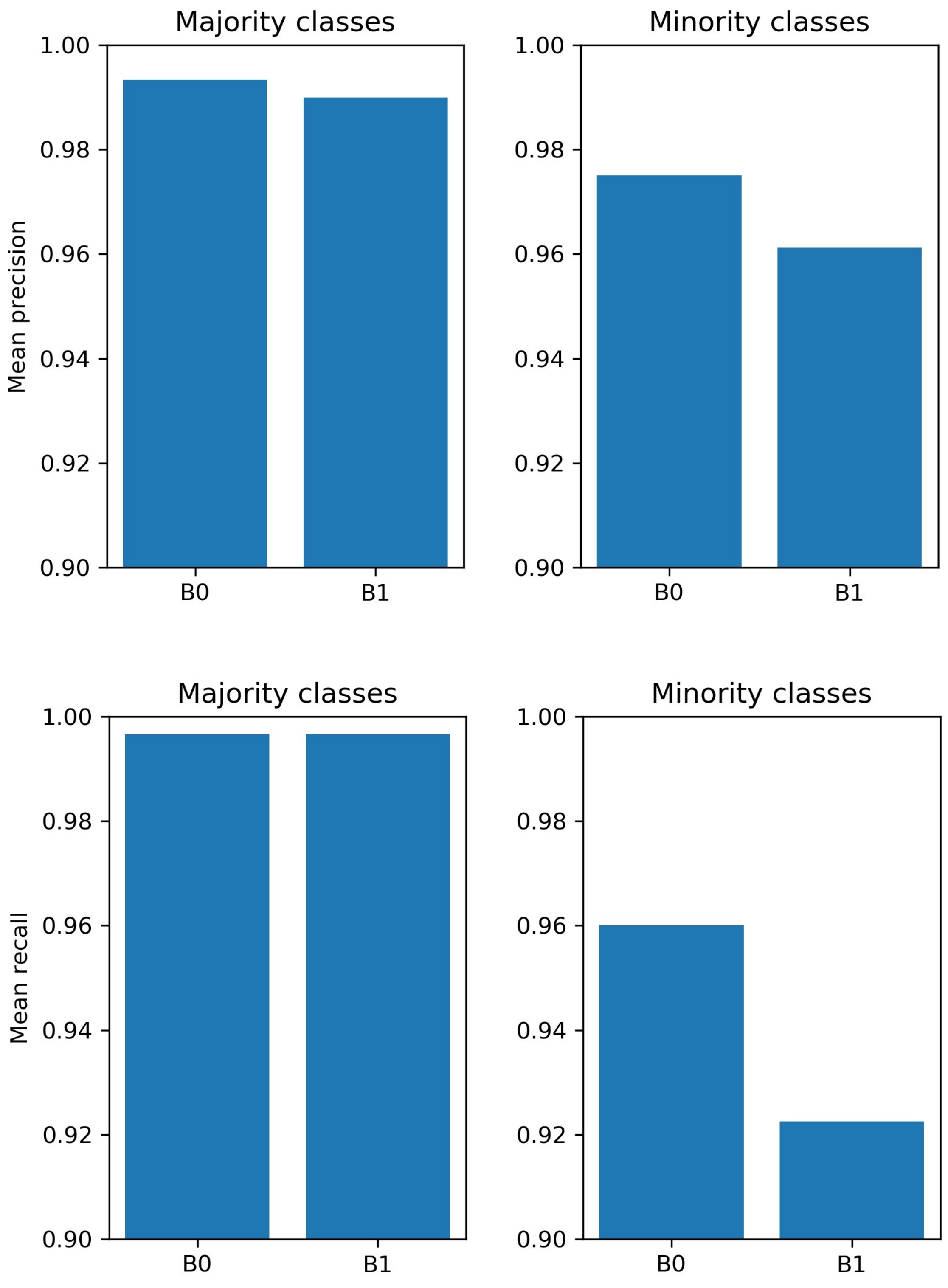


Figure 3.35: Third stage model performance after class 575 resampling. 5-times 4-fold cross-validation per EfficientNet architecture. Mean precision, first row, and recall, second row, over majority and minority classes, Figure 3.9. See Figure 6.16 for further details.

4 Results

Out of the 20 models trained, remember we applied 5-times 4-fold cross-validation in Section 3.2.6, we proceeded to select what we considered the best one based on the following conditions: average precision, average recall and heatmap consistency. Table 4.1 provides a summary of the hyperparameters used during training.

For the EfficientNet B0 family, the chosen model possessed great values in precision and recall, over 93% and 88% for all classes respectively; little confusion among classes, with 200 and 575 as the most conflicting ones; stable training, with no oscillations neither for the training nor validation set, and high robustness, where up to 95% of the images were predicted with over 99% confidence. For the EfficientNet B1 family, the chosen model behaves almost identically to the one from B0 (Figures 4.1,4.2).

Since both models are quasi-equivalent in performance, it would stand to reason to select the one belonging to the B0 architecture for implementation in the production line, since its size is almost half that of the B1 family, whose implementation would therefore spare in computational effort. However, all performance measurements so far have only taken into account the validation set, so a proper evaluation of the model with a new unseen testing set would provide us further insight in which architecture to select for our final decision. As a consequence, we further tested the robustness of each model on a new noisy dataset, that is, samples were not cropped and there were additional labels on the top for all classes. However, images were normalized and rescaled. This testing dataset consisted of the samples shown in Figure 4.4. Unfortunately, we were not able to obtain additional samples for class 100, 301 and 550, due to their low frequency of appearance.

The model from the B1 architecture surpassed the one from the B0 by a large margin, with a difference of almost 15% in precision, and 10% in recall (Figure 4.5). As expected, model prediction/recall in both cases decreased significantly due to the presence of noise, especially for class 150, 255, 300 and 350 for B0, but only 100 and 350 for B1, see Appendix.

Furthermore, B1 validation heatmaps, Figure 4.3, showed proper focus on the defects, with little to no background overactivation for all classes, remaining most active on the

features of interest.
Model was implemented in the production line for a week, and the samples classified by it were compared with those of an operator in order to test its precision. The algorithm reached an accuracy of over 98%, with little to no missclassification rate. Regarding headcount reduction, that is, the amount of time saved by the model in contrast to manual classification, an operator has to inspect a minimum of 7 wafer lots per week, and this inspection process takes up to 30 minutes per lot. On the other hand, our model requires only a few seconds for lot inspection, which leads to a weekly headcount reduction of 2 hours and 30 minutes per operator. Given these results, we selected the EfficientNet B1 model as our final choice for implementation in the production line.

Max. size	40000
Training mode	one-step
Nº Epochs	100
lr_{min}	10^{-5}
lr_{max}	1
Nº steps lr	100
lr_{decay}	0
Optimizer	RAdam
Patience stop	15
$lr_{patience}$	8
lr_{factor}	0.33
Callback metric	val_acc
Class weighted loss	False
Label smoothing level	0.025

Table 4.1: Final EfficientNet B0 model hyperparameters. *Max.size* stands for the maximum number of samples allowed per class. *Training mode* is used to set how the model will be trained, it can be *one-step, two-step, frozen-base* or *pretrained-top. Nº Epochs* sets the maximum number of epochs for training, which is only reached if early stopping is not triggered. lr_{min} and lr_{max} stand for the maximum and minimum value of the learning rate search. *Nº steps lr* sets the number of steps to take in the learning rate search space. lr_{decay} is the value of the learning rate decay rate. *Optimizer* is used to specify the kind of optimizer used for model training. *Patience stop.* $lr_{patience}$. lr_{factor}. *Callback metric. Class weighted loss. Label smoothing level.*

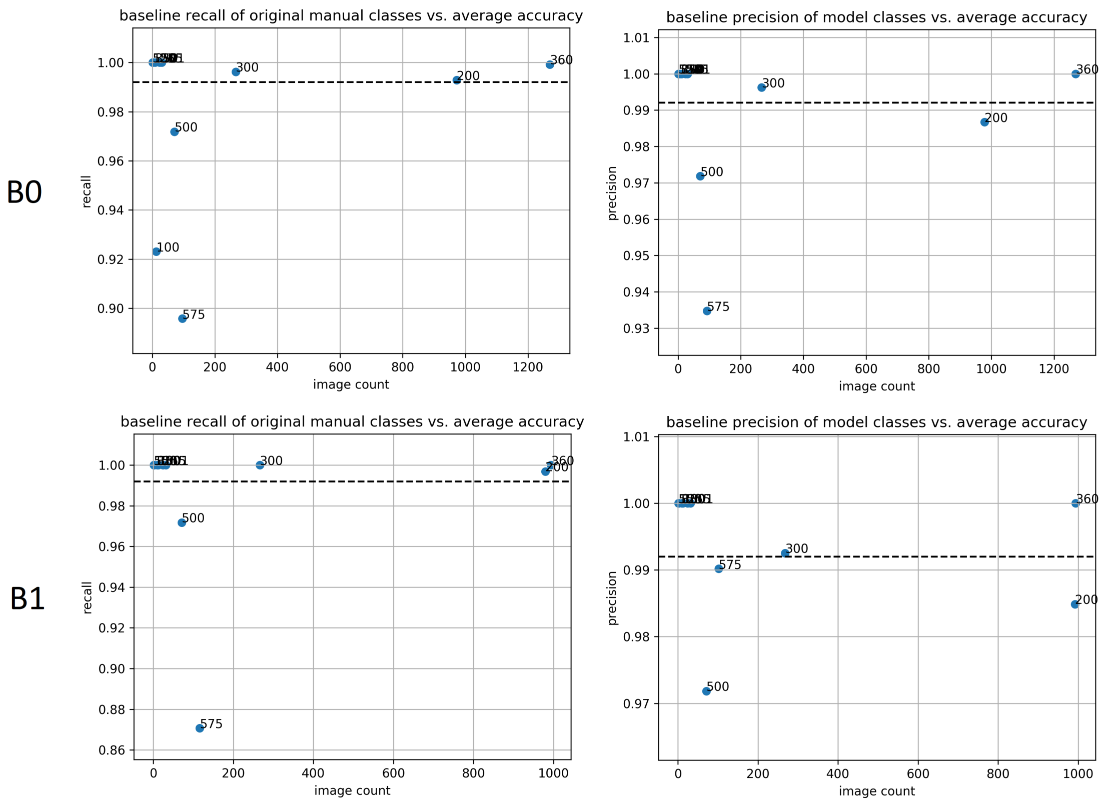


Figure 4.1: Best models precision/recall over defect classes. First row plots correspond to EfficientNet B0 architecture, second row to B1. X-axis contains the number of validation samples per class. Y-axis contains the corresponding precision/Recall.

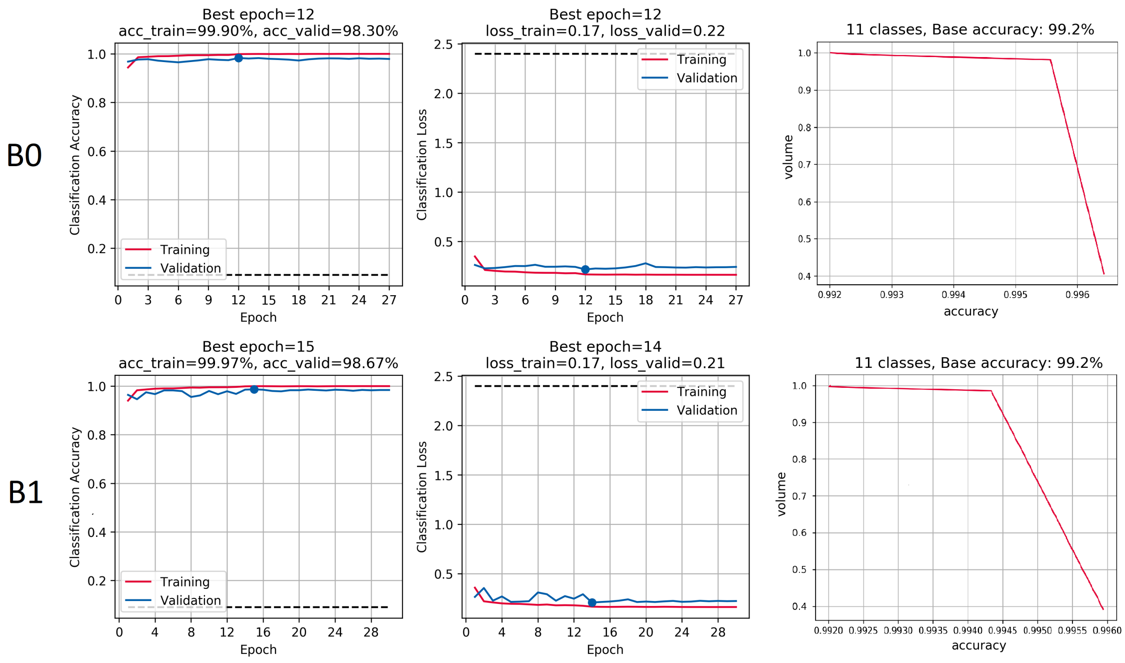


Figure 4.2: Best models training history. First row plots correspond to EfficientNet B0 architecture, second row to B1. Evolution over number of epochs of training/validation accuracy (left) and loss (middle). Model operating curve (right) with validation confidence in the X-axis and proportion of the validation images predicted with such confidence on the Y-axis.

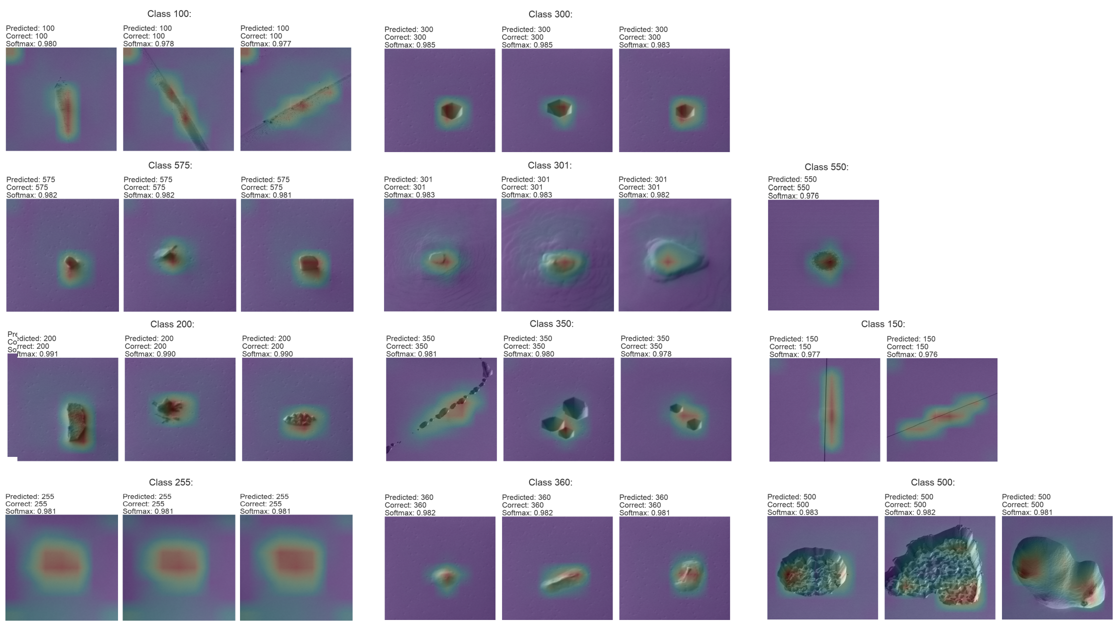


Figure 4.3: Final model activation heatmaps. EfficientNet B1 architecture. Each heatmap indicates at the top: *Predicted*, model prediction based on the sample, *Correct*, actual label of the sample, and *Softmax*, confidence of the prediction.

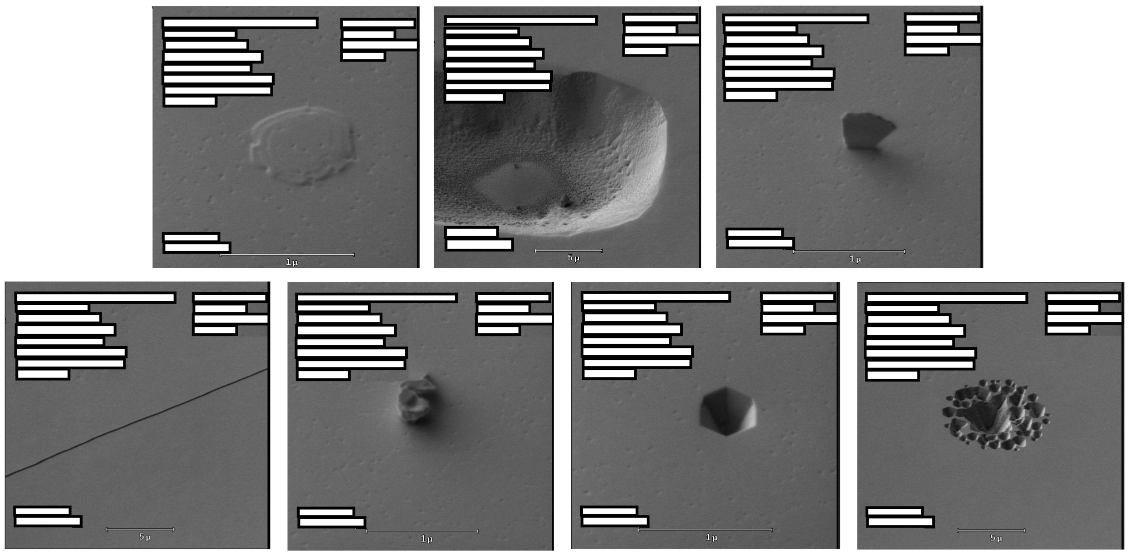


Figure 4.4: Illustration of noisy samples from the testing dataset. From left to right: class 360, 500, 575, 150, 200, 300 and 350. Labels have been concealed since they contained sensitive information.

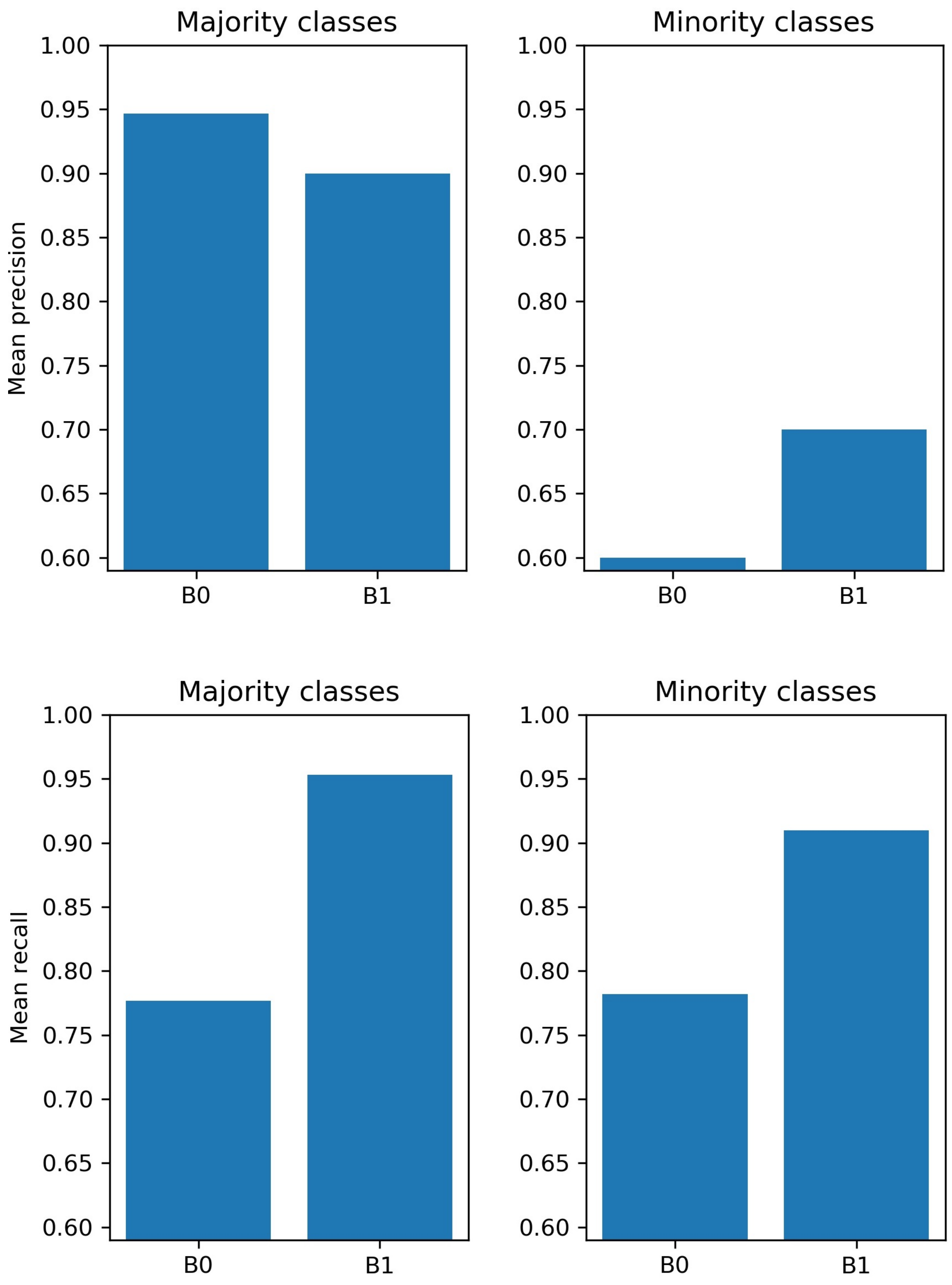


Figure 4.5: Final model testing acccuracy for raw dataset. Investigation on EfficientNet B0 and B1 architecture. Mean precision, first row, and recall, second row, over majority and minority classes, Figure 3.9. See Figure 6.17 for further details

5 Discussion

Data imbalances and their treatment through data augmentation techniques, mainly by oversampling, was the main focus of this work. However, the geometrical augmentations applied led to defect duplication and even deformation for some of the classes. Though this event does not seem to produce losses in model performance, which further proves the robustness of the EfficientNet family, it is recommended to avoid generating this kind of artificial samples.
The algorithm developed for this thesis applied the same set of transformations, with the same magnitudes, to all defect classes. However, as stated before, these transformations lead to feature degradation for some of these classes. Therefore, it would stand to reason to develop a more customized version of the algorithm used, which would use different magnitudes according to the class being oversampled.
Additionally, previous studies trained independent models that used SEM channels 2 and 3 respectively, which attained identical performance to those based only on channel 1. However, a combination of all three channel images into one same image may provide further insight into the sample's features. Since all three channels share the exact same defect location, blending is a plausible approach. Another alternative was to grayscale each SEM channel image and then build a new sample where each RGB channel is composed of one of the grayscaled images.
A few samples contain more than one defect at once. These images are often seen in the production line, and more than once these defects share a causal relationship. For example, class 360 defects make wafers more susceptible to the appearance of cracks, and therefore tend to appear together. Though not as frequent as individual defects, the presence of these kind of samples made us wonder whether the use of a multi-label classification model would prove more efficient for production than the current multi-classification technique. A more detailed study of the frequency of multiple defect images and consequent training of new models would provide further information to optimize defect classification.
In the production, defects are detected automatically by the equipment, and the presence of no-defect samples are a consequence of its lack of precision. Unfortunately, since these

kind of images are unavoidable during wafer inspection, the inclusion of such class was the most reasonable step to ensure proper classification. Ideally, only relevant samples should be detected by the equipment, and working on its improvement would speed up wafer evaluation.
Furthermore, images' bottom label cannot be disabled during sampling, so we need to crop them out during preprocessing. This is fairly easy, but not optimal, since we remove a section of the sample that might contain relevant features. The study of other alternatives, such as automatic detection and replacement of the label with background information, remains to be done. However, enabling label removal during sampling would prove the best solution.
Originally, we planned on investigating the first five versions of the EfficientNet family. However, due to hardware limitations, not only did we have to reduce significantly both the size of the training set and the batch size, but also discard the EfficientNet B5 architecture, which was of particular interest to us, since its input dimensions were closest to those of our samples. A proper study of these architectures with the same set of parameters as our final models could lead to higher performance classifiers.
Finally, knowledge distillation [32] was considered in order to train higher order architectures. However, due to lack of data, we were unable to see this process through. Investigation of *Master-Student* learning as a means to improve generalization remains to be done, and could potentially lead to higher performance in defect classification.
In conclusion, this research aimed to determine the viability of oversampling as a means to optimize minority class wafer defect classification. Using the EfficientNet family as our main architecture, we proved our oversampling strategies do improve model generalization, with results that indicated consistent increases in class recall when implemented, as well as significant robustness when facing noisy samples. Additionally, our model was considerably smaller than previous architectures considered for defect classification, such as Xception [1], as well as noticeably fast in comparison with manual inspection. While the model required approximately 5 minutes in order to completely inspect a wafer lot, an operator needed up to 30 minutes in order to fulfill the same task. Thus, we obtained a headcount reduction of over 2 hours per week when applying automated classification, reducing the workload on the operators, which can use this saved-up time to focus on other operations.

Bibliography

[1] Francois Chollet. Xception: Deep Learning with Depthwise Separable Convolutions. (page 20, 29, 104)

[2] Frank Rosenblatt. The perceptron perceiving and recognizing automaton. Technical report, Cornell Aeronautical Laboratory, 1957. (page 7, 24)

[3] Zafeirios Fountas. Imperial College Spiking Neural Networks for Human-like Avatar Control in a Simulated Environment. (page 7, 24)

[4] Sebastian Raschka. What is the Role of the Activation Function in a Neural Network? (page 7, 26)

[5] MissingLink.ai. Convolutional Neural Network Architecture: Forging Pathways to the Future. (page 7, 28)

[6] Nicole Gleichmann. SEM vs TEM. (page 8, 47)

[7] Divyanshu Mishra. Demystifying Convolutional Neural Networks Using Class Activation Maps. (page 9, 56)

[8] ODSC - Open Data Science. Visualizing Your Convolutional Neural Network Predictions With Saliency Maps (page 9, 57)

[9] Robert Legenstein, Institute of Theoretical Computer Science. Neural Networks training introduction. (page 7, 33)

[10] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song & Serge Belongie. Class-Balanced Loss Based on Effective Number of Samples. (page 53, 63)

[11] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh & Dhruv Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. (page 58)

[12] https://towardsdatascience.com/demystifying-convolutional-neural-networks-using-gradcam-554a85dd4e48 (page)

[13] Mingxing Tan & Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. (page 7, 20, 29, 30)

[14] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. CVPR, pp. 770–778, 2016. (page 29)

[15] Zagoruyko, S. and Komodakis, N. Wide residual networks. BMVC, 2016. (page 29)

[16] Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J., Le, Q. V., and Chen, Z. Gpipe: Efficient training of giant neural networks using pipeline parallelism. arXiv preprint arXiv:1808.07233, 2018. (page 29)

[17] Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. Learning deep features for discriminative localization. CVPR, pp. 2921–2929, 2016. (page 30)

[18] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna. Le. Rethinking the Inception Architecture for Computer Vision (page 20, 29)

[19] Fukushima, K. (2007). ”Neocognitron”. Scholarpedia. 2 (1): 1717. Bibcode:2007SchpJ...2.1717F. doi:10.4249/scholarpedia.1717. (page 28)

[20] Hubel, D. H.; Wiesel, T. N. (1968-03-01). ”Receptive fields and functional architecture of monkey striate cortex”. The Journal of Physiology. 195 (1): 215–243. doi:10.1113/jphysiol.1968.sp008455. ISSN 0022-3751. PMC 1557912. PMID 4966457. (page 28)

[21] Fukushima, Kunihiko (1980). ”Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position” (PDF). Biological Cybernetics. 36 (4): 193–202. doi:10.1007/BF00344251. PMID 7370364. S2CID 206775608. Retrieved 16 November 2013. (page 28)

[22] Karen Simonyan, Andrea Vedaldi, Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps (2014) (page 57, 58)

[23] Matusugu, Masakazu; Katsuhiko Mori; Yusuke Mitari; Yuji Kaneda (2003). ”Subject independent facial expression recognition with robust face detection using a convolutional neural network” (PDF). Neural Networks. 16 (5): 555–559. doi:10.1016/S0893-6080(03)00115-1. PMID 12850007. Retrieved 17 November 2013. (page 28)

[24] Leaderboard: Image classification on imagenet. https://tinyurl.com/yygsyy8q. (page 21)

[25] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In Proc. ECCV, 2016. (page 21)

[26] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten. Exploring the limits of weakly supervised pretraining. ECCV, 2018. (page 21)

[27] S. J. Pan and Q. Yang. A survey on transfer learning. TPAMI, 22(10):1345–1359, 2010. (page 21)

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In nips, 2012. (page 21)

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition (page 29, 30)

[30] Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J., Le, Q. V., and Chen, Z. Gpipe: Efficient training of giant neural networks using pipeline parallelism. arXiv preprint arXiv:1808.07233, 2018. (page 21, 30)

[31] Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition (page 20, 29)

[32] Qizhe Xie, Minh-Thang Luong, Eduard Hovy & Quoc V. Le. Self-training with Noisy Student improves ImageNet classification. (page 104)

[33] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens & Quoc V. Le. RandAugment: Practical automated data augmentation with a reduced search space. (page 21, 50, 52)

[34] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. arXiv:1710.09412, 2017. (page 21)

[35] https://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html (page 50)

[36] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan & Quoc V. Le. AutoAugment: Learning Augmentation Policies from Data. (page 21, 50, 52)

[37] R. Müller, S. Kornblith, and G. Hinton. When Does Label Smoothing Help? (2019), NeurIPS 2019. (page 54, 80)

[38] Mingxing Tan, Bo Chen, Quoc V. Le, Ruoming Pang & Vijay Vasudevan. MnasNet: Towards Automating the Design of Mobile Machine Learning Models. (page 20)

[39] Geoffrey Hinton, Oriol Vinyals & Jeff Dean. Distilling the Knowledge in a Neural Network. (page)

[40] Zeki Yalniz, Herve Jegou, Kan Chen, Manohar Paluri & Dhruv Mahajan. Billion-scale semi-supervised learning for image classification. (page 29)

[41] Gregory Koch, Richard Zemel & Ruslan Salakhutdinov. Siamese Neural Networks for One-shot Image Recognition. (page 20, 22)

[42] I. Zeki Yalniz, Herve Jegou, Kan Chen Facebook AI, Manohar Paluri, Dhruv Mahajan. Billion-scale semi-supervised learning for image classification. (page 21)

[43] Stephen I. Gallant. Perceptron-Based Learning Algorithms. (page 20, 23)

[44] Mohammad Hadi Modarres, Rossella Aversa, Stefano Cozzini, Regina Ciancio, Angelo Leto & Giuseppe Piero Brandino. Neural Network for Nanoscience Scanning Electron Microscope Image Recognition. (page 20)

[45] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao & Jiawei Han. On the Variance of the Adaptive Learning Rate and Beyond. (page 7, 37, 39, 80)

[46] Diederik Kingma, Jimmy Ba. Adam: A Method for Stochastic Optimization. (page 35)

[47] Y. Fan, F. Tian, T. Qin, J. Bian, and T. Liu. Learning to teach. In ICLR, 2018. (page 21)

[48] C. Gong, D. Tao, J. Yang, and W. Liu. Teaching-to-learn and learning-to-teach for multi-label propagation. In AAAI, 2016. (page 21)

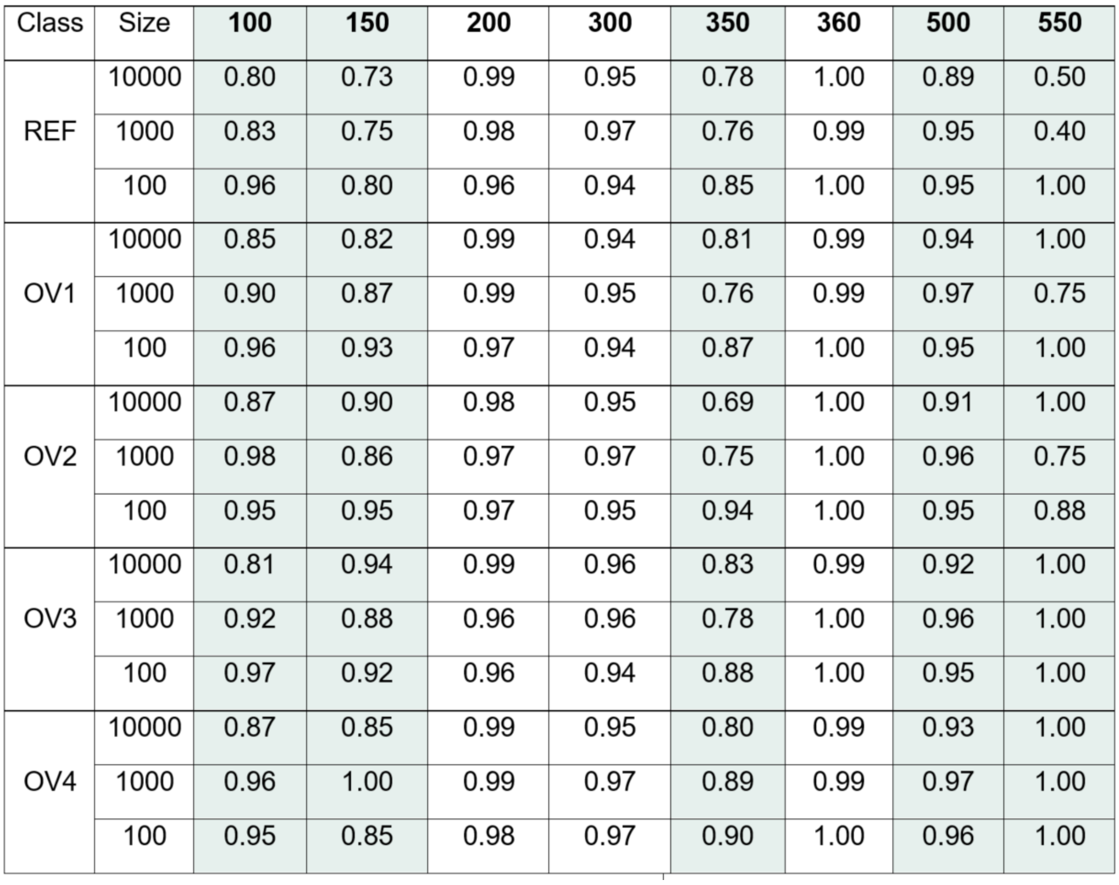
[49] L. Jiang, Z. Zhou, T. Leung, L. Li, and L. Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In ICML, 2018. (page 21)

[50] Li Fei-Fei, Member, IEEE, Rob Fergus, Student Member, IEEE, and Pietro Perona, Member, IEEE. One-Shot Learning of Object Categories.. (page 22)

[51] https://www.halbleiter.org/en/waferfabrication/ (page 7, 8, 39, 41, 42, 43)

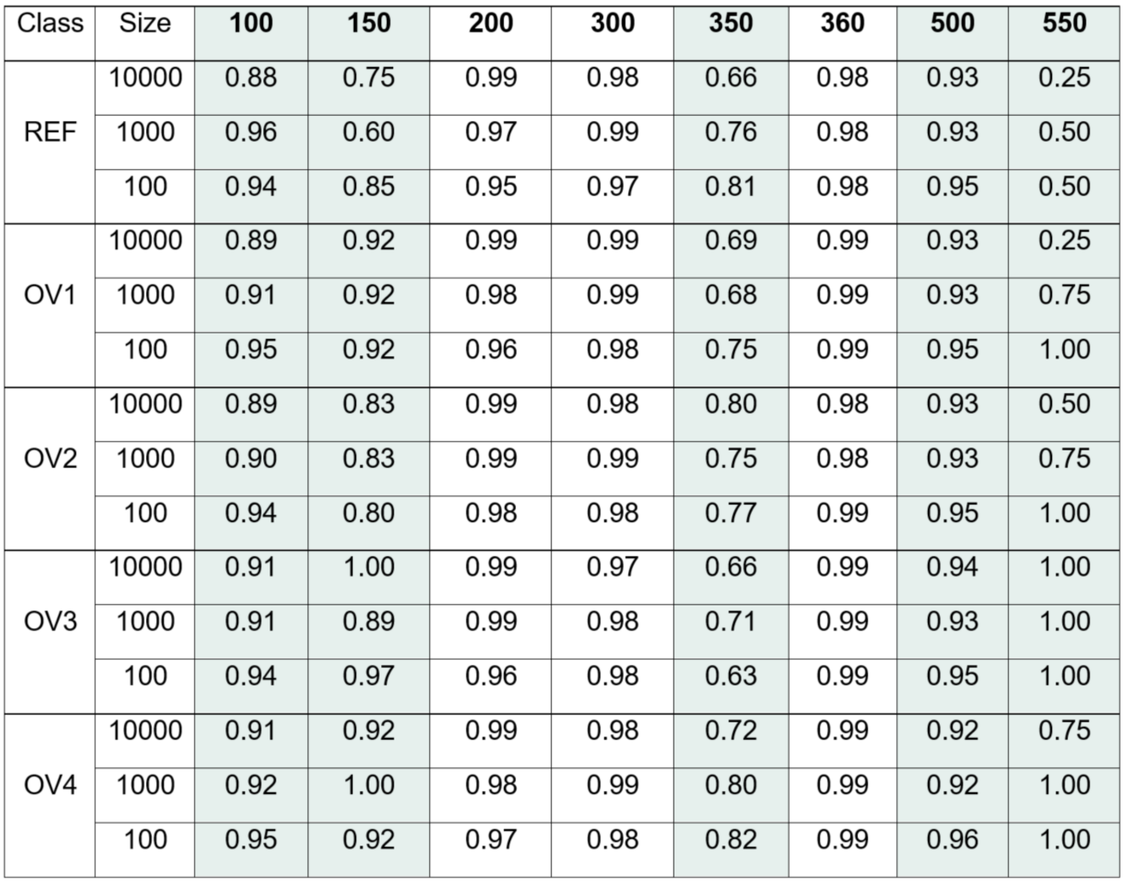
[52] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg & Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge (page 20)

6 Appendix



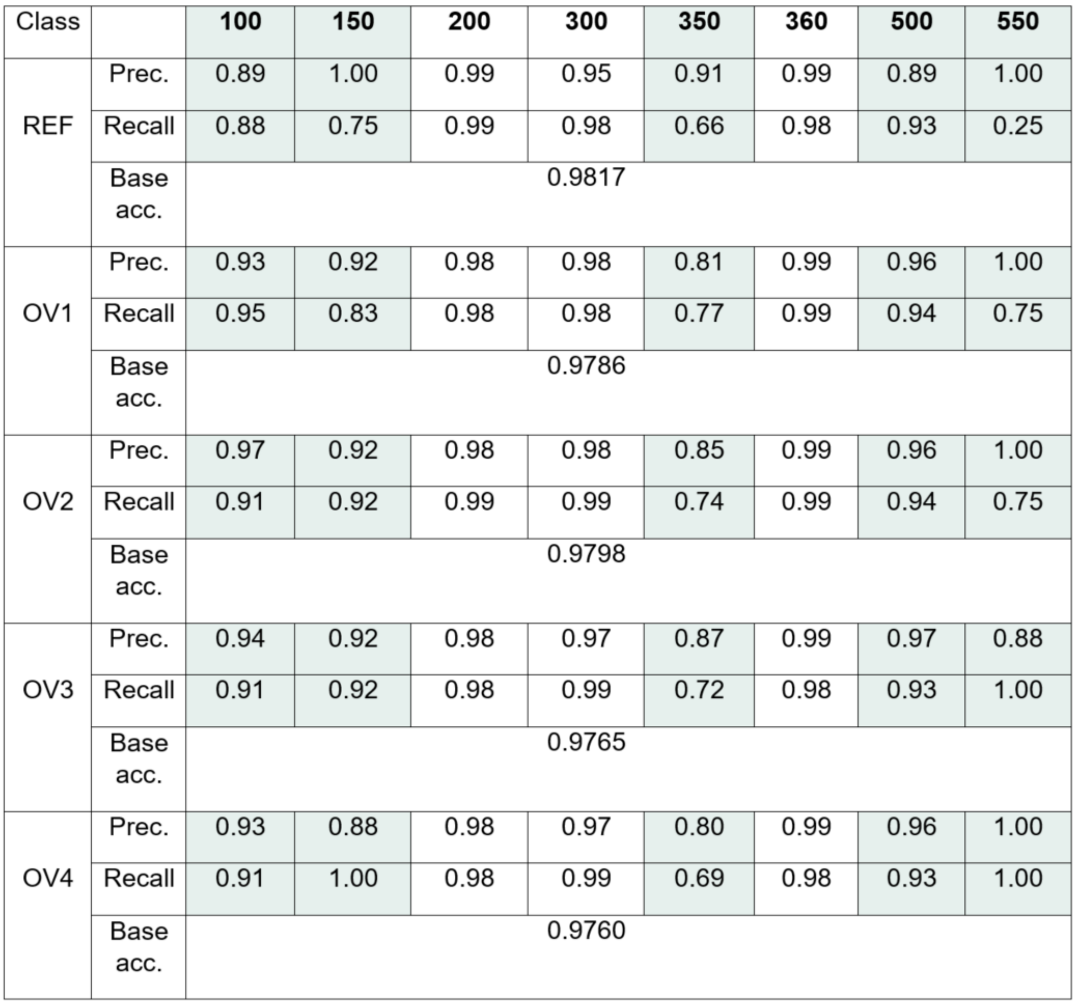
Class	Size	100	150	200	300	350	360	500	550
REF	10000	0.80	0.73	0.99	0.95	0.78	1.00	0.89	0.50
	1000	0.83	0.75	0.98	0.97	0.76	0.99	0.95	0.40
	100	0.96	0.80	0.96	0.94	0.85	1.00	0.95	1.00
OV1	10000	0.85	0.82	0.99	0.94	0.81	0.99	0.94	1.00
	1000	0.90	0.87	0.99	0.95	0.76	0.99	0.97	0.75
	100	0.96	0.93	0.97	0.94	0.87	1.00	0.95	1.00
OV2	10000	0.87	0.90	0.98	0.95	0.69	1.00	0.91	1.00
	1000	0.98	0.86	0.97	0.97	0.75	1.00	0.96	0.75
	100	0.95	0.95	0.97	0.95	0.94	1.00	0.95	0.88
OV3	10000	0.81	0.94	0.99	0.96	0.83	0.99	0.92	1.00
	1000	0.92	0.88	0.96	0.96	0.78	1.00	0.96	1.00
	100	0.97	0.92	0.96	0.94	0.88	1.00	0.95	1.00
OV4	10000	0.87	0.85	0.99	0.95	0.80	0.99	0.93	1.00
	1000	0.96	1.00	0.99	0.97	0.89	0.99	0.97	1.00
	100	0.95	0.85	0.98	0.97	0.90	1.00	0.96	1.00

Figure 6.1: Precision over class maximum size. Mean precision out of 1-time 4-fold cross validation per hyperparameter set. Minority classes highlighted in green.



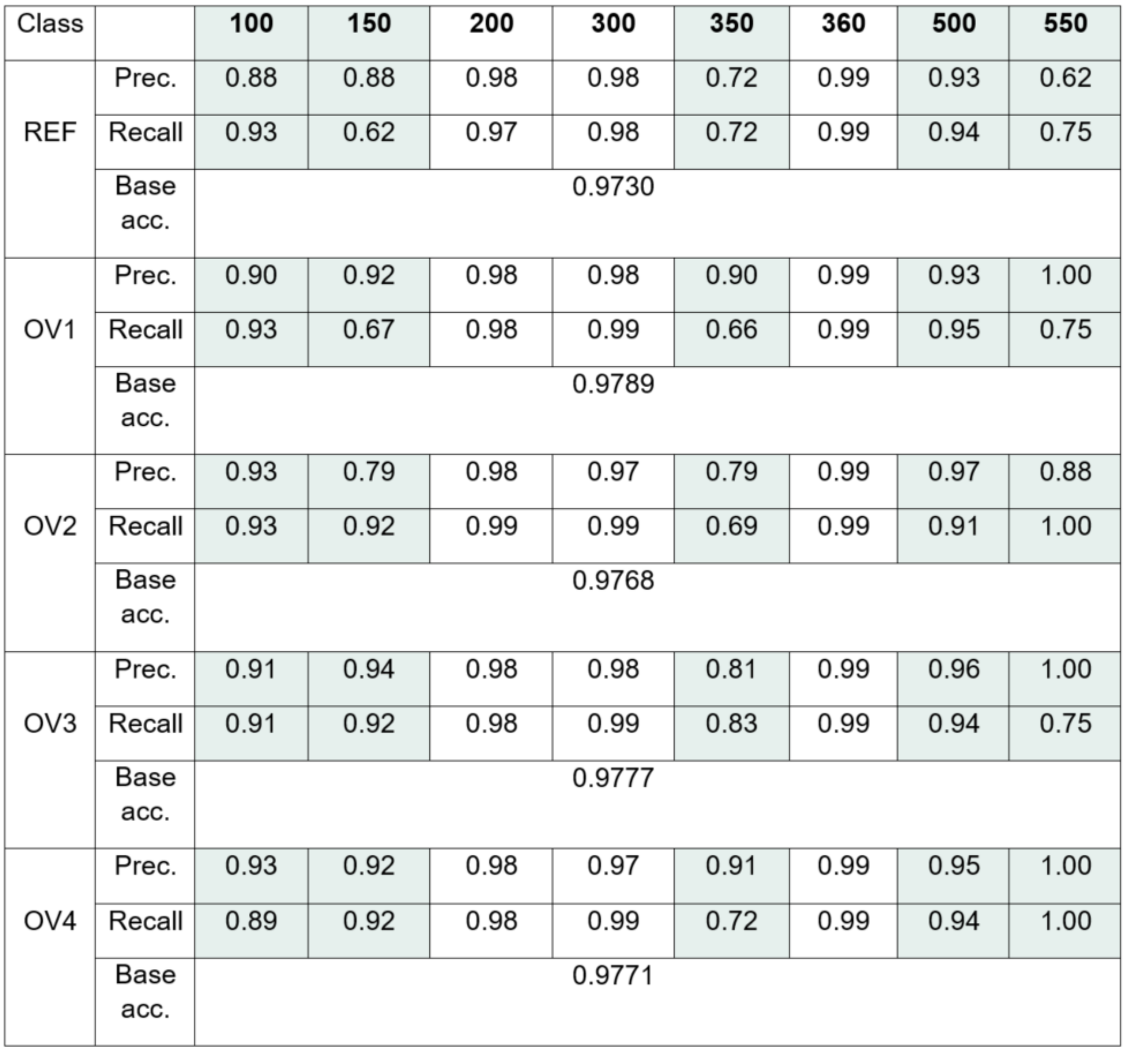
Class	Size	100	150	200	300	350	360	500	550
REF	10000	0.88	0.75	0.99	0.98	0.66	0.98	0.93	0.25
	1000	0.96	0.60	0.97	0.99	0.76	0.98	0.93	0.50
	100	0.94	0.85	0.95	0.97	0.81	0.98	0.95	0.50
OV1	10000	0.89	0.92	0.99	0.99	0.69	0.99	0.93	0.25
	1000	0.91	0.92	0.98	0.99	0.68	0.99	0.93	0.75
	100	0.95	0.92	0.96	0.98	0.75	0.99	0.95	1.00
OV2	10000	0.89	0.83	0.99	0.98	0.80	0.98	0.93	0.50
	1000	0.90	0.83	0.99	0.99	0.75	0.98	0.93	0.75
	100	0.94	0.80	0.98	0.98	0.77	0.99	0.95	1.00
OV3	10000	0.91	1.00	0.99	0.97	0.66	0.99	0.94	1.00
	1000	0.91	0.89	0.99	0.98	0.71	0.99	0.93	1.00
	100	0.94	0.97	0.96	0.98	0.63	0.99	0.95	1.00
OV4	10000	0.91	0.92	0.99	0.98	0.72	0.99	0.92	0.75
	1000	0.92	1.00	0.98	0.99	0.80	0.99	0.92	1.00
	100	0.95	0.92	0.97	0.98	0.82	0.99	0.96	1.00

Figure 6.2: Recall over class maximum size. Mean recall out of 1-time 4-fold cross validation per hyperparameter set. Minority classes highlighted in green.



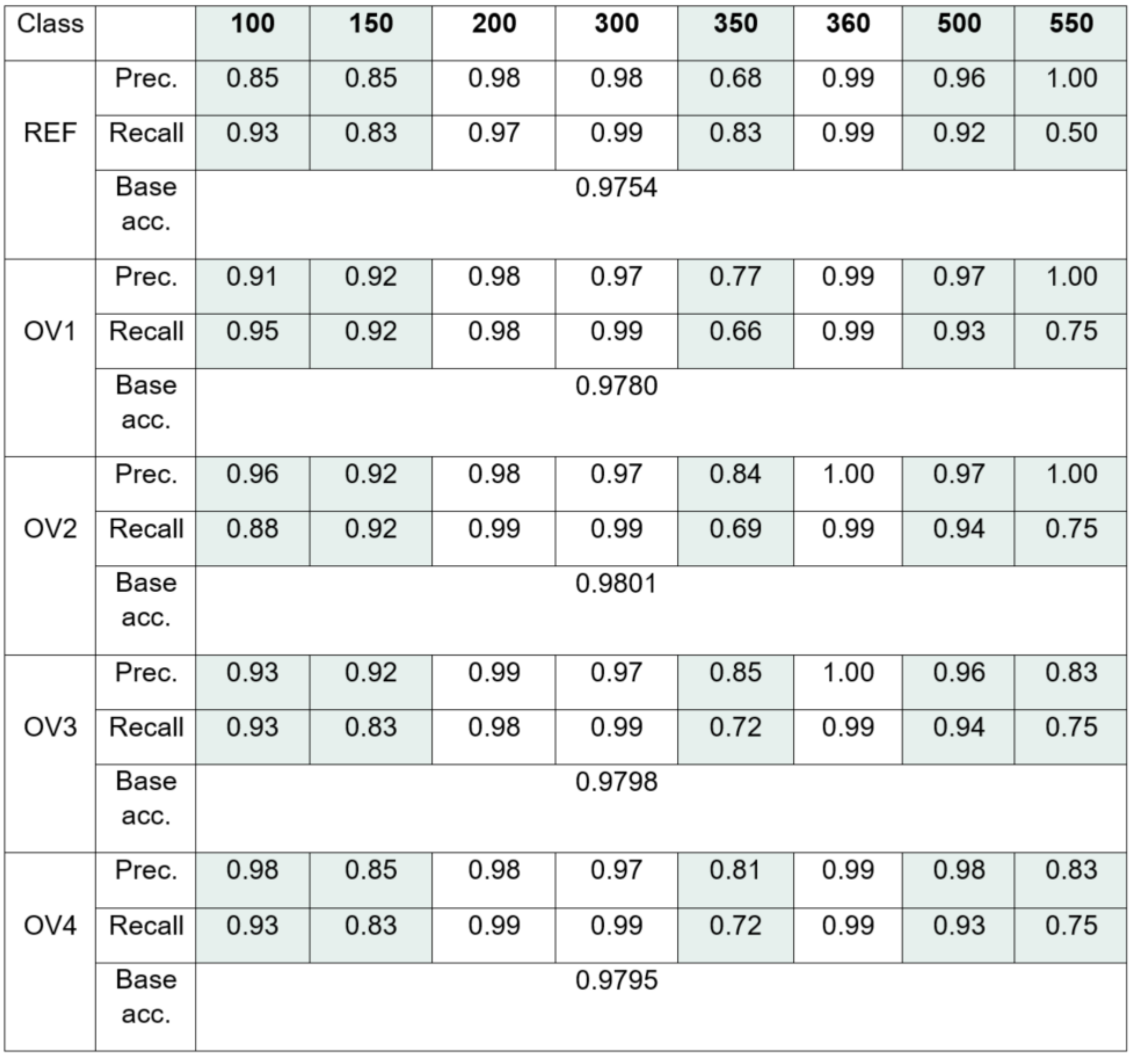
Class		100	150	200	300	350	360	500	550
REF	Prec.	0.89	1.00	0.99	0.95	0.91	0.99	0.89	1.00
	Recall	0.88	0.75	0.99	0.98	0.66	0.98	0.93	0.25
	Base acc.	0.9817							
OV1	Prec.	0.93	0.92	0.98	0.98	0.81	0.99	0.96	1.00
	Recall	0.95	0.83	0.98	0.98	0.77	0.99	0.94	0.75
	Base acc.	0.9786							
OV2	Prec.	0.97	0.92	0.98	0.98	0.85	0.99	0.96	1.00
	Recall	0.91	0.92	0.99	0.99	0.74	0.99	0.94	0.75
	Base acc.	0.9798							
OV3	Prec.	0.94	0.92	0.98	0.97	0.87	0.99	0.97	0.88
	Recall	0.91	0.92	0.98	0.99	0.72	0.98	0.93	1.00
	Base acc.	0.9765							
OV4	Prec.	0.93	0.88	0.98	0.97	0.80	0.99	0.96	1.00
	Recall	0.91	1.00	0.98	0.99	0.69	0.98	0.93	1.00
	Base acc.	0.9760							

Figure 6.3: Model performance without weight loss. 1-time 4-fold cross-validation per hyperparameter set. Maximum class size set to 1000. Minority classes contained in light green boxes.



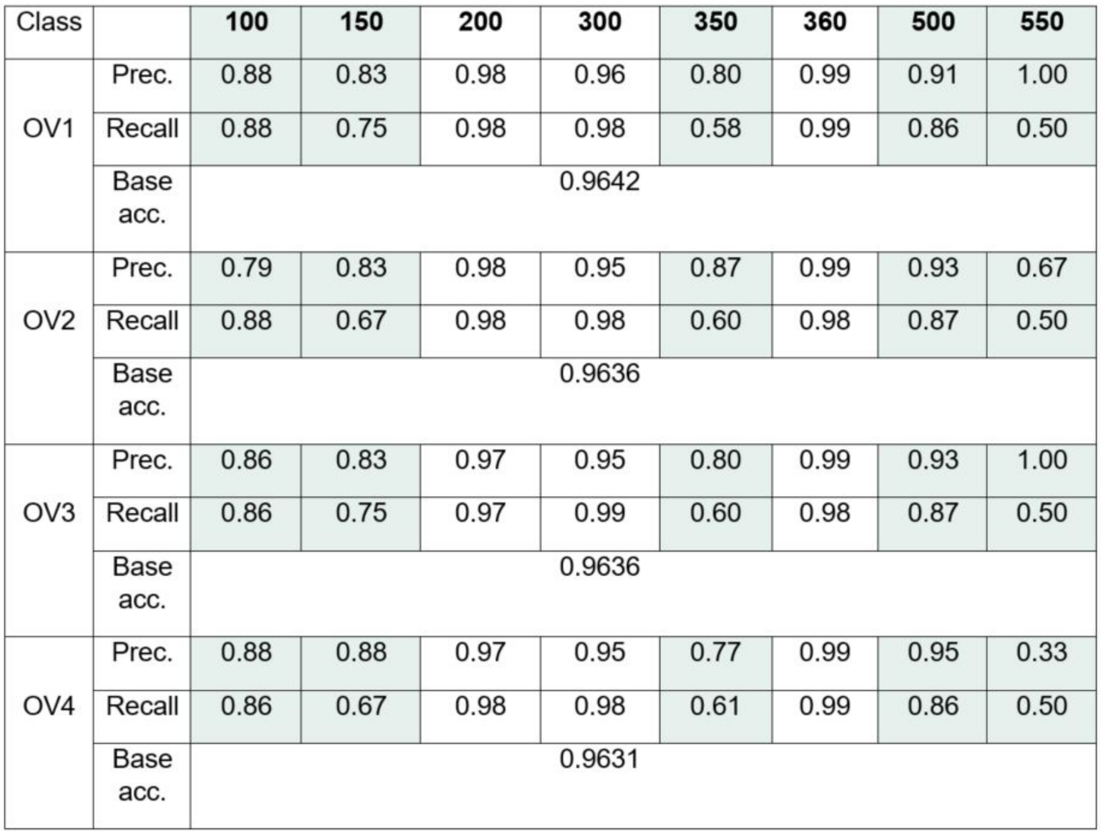
Class		100	150	200	300	350	360	500	550
REF	Prec.	0.88	0.88	0.98	0.98	0.72	0.99	0.93	0.62
	Recall	0.93	0.62	0.97	0.98	0.72	0.99	0.94	0.75
	Base acc.	0.9730							
OV1	Prec.	0.90	0.92	0.98	0.98	0.90	0.99	0.93	1.00
	Recall	0.93	0.67	0.98	0.99	0.66	0.99	0.95	0.75
	Base acc.	0.9789							
OV2	Prec.	0.93	0.79	0.98	0.97	0.79	0.99	0.97	0.88
	Recall	0.93	0.92	0.99	0.99	0.69	0.99	0.91	1.00
	Base acc.	0.9768							
OV3	Prec.	0.91	0.94	0.98	0.98	0.81	0.99	0.96	1.00
	Recall	0.91	0.92	0.98	0.99	0.83	0.99	0.94	0.75
	Base acc.	0.9777							
OV4	Prec.	0.93	0.92	0.98	0.97	0.91	0.99	0.95	1.00
	Recall	0.89	0.92	0.98	0.99	0.72	0.99	0.94	1.00
	Base acc.	0.9771							

Figure 6.4: Model performance after label removal. 1-time 4-fold cross-validation per hyperparameter set. Maximum class size set to 1000. Minority classes contained in light green boxes.



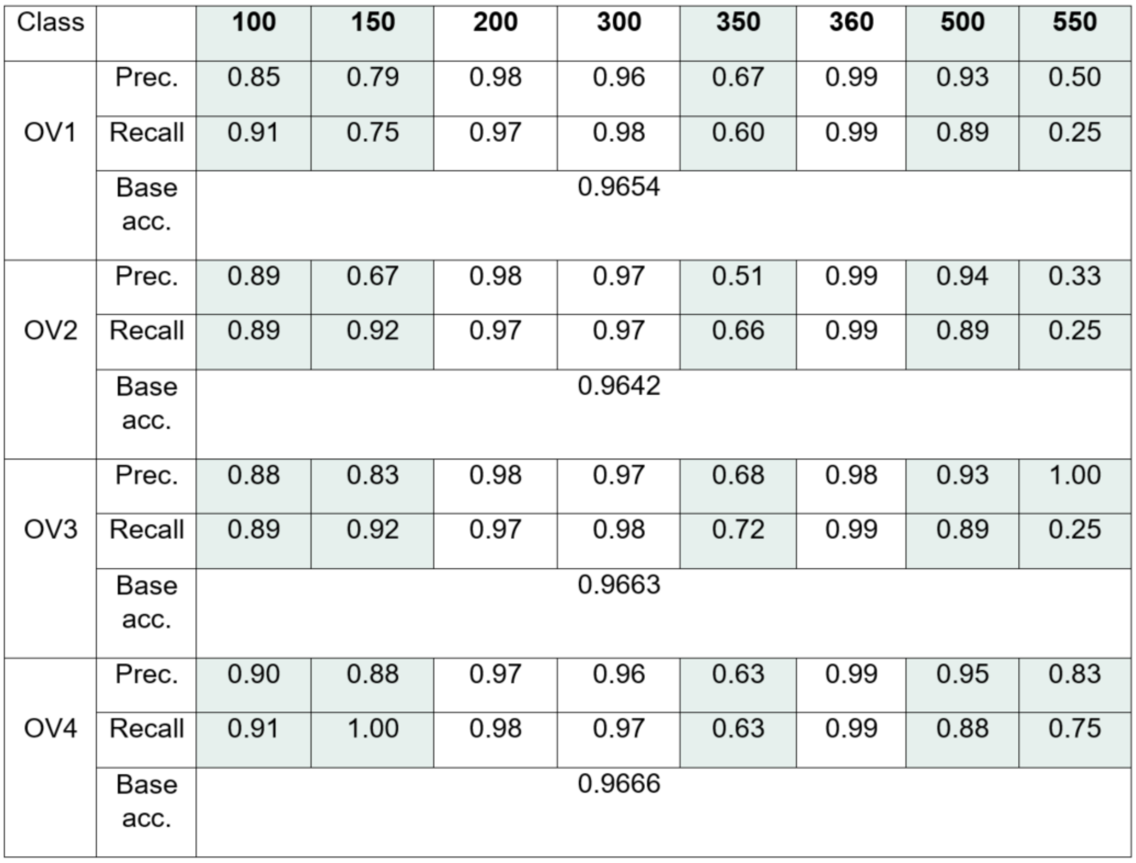
Class		100	150	200	300	350	360	500	550
REF	Prec.	0.85	0.85	0.98	0.98	0.68	0.99	0.96	1.00
	Recall	0.93	0.83	0.97	0.99	0.83	0.99	0.92	0.50
	Base acc.	0.9754							
OV1	Prec.	0.91	0.92	0.98	0.97	0.77	0.99	0.97	1.00
	Recall	0.95	0.92	0.98	0.99	0.66	0.99	0.93	0.75
	Base acc.	0.9780							
OV2	Prec.	0.96	0.92	0.98	0.97	0.84	1.00	0.97	1.00
	Recall	0.88	0.92	0.99	0.99	0.69	0.99	0.94	0.75
	Base acc.	0.9801							
OV3	Prec.	0.93	0.92	0.99	0.97	0.85	1.00	0.96	0.83
	Recall	0.93	0.83	0.98	0.99	0.72	0.99	0.94	0.75
	Base acc.	0.9798							
OV4	Prec.	0.98	0.85	0.98	0.97	0.81	0.99	0.98	0.83
	Recall	0.93	0.83	0.99	0.99	0.72	0.99	0.93	0.75
	Base acc.	0.9795							

Figure 6.5: Model performance with background reflection. 1-time 4-fold cross-validation per hyperparameter set. Maximum class size set to 1000. Minority classes contained in light green boxes.



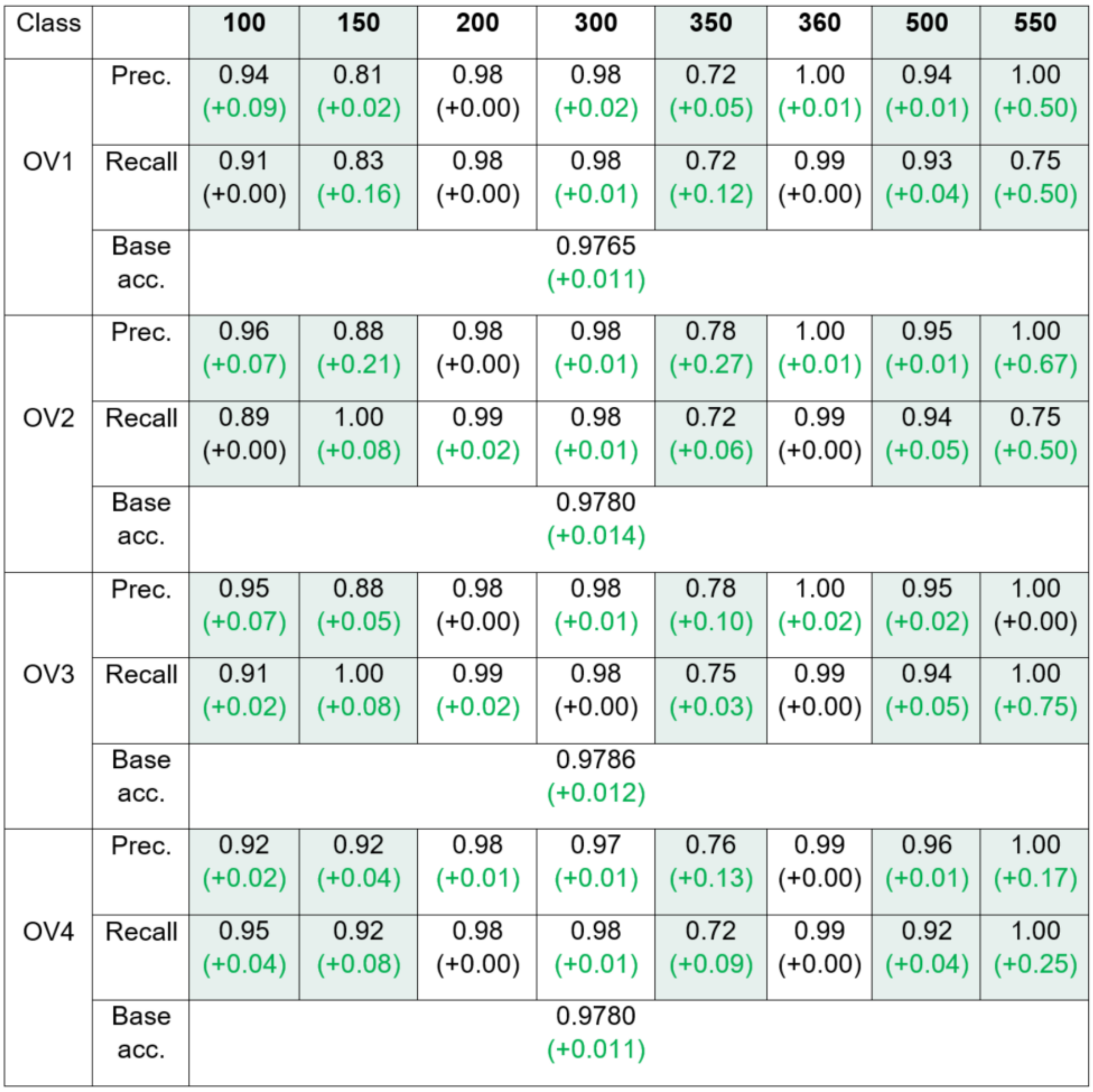
Class		**100**	**150**	**200**	**300**	**350**	**360**	**500**	**550**
OV1	Prec.	0.88	0.83	0.98	0.96	0.80	0.99	0.91	1.00
	Recall	0.88	0.75	0.98	0.98	0.58	0.99	0.86	0.50
	Base acc.				0.9642				
OV2	Prec.	0.79	0.83	0.98	0.95	0.87	0.99	0.93	0.67
	Recall	0.88	0.67	0.98	0.98	0.60	0.98	0.87	0.50
	Base acc.				0.9636				
OV3	Prec.	0.86	0.83	0.97	0.95	0.80	0.99	0.93	1.00
	Recall	0.86	0.75	0.97	0.99	0.60	0.98	0.87	0.50
	Base acc.				0.9636				
OV4	Prec.	0.88	0.88	0.97	0.95	0.77	0.99	0.95	0.33
	Recall	0.86	0.67	0.98	0.98	0.61	0.99	0.86	0.50
	Base acc.				0.9631				

Figure 6.6: Model performance after grayscale reduction. 1-time 4-fold cross-validation per hyperparameter set. Maximum class size set to 1000. Minority classes contained in light green boxes.



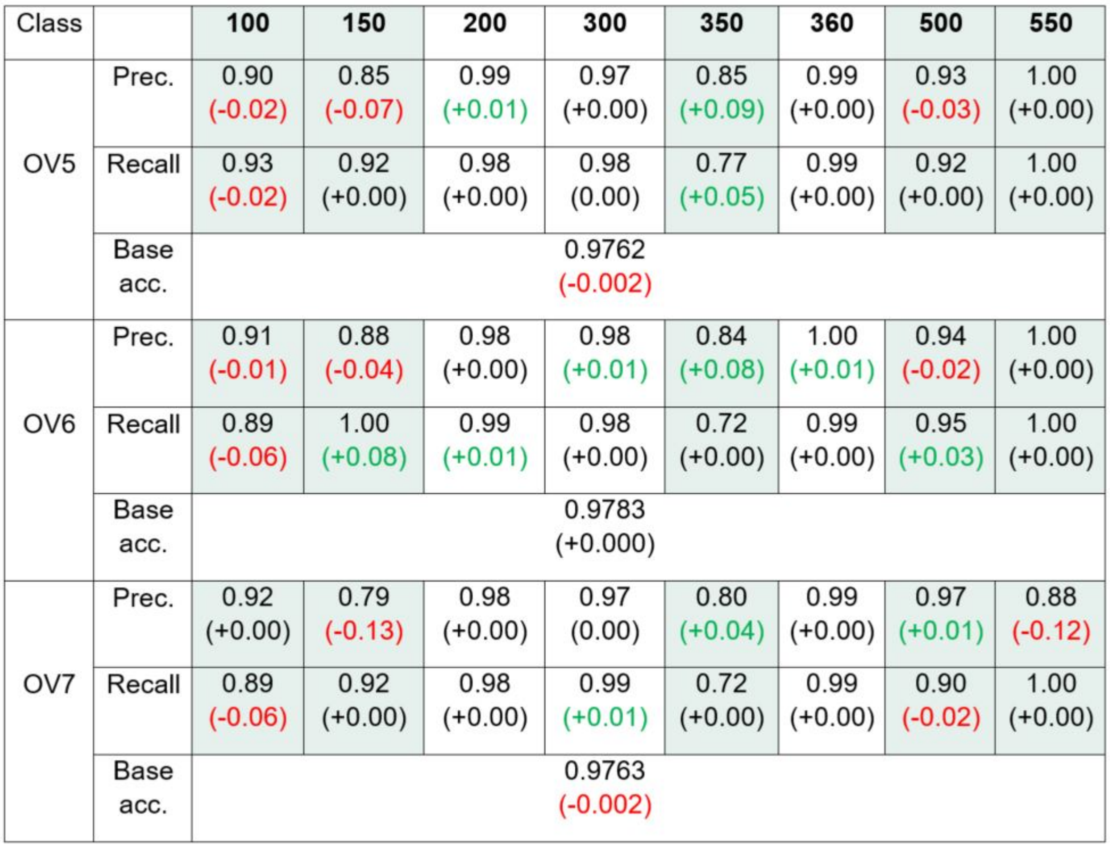
Class		100	150	200	300	350	360	500	550
OV1	Prec.	0.85	0.79	0.98	0.96	0.67	0.99	0.93	0.50
	Recall	0.91	0.75	0.97	0.98	0.60	0.99	0.89	0.25
	Base acc.				0.9654				
OV2	Prec.	0.89	0.67	0.98	0.97	0.51	0.99	0.94	0.33
	Recall	0.89	0.92	0.97	0.97	0.66	0.99	0.89	0.25
	Base acc.				0.9642				
OV3	Prec.	0.88	0.83	0.98	0.97	0.68	0.98	0.93	1.00
	Recall	0.89	0.92	0.97	0.98	0.72	0.99	0.89	0.25
	Base acc.				0.9663				
OV4	Prec.	0.90	0.88	0.97	0.96	0.63	0.99	0.95	0.83
	Recall	0.91	1.00	0.98	0.97	0.63	0.99	0.88	0.75
	Base acc.				0.9666				

Figure 6.7: Model performance with grayscale. 1-time 4-fold cross-validation per hyperparameter set. Maximum class size set to 1000. Minority classes contained in light green boxes.



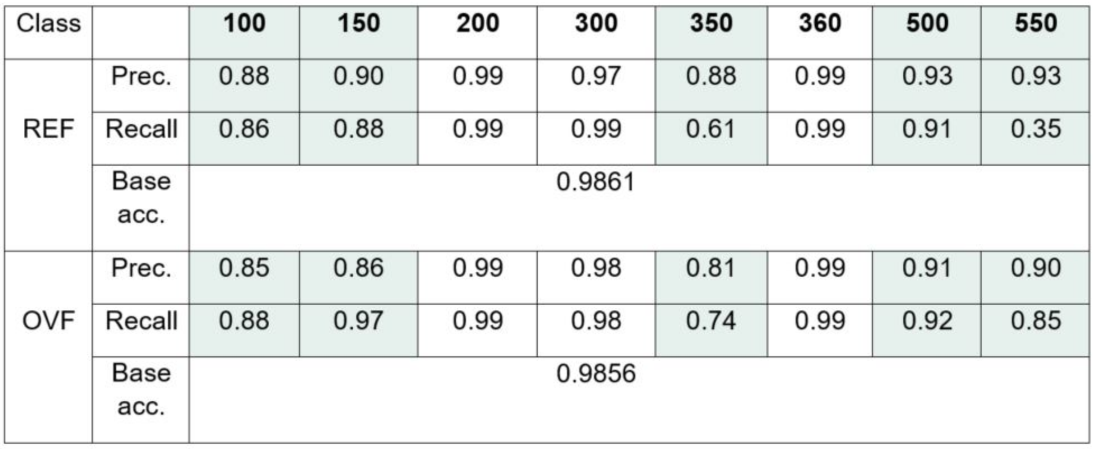
Class		**100**	**150**	**200**	**300**	**350**	**360**	**500**	**550**
OV1	Prec.	0.94 (+0.09)	0.81 (+0.02)	0.98 (+0.00)	0.98 (+0.02)	0.72 (+0.05)	1.00 (+0.01)	0.94 (+0.01)	1.00 (+0.50)
	Recall	0.91 (+0.00)	0.83 (+0.16)	0.98 (+0.00)	0.98 (+0.01)	0.72 (+0.12)	0.99 (+0.00)	0.93 (+0.04)	0.75 (+0.50)
	Base acc.	0.9765 (+0.011)							
OV2	Prec.	0.96 (+0.07)	0.88 (+0.21)	0.98 (+0.00)	0.98 (+0.01)	0.78 (+0.27)	1.00 (+0.01)	0.95 (+0.01)	1.00 (+0.67)
	Recall	0.89 (+0.00)	1.00 (+0.08)	0.99 (+0.02)	0.98 (+0.01)	0.72 (+0.06)	0.99 (+0.00)	0.94 (+0.05)	0.75 (+0.50)
	Base acc.	0.9780 (+0.014)							
OV3	Prec.	0.95 (+0.07)	0.88 (+0.05)	0.98 (+0.00)	0.98 (+0.01)	0.78 (+0.10)	1.00 (+0.02)	0.95 (+0.02)	1.00 (+0.00)
	Recall	0.91 (+0.02)	1.00 (+0.08)	0.99 (+0.02)	0.98 (+0.00)	0.75 (+0.03)	0.99 (+0.00)	0.94 (+0.05)	1.00 (+0.75)
	Base acc.	0.9786 (+0.012)							
OV4	Prec.	0.92 (+0.02)	0.92 (+0.04)	0.98 (+0.01)	0.97 (+0.01)	0.76 (+0.13)	0.99 (+0.00)	0.96 (+0.01)	1.00 (+0.17)
	Recall	0.95 (+0.04)	0.92 (+0.08)	0.98 (+0.00)	0.98 (+0.01)	0.72 (+0.09)	0.99 (+0.00)	0.92 (+0.04)	1.00 (+0.25)
	Base acc.	0.9780 (+0.011)							

Figure 6.8: Model performance with RGB. 1-time 4-fold cross-validation per hyperparameter set. Maximum class size set to 1000. Minority classes contained in light green boxes. Results comparison with Figure 6.7 indicate consistent improvements (green values) for all classes



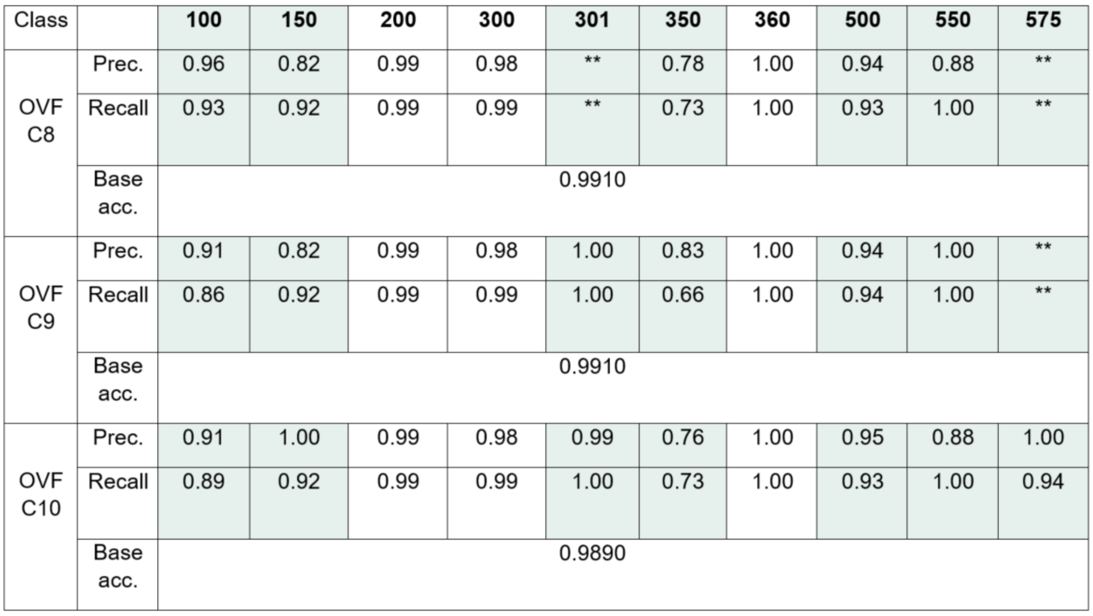
Class		100	150	200	300	350	360	500	550
OV5	Prec.	0.90 (-0.02)	0.85 (-0.07)	0.99 (+0.01)	0.97 (+0.00)	0.85 (+0.09)	0.99 (+0.00)	0.93 (-0.03)	1.00 (+0.00)
	Recall	0.93 (-0.02)	0.92 (+0.00)	0.98 (+0.00)	0.98 (0.00)	0.77 (+0.05)	0.99 (+0.00)	0.92 (+0.00)	1.00 (+0.00)
	Base acc.	0.9762 (-0.002)							
OV6	Prec.	0.91 (-0.01)	0.88 (-0.04)	0.98 (+0.00)	0.98 (+0.01)	0.84 (+0.08)	1.00 (+0.01)	0.94 (-0.02)	1.00 (+0.00)
	Recall	0.89 (-0.06)	1.00 (+0.08)	0.99 (+0.01)	0.98 (+0.00)	0.72 (+0.00)	0.99 (+0.00)	0.95 (+0.03)	1.00 (+0.00)
	Base acc.	0.9783 (+0.000)							
OV7	Prec.	0.92 (+0.00)	0.79 (-0.13)	0.98 (+0.00)	0.97 (0.00)	0.80 (+0.04)	0.99 (+0.00)	0.97 (+0.01)	0.88 (-0.12)
	Recall	0.89 (-0.06)	0.92 (+0.00)	0.98 (+0.00)	0.99 (+0.01)	0.72 (+0.00)	0.99 (+0.00)	0.90 (-0.02)	1.00 (+0.00)
	Base acc.	0.9763 (-0.002)							

Figure 6.9: Model performance with additional oversampling. 1-time 4-fold cross-validation per hyperparameter set. Maximum class size set to 1000. Minority classes contained in light green boxes. Comparison with OV4, Figure 6.8, to highlight improvements, green values, or losses, red values, in performance.



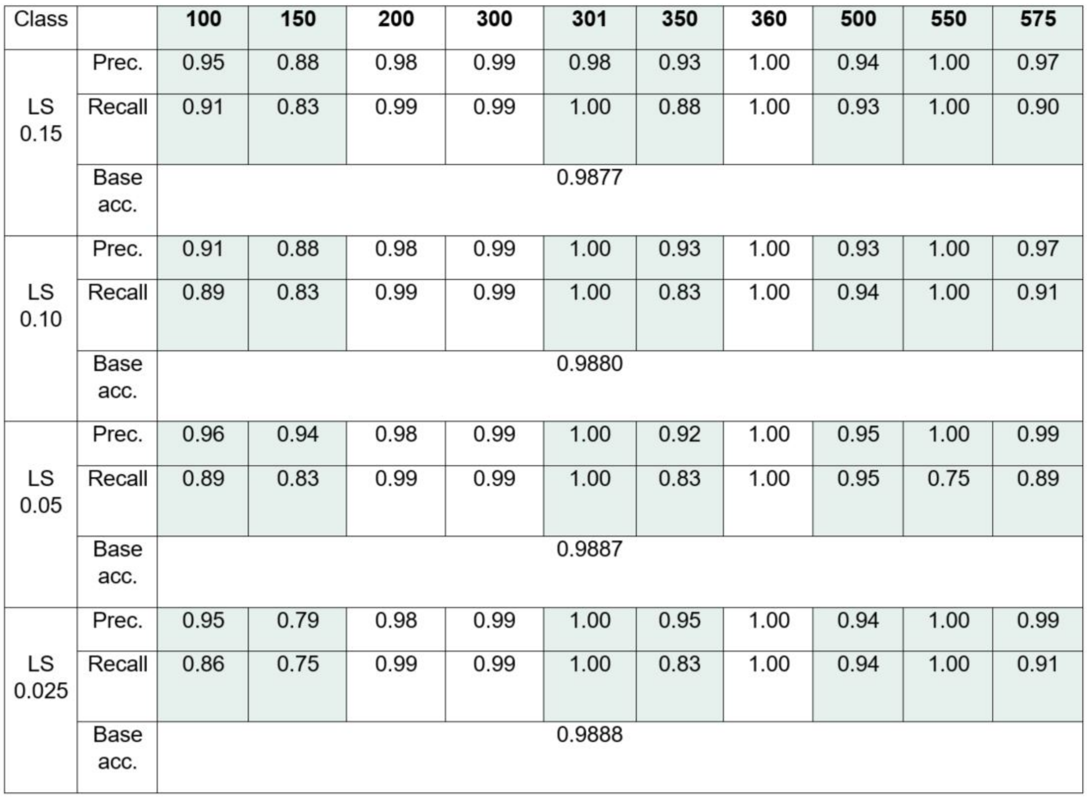
Class		100	150	200	300	350	360	500	550
REF	Prec.	0.88	0.90	0.99	0.97	0.88	0.99	0.93	0.93
	Recall	0.86	0.88	0.99	0.99	0.61	0.99	0.91	0.35
	Base acc.	0.9861							
OVF	Prec.	0.85	0.86	0.99	0.98	0.81	0.99	0.91	0.90
	Recall	0.88	0.97	0.99	0.98	0.74	0.99	0.92	0.85
	Base acc.	0.9856							

Figure 6.10: First stage model performance. 5-times 4-fold cross-validation per hyperparameter set. Maximum class size set to 5103. Minority classes contained in light green boxes.



Class		100	150	200	300	301	350	360	500	550	575
OVF C8	Prec.	0.96	0.82	0.99	0.98	**	0.78	1.00	0.94	0.88	**
	Recall	0.93	0.92	0.99	0.99	**	0.73	1.00	0.93	1.00	**
	Base acc.	0.9910									
OVF C9	Prec.	0.91	0.82	0.99	0.98	1.00	0.83	1.00	0.94	1.00	**
	Recall	0.86	0.92	0.99	0.99	1.00	0.66	1.00	0.94	1.00	**
	Base acc.	0.9910									
OVF C10	Prec.	0.91	1.00	0.99	0.98	0.99	0.76	1.00	0.95	0.88	1.00
	Recall	0.89	0.92	0.99	0.99	1.00	0.73	1.00	0.93	1.00	0.94
	Base acc.	0.9890									

Figure 6.11: Second stage model performance over class expansion. 1-time 4-fold cross-validation. Evaluation of model base accuracy, precision and recall after implementation of eight, C8, nine, C9, and ten, C10, different defect classes.



Class		100	150	200	300	301	350	360	500	550	575
LS 0.15	Prec.	0.95	0.88	0.98	0.99	0.98	0.93	1.00	0.94	1.00	0.97
	Recall	0.91	0.83	0.99	0.99	1.00	0.88	1.00	0.93	1.00	0.90
	Base acc.	0.9877									
LS 0.10	Prec.	0.91	0.88	0.98	0.99	1.00	0.93	1.00	0.93	1.00	0.97
	Recall	0.89	0.83	0.99	0.99	1.00	0.83	1.00	0.94	1.00	0.91
	Base acc.	0.9880									
LS 0.05	Prec.	0.96	0.94	0.98	0.99	1.00	0.92	1.00	0.95	1.00	0.99
	Recall	0.89	0.83	0.99	0.99	1.00	0.83	1.00	0.95	0.75	0.89
	Base acc.	0.9887									
LS 0.025	Prec.	0.95	0.79	0.98	0.99	1.00	0.95	1.00	0.94	1.00	0.99
	Recall	0.86	0.75	0.99	0.99	1.00	0.83	1.00	0.94	1.00	0.91
	Base acc.	0.9888									

Figure 6.12: Second stage model performance over label smoothing alpha. 1-time 4-fold cross-validation per alpha value. Maximum class size set to 1000. Minority classes contained in light green boxes.

Class		100	150	200	300	301	350	360	500	550	575
B1	Prec.	0.9884	0.8888	0.9734	0.9877	1.00	0.9553	0.9973	0.9622	1.00	0.9912
	Recall	0.9226	0.8888	0.9911	0.9939	1.00	0.8371	0.9963	0.9518	1.00	0.9611
	Base acc.					0.9847					
B2	Prec.	0.9738	0.8958	0.9769	0.9884	0.9973	0.9276	0.9970	0.9721	1.00	0.9725
	Recall	0.9503	0.8611	0.9842	0.9966	1.00	0.8269	0.9969	0.9484	0.80	0.9692
	Base acc.					0.9845					
B3	Prec.	0.9760	0.9583	0.9835	0.9920	1.00	0.9377	0.9966	0.9627	0.9285	0.9755
	Recall	0.9871	0.8194	0.9831	0.9946	1.00	0.8686	0.9976	0.9610	1.00	0.9740
	Base acc.					0.9863					
B4	Prec.	0.9722	0.8819	0.9836	0.9913	0.9973	0.8897	0.9950	0.9742	1.00	0.9760
	Recall	0.9342	0.9444	0.9823	0.9955	1.00	0.8675	0.9976	0.9500	0.7777	0.9801
	Base acc.					0.9859					

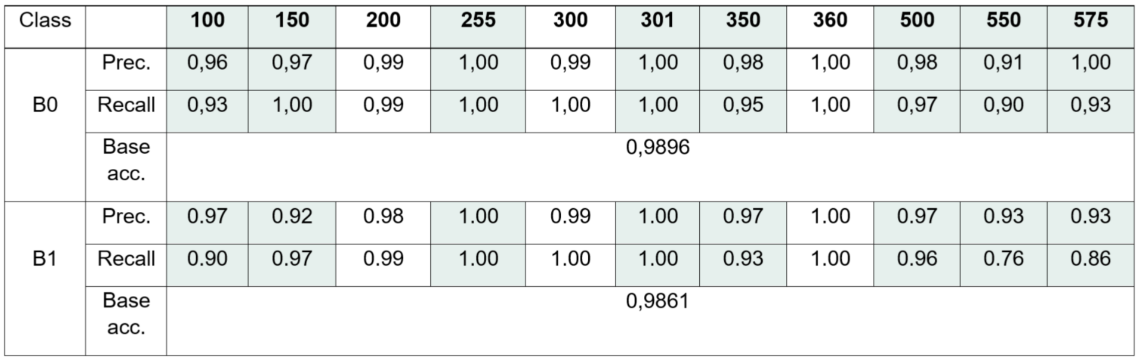
Figure 6.13: EfficicentNet architectures performance. 1-time 4-fold cross-validation per architecture. Maximum class size set to 1000. Minority classes contained in light green boxes. Improvements in performance (green values) and losses in performance (red values) using B1 architecture as reference.

Class		100	150	200	255	300	301	350	360	500	550	575
OVF	Prec.	0.96	0.79	0.99	1.00	0.99	1.00	0.97	1.00	0.94	1.00	0.97
	Recall	0.88	0.88	0.99	1.00	0.99	1.00	0.92	1.00	0.95	1.00	0.89
	Base acc.						0.9897					

Figure 6.14: Third stage model performance after No-Defect inclusion. 1-time 4-fold cross-validation on EfficientNet B0 architecture. Maximum class size set to 5103. Minority classes contained in light green boxes.

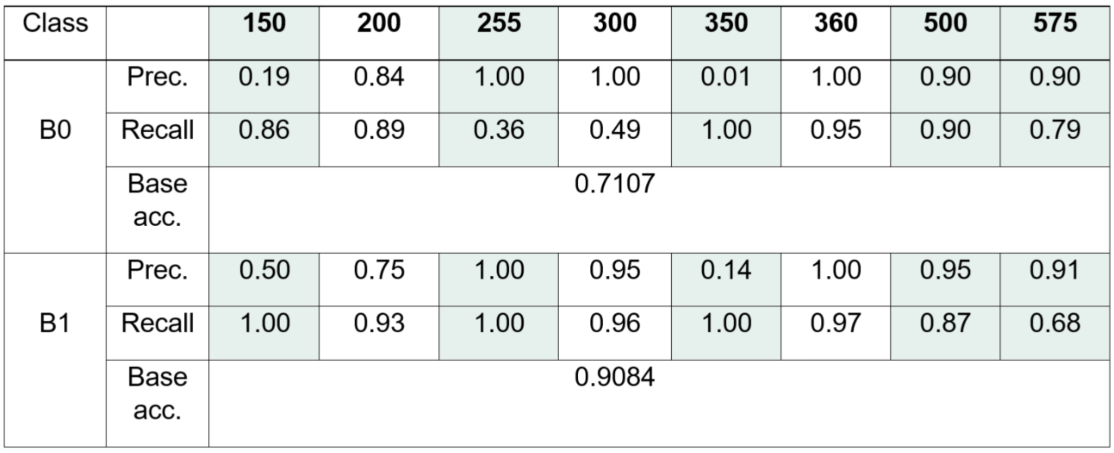
Class		100	150	200	255	300	301	350	360	500	550	575
OVF/2	Prec.	0,95	0,98	0,99	1,00	0,99	1,00	0,99	1,00	0,98	1,00	0,96
	Recall	0,91	1,00	0,99	1,00	1,00	1,00	0,99	1,00	0,97	0,92	0,88
	Base acc.						0,9918					

Figure 6.15: Third stage model performance after oversample mitigation. 1-time 4-fold cross-validation on EfficientNet B0 architecture. Maximum class size set to 5103. Minority classes contained in light green boxes.



Class		100	150	200	255	300	301	350	360	500	550	575
B0	Prec.	0,96	0,97	0,99	1,00	0,99	1,00	0,98	1,00	0,98	0,91	1,00
	Recall	0,93	1,00	0,99	1,00	1,00	1,00	0,95	1,00	0,97	0,90	0,93
	Base acc.	0,9896										
B1	Prec.	0.97	0.92	0.98	1.00	0.99	1.00	0.97	1.00	0.97	0.93	0.93
	Recall	0.90	0.97	0.99	1.00	1.00	1.00	0.93	1.00	0.96	0.76	0.86
	Base acc.	0,9861										

Figure 6.16: Third stage model performance after class 575 resampling. 5-times 4-fold cross-validation on EfficientNet B0 and B1 architecture. Maximum class size set to 5103. Minority classes contained in light green boxes.



Class		150	200	255	300	350	360	500	575
B0	Prec.	0.19	0.84	1.00	1.00	0.01	1.00	0.90	0.90
	Recall	0.86	0.89	0.36	0.49	1.00	0.95	0.90	0.79
	Base acc.	0.7107							
B1	Prec.	0.50	0.75	1.00	0.95	0.14	1.00	0.95	0.91
	Recall	1.00	0.93	1.00	0.96	1.00	0.97	0.87	0.68
	Base acc.	0.9084							

Figure 6.17: Final model testing acccuracy for raw dataset. EfficicentNet B0 & B1 architecture. Minority classes contained in light green boxes. Class 150 and 350 defects proved the most difficult to predict in both families.