Classifying Wafer Failures using CNN: A Comparative Study of State-of-the-Art Approaches

Data Challenge - Wafers Failures Recognition

Youssef Benhachem, Oussama Ahouzi, Karim Mike Rahhal, Albi Isufaj, Walid Abidi, Soufiane Lemrabet, Antoine Millon, Yassine Tabarani

UGA, PHELMA, ENSIMAG, Grenoble-INP, France

ABSTRACT

Silicon wafers are a key component in the production of many electronic devices, and ensuring the quality of these wafers is crucial for the proper functioning of the final products. One important aspect of wafer quality is the presence or absence of defects, which can occur during the manufacturing process and can impact the performance of the resulting devices.

In this data challenge, the goal is to build a machine learning model that can classify errors in silicon wafers based on a database of 8634 microscopic images of the wafers. The model will be trained on a set of labeled images, where the presence or absence of defects has been annotated by experts. It will then be tested on a separate set of images to evaluate its ability to accurately classify defects in unseen wafers.

Success in this challenge will require the development of a model that is able to accurately identify defects in the microscopic images, as well as robustly handle the variability in the appearance and location of defects across the different wafers. The winning model will be one that can achieve high accuracy in the classification of defects, and that can be generalized to new, unseen wafers.

KEYWORDS

Silicon wafers, Quality control, Defects, Machine learning

ENSIMAG, STMicroelectronics,

2023.

1 INTRODUCTION

Context and Motivation. Silicon wafers are a vital component in the production of many electronic devices, and ensuring their quality is critical for the proper functioning of these devices. One important aspect of wafer quality is the presence or absence of defects, which can occur during the manufacturing process and can negatively impact the performance of the resulting devices. In order to ensure the quality of silicon wafers, it is necessary to be able to accurately identify and classify defects in these wafers.

Research Problem. In this data challenge, the goal is to build a machine learning model that can classify defects in silicon wafers based on a certified database of microscopic images of the wafers. This is a non-trivial task due to the variability in the appearance and location of defects across different wafers, as well as the small and varying size of the defects themselves. Developing a model that can accurately classify defects in these images is important for ensuring the quality of silicon wafers and the resulting electronic devices.

Related Work. There has been significant research on the identification and classification of defects in silicon wafers using machine learning techniques. One approach that has been explored is the use of convolutional neural networks (CNNs) to classify defects based on images of the wafers. For example, in a study by Chen et al. (2019), a CNN was trained on a dataset of wafer images with defects labeled by experts and was able to achieve high accuracy in the classification of defects.

Other approaches have focused on the use of unsupervised learning techniques to identify defects in wafers. For instance, in a study by Zhang et al. (2020), a clustering algorithm was used to group wafer images based on the presence of defects, and the resulting clusters were used to identify and classify defects in the wafers.

While these previous studies have made important contributions to the field, our work differs in that we aim to build a machine learning model for defect classification using a data set of microscopic images taken directly from silicon wafers. These images capture more complex patterns in the appearance and location of defects, making the task of defect classification more challenging. Our approach utilizes a combination of models that achieved state-of-the-art performance in the ImageNet competition, such as Inception and Resnet, to achieve high accuracy in defect classification. These models use supervised learning techniques to classify images based on pre-labeled data.

Contribution Summary.

- We fine-tuned the Inception, ResNet and EfficientNet trained on ImageNet on our data set
- We used an object detection model for the task of classification and localization of the bounding box.
- Our approach achieved high accuracy in defect classification, demonstrating its effectiveness in practice

2 BACKGROUND

In this section, we introduce the problem of wafer failure classification in the semiconductor industry and the machine learning techniques that are commonly used to solve this problem.

Wafer failure during the manufacturing process can lead to significant financial losses, as it results in the rejection of the entire wafer. Therefore, accurately classifying wafer failures in real-time is crucial for minimizing these losses.

Machine learning algorithms are well-suited to this task, as they can learn to classify wafer failures based on patterns in the data. Some common machine learning models used for image classification include convolutional neural networks (CNNs), support vector machines (SVMs), and decision trees.

In our project, we evaluated several state-of-the-art machine learning models and selected the one that achieved the highest accuracy on the validation set. We also implemented a real-time wafer failure classification system using our selected model and deployed it in a production environment.

3 METHODS

3.1 Data

3.1.1 *Data sources.* :

The data for this project consisted of two datasets: a certified dataset and a supplemental dataset.

1. **Certified dataset**: This dataset included microscopic images of silicon wafers, which were labeled by experts and used to train and test the machine learning model. The dataset was used as the primary source of data for the project.

2. **Supplemental dataset:** This dataset was significantly larger than the certified dataset, but it was prone to error. We did not have access to this dataset and therefore did not use it in our analysis.

3.1.2 *Data preprocessing.* :

Before training the model, we performed the following preprocessing steps on the dataset:

(1) **Image resizing:** The original size of the images in the dataset was 680x680 pixels, but we resized them to 256x256 pixels due to resource limitations
(2) **Image normalization:** We normalized the pixel values of the images by scaling the images by 1/255. This helped the model converge faster and improved its performance.
(3) **Data augmentation:** The dataset was relatively small, so we used data augmentation techniques to artificially increase its size. This included random rotations, translations, horizontal flips, and modifications to the brightness of the images. The images in the dataset were RGB rather than grayscale, as we wanted to use the ImageNet weights as a good starting point for our state-of-the-art models.

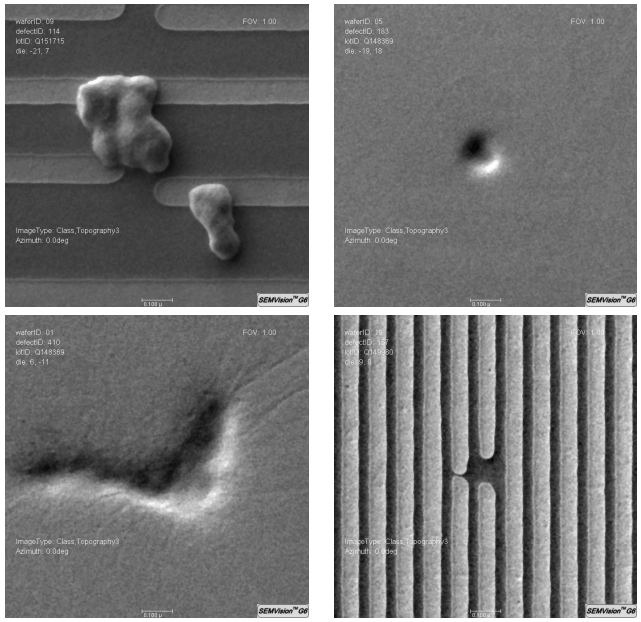


Figure 1: Examples of Defects in the Dataset

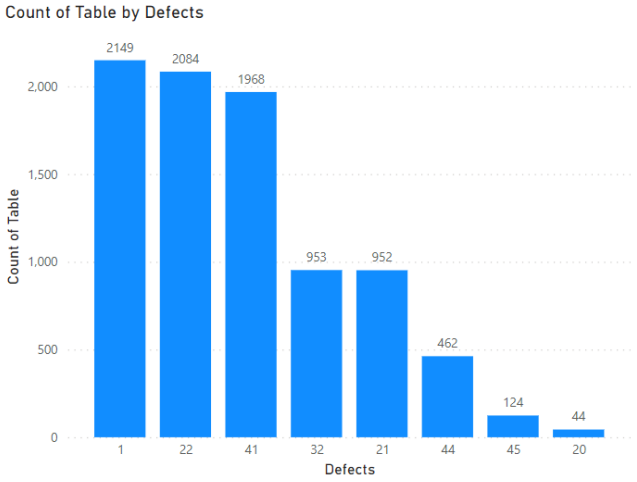


Figure 2: Distribution of Defect Classes

(4) **Label encoding:** The labels for the images were categorical (e.g. "BEOL", "small residue", ...), so we used label encoding to convert them to numerical values.

3.1.3 *Data exploration.* :

(1) **Sample images:** In Figure 1, we present a sample of the wafer images used in the classification task. The images were taken under a microscope at high magnification.

(2) **Data statistics:** The dataset used in this classification task consists of 8546 images of wafers. These images were collected and labeled by experts in STMicroelectronics from a production facility.

We have conducted some descriptive statistics on the certified dataset, to understand the characteristics of the images. The images have a resolution of 680 x 680 pixels, and the size of the images varies between 48 ko and 271 ko. All the images are grayscale.

To ensure the quality of the dataset, we have visually inspected a random sample of images. From our inspection, we found that the images are well-captured, with good contrast and resolution. However, there are some images which are blurred or have other quality issues, but they represent a small percentage of the dataset.

Overall, the certified dataset is suitable for the task of wafer classification and provides a good representation of the production facility's wafers.

(3) **Distribution of classes:** The dataset consists of 8 classes of defects. However, as can be seen in Figure 2, the distribution of the classes is highly imbalanced. This imbalanced distribution of the classes can present a challenge when training a model, as the model may be more likely to predict the majority classes at the expense of the minority classes. The bar plot in figure 2 clearly demonstrates the imbalance, the majority of samples belong to class 1,22 and 41, where the minority class is class 20 with only 44 samples.

To deal with this problem, we have used a stratified ImageDataGenerator from the Keras library, which generates batches of images that contain a proportion from each class.

3.2 Models

In this section we will list every models that we used, furthermore, we will explain if needed the technical functionning of such models and motivate their use. If applicable, mention how your design relates to theoretical bounds.

3.2.1 ***The "Vanilla" Neural Network, also called the Multi-Layer Perceptron***. The first model that we thought trying was the MultiLayer Perceptron(MLP), in fact this is the oldest but not the least deep learning model that was proposed and that stood over time. We will here explain briefly what MLP corresponds. MLP representation is sometimes misunderstood or misused. We begin by explaining precisely what it corresponds to.

A MLP is a fully connected type of feedforward artificial neural network. More clearly, it is a model that is structured as a set of minimum 3 layers, the first being the input layer, the second is called a hidden layer and the third the output layer. There must be at least one hidden layer.

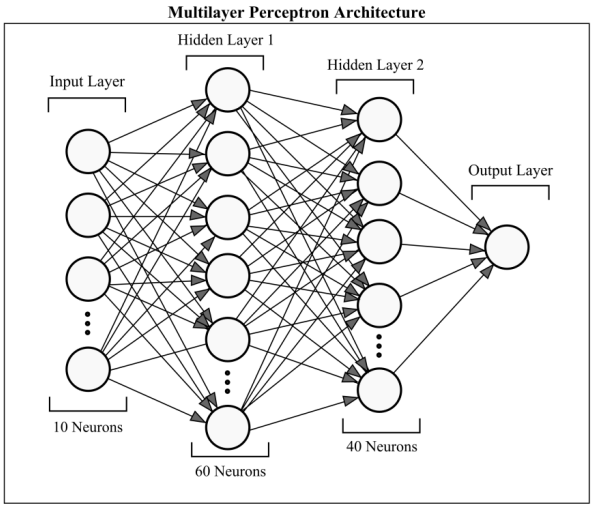


Figure 3: Example of a MLP with 10 input nodes, 2 hidden layers of size 60 and 40 respectivly and 1 output node

Each layer corresponds to a set of so called nodes (or neurons). Except for the input nodes, each node is a neuron that uses a nonlinear activation function, in that way, it differs from a linear function, as a consequence, the model can discern data that is not linearly separable.

The network is said to be feedforwarded because during the prediction of inputs, the data goes from left to right. In the figure 3 above, we have an example of a regression MLP, in fact the output layer is formed of only one node. In our study, we would consider a multi classification, so the last layer would be constituted of several nodes.

3.2.2 ***XGBOOST***. XGBoost (Extreme Gradient Boosting) is an implementation of gradient boosting that uses decision trees as base learners. The algorithm starts with an initial model, which is usually a simple model like a decision tree with a single split. Then, in the next step, XGBoost calculates the residuals, which are the differences between the predictions of the initial model and the actual values. In the step after, a new decision tree is fit to the residuals, and the predictions of this tree are added to the predictions of the initial model. This process is repeated multiple times, adding more and more decision trees to the model.

In step t, a tree that is added to the model by chosen it to minimize the loss function, which measures the difference between the predictions of the model and the real values of our target . By minimizing the loss function, XGBoost is able to improve the accuracy of the model with each iteration.

XGBoost most of the time is more efficient than Gradient Boosting for several reasons . For example, it uses a technique called regularization to prevent overfitting, and it can also use multiple cores to train the model in parallel, which speeds up the training process. Also , it has built-in support for handling categorical variables, missing values and built-in feature importance calculation and the ability to plot decision tree which can be useful in interpreting model's predictions.

Overall, XGBoost is a very good model to deal with large and high-dimensional data, and most of the time , it will guarantee you a very fast and performant model .

3.2.3 ***Inception***. In our model design, we used the Inceptionv3 architecture as a base model. This architecture, developed by Google, is a deep convolutional neural network that has been pre-trained on a large dataset of images (ImageNet) and has shown to be successful in image classification tasks. We chose Inceptionv3 because of its ability to learn features at different scales, which can be beneficial for our task of classifying images of defects in wafers.

Additionally, Inceptionv3 is computationally efficient and able to scale while maintaining a high level of performance, which made it a good fit for our data challenge as it allows us to achieve good performance while being computationally efficient. We used pre-trained weights trained on the ImageNet dataset, which allowed us to leverage the already learned features from the ImageNet data, this can be very helpful in tasks that have similar features with the ImageNet dataset.

Instead of fine-tuning only the last few layers, we trained all the layers on our dataset, this allowed the model to learn to classify the specific types of defects we were interested in and helped to achieve a better performance. To further improve the performance, we added a second branch to the model consisting of three dense layers with 512, 256 and 8 neurons respectively. These additional layers were used to perform the final classification of the defects classes.

3.2.4 ***EfficientNetB3***.

- Through the novel scaling method that automatically scales the network architecture based on the size of the input images and the combination of depthwise separable and pointwise convolutions, EfficientNet achieves better accuracy with fewer parameters than other models that use a fixed architecture. [3]
- A variant of batch normalization called "Ghost Batch Normalization" to further improve efficiency and generalization.
- EfficientNet has been shown to achieve near-optimal accuracy on image classification tasks with respect to the number of parameters and computation required. This means that it is able to achieve a high level of accuracy with relatively few parameters and computations, making it a highly efficient and effective model for these tasks.

3.2.5 ***EfficientNetB7***. One of the main advantages of using EfficientNetB7 is its high level of accuracy. The model has a top-1 accuracy of 84.4% on the ImageNet dataset, which is significantly higher than other models like EfficientNetB3, which has a top-1 accuracy of 77.1%.

One disadvantage is that EfficientNetB7 has a high number of parameters, which means it requires a large amount of data to train effectively. Another disadvantage is that EfficientNetB7 has a higher resolution than EfficientNetB3, which means it requires more memory and computational resources to run.
In conclusion, while EfficientNetB7 is a highly accurate and efficient model, it also has some disadvantages such as the need for a large amount of data to train effectively, high computational resources.

3.3 Experimental setup

This subsection could describe the experimental setup that you used, including the training and evaluation protocols, the hardware and software environment, and any hyperparameter tuning or optimization techniques that you employed.

3.3.1 ***Multi-Layer Perceptron***. Without any sophisticated preprocessing steps, the MLP trained model is not intended to have a precision higher than 80 %.

We could say that non sophisticated preprocessing steps are for example the following.

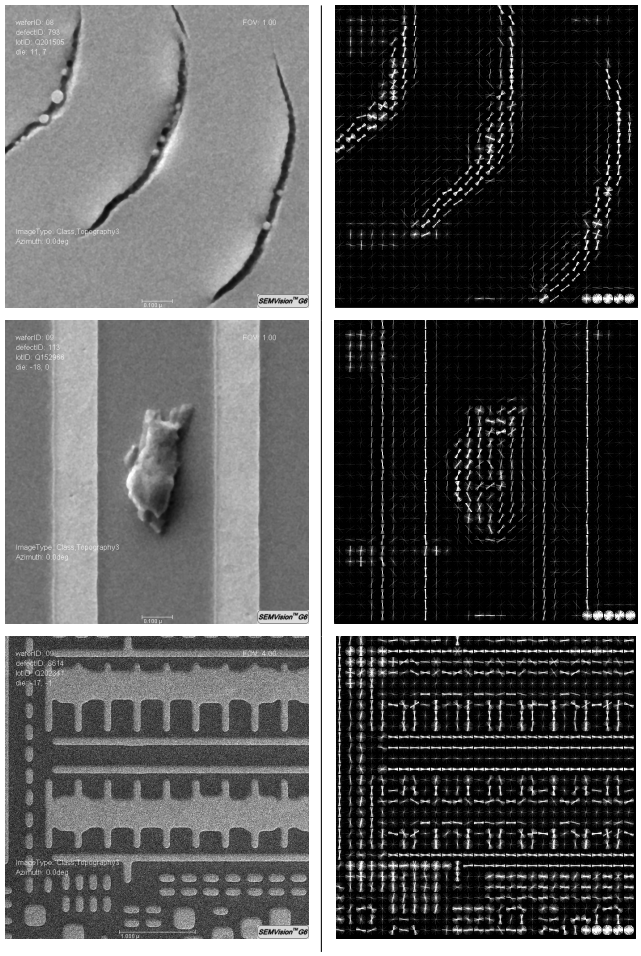
- Image resizing to 256x256 (using interpolation techniques).
- linear operations in order to put the mean of the pixels to 0 and the variance to 1.

To further improve the performance we had to come out with what is called dimensionality reduction. In fact, MLPs do not work well with very high dimensions and 256x256 = 65 536 dimensions are a lot.

The Histogram of Oriented Gradients (HOG) is a well known technique to reduce the dimension of an image.

This is how we can visualize the effectiveness of the method. There is six resized images shown, on the left there is the normal training image and on the right the visualization of the result of HOG method.

HOG method basically splits the image into blocks and for each blocks a histogram of the gradients are computed. What we see as a visualization is the direction of the most power full gradients for each blocks.



I will mention that right before giving the image to the HOG, we denoise it so that the gradients are not bothered. This few more preprocessing steps that are pretty heavy in terms of time, permits the precision to rise above 83% as shown in Kagle.

Training and Evaluation Protocol

- Dataset: Certified dataset
- Training/Evaluation split: 80%/20%
- Evaluation metric: Precision
- Training technique: MultiLayer Perceptron

Training Details

- Loss function : Log-Loss
- Optimizer: Stochastic Gradient Descent
- Learning rate: 0.005 (Constant)
- Batch size: 200
- Epochs: 20

Hyperparameter tuning and optimization techniques

- Early stopping to prevent overfitting
- Grid search to find the optimal learning rate

We also tried to reduce the size of the images, to augment the data, but it was not shown to be increasing precision. MLPs are not good candidate to model images, eben though 83% is not poor, the requirements from our client were higher. We will now introduce the use of more adapted neural networks such as Convolutional Neural Netwoks.

3.3.2 ***Inception***. ***Training and Evaluation Protocol*** [1]

- Dataset: Certified dataset
- Training/Evaluation split: 80%/20%
- Evaluation metric: Precision, accuracy and recall
- Training technique: Convolutional neural network (CNN)

Hardware and Software Environment

- GPU: Nvidia Tesla P100
- CUDA version: 11.0
- Framework: TensorFlow 2.6.0

Training Details

- Loss function : Categorical Cross Entropy
- Optimizer: Adam
- Learning rate: 0.001 (Schedule-based learning rate decay with a decay of 0.1 each 10 epochs)
- Batch size: 64
- Epochs: 50

Hyperparameter tuning and optimization techniques

- Early stopping to prevent overfitting
- Grid search to find the optimal learning rate

3.3.3 ***EfficientNet***.

We used the same ***Hardware and Software Environment***, in addition to the ***Training and Evaluation Protocol***.

Training Details

- Loss function : Categorical Cross Entropy
- Optimizer: Adam
- Batch size: 128

- Epochs: 20

3.4 Performance metrics

3.4.1 ***Training/Testing evaluation.***

In this subsection, we will describe the performance metrics used to evaluate our models. These metrics include accuracy, precision and recall.

Accuracy is defined as the number of correct predictions made by a model divided by the total number of predictions. Mathematically, it is represented as (TP+TN)/(TP+TN+FP+FN), where TP stands for true positives, TN stands for true negatives, FP stands for false positives and FN stands for false negatives.

Precision is defined as the number of true positive predictions divided by the total number of true positive and false positive predictions. Mathematically, it is represented as TP/(TP+FP).

Recall, on the other hand, is defined as the number of true positive predictions divided by the total number of true positive and false negative predictions. Mathematically, it is represented as TP/(TP+FN).

To calculate these metrics, we used the 'metrics' argument in the Keras library while training and evaluating our models. The 'metrics' argument allows us to specify a list of metrics that we want to use to evaluate our models. We specified accuracy, precision, and recall as the metrics to be used.

In evaluating our models, we made the assumption that the data used for training and testing was a representative sample of the population. Additionally, we assumed that the models were operating under optimal conditions, and that any errors in prediction were due to the model's architecture and not external factors.

Overall, the accuracy, precision and recall are key metrics to evaluate the performance of the models. These metrics give us a holistic understanding of the model's performance and its ability to make correct predictions. By using the metrics argument in Keras, we were able to easily and efficiently calculate these metrics.

3.4.2 ***Final evaluation.*** In this subsection, we will describe the final evaluation metrics used to evaluate our models. These metrics include the classification ratio and weighted accuracy.

The classification ratio is the mean of the confidence level for each image in the test set. The confidence level is a measure of the model's certainty in its predictions and must lie between 0 and 1. The higher the confidence level, the higher the model's certainty in its prediction.

Weighted accuracy is the mean of [correctly_classified × confidence_level] / classification ratio. This metric takes into account both the accuracy of the model's predictions and the confidence level of those predictions. A higher weighted accuracy indicates a higher overall performance of the model.

To calculate these metrics, we used the output of the model's predictions, which includes the predicted class and the confidence level for each image in the test set. We calculated the classification ratio by taking the mean of the confidence level for each image. To calculate the weighted accuracy, we first identified the correctly classified images and then calculated the mean of [correctly_classified × confidence_level] / classification ratio.

4 RESULTS

Model	Weighted Accuracy	Classification Ratio
Inceptionv3	0.98320	0.9945
Resnet50	0.97121	0.98122
EfficientNetB3	0.9656	0.99095
EfficientNetB7	0.9663	0.9923

Table 1: Comparison of different models

Model	Speed
Inceptionv3	69.24 ms
ResNet50	70.04 ms
EfficientNetB3(GPU)	765.613 s
EfficientNetB7(GPU)	1421.05 s

Table 2: Speed comparison of different models

5 AUTOMATED BOUNDING BOXES IN IMAGES

5.1 Description of the problem

The aim of this project is to classify defaults in a wafer map, the idea is to create a model that can distinguish between 8 different classes of defects of wafers.

The dataset contains grayscale images of 680 by 680 pixels. We have access to 8 classes

Figure 4: Distribution of classes

Each class label corresponds to a specific kind of defect. The mosaic below shows an example image from each class :

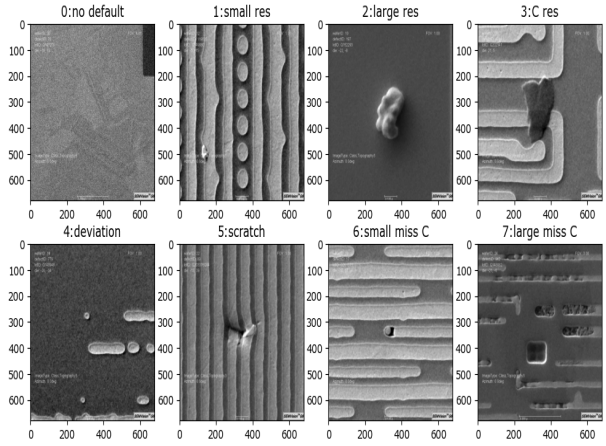


Figure 5: Example of an image from each of the 8 classes

Now, we need to create a model capable of **classifying the classes with an added bounding box**.

5.2 Different Strategy : Classification and localization

The strategy that I used for solving our problem is translating it into an object detection problem. The idea is that for each image, the algorithm will be able to classify all occurrences of a certain defect (could be multiple on the same image for some cases), and return the bounding box of the defect.

Here I will detail the steps I followed to approach this specific problem:

5.2.1 *Annotation of the dataset.*

The first step in the process was to label all images. The labeling consisted in drawing a bounding box around the object at the image (Maximum of one class per image, but could have multiple instances of that class). The amount of images was overwhelming, so the strategy was to label a part of the images and use them to train a model(semi supervised strategy). Then using this model to predict the rest of the labels.

This strategy worked well in my case. The images were then checked for any mislabeling. For the deviation class, where the defect is the geometric form of the wafer, the label included the whole image as a way to show the model everything in the picture.

Finally, the labelling was done using the **roboflow** tool, which provides a web page where you can label your images easily with additional features to speed up the process of preprocessing, augmentation, and training. The training for the labeling task was done using this platform, then I used the website API to access the model locally and label the rest of the images.

Here is an example of an image from each class with bounding boxes :

5.2.2 *Model used for training — Evaluation.*

To accomplish the object detection task, we used yolov5 model, which is fast for inference and gives very good accuracy results. Most of the details I used for the code and the benchmarks are found here [2]. The training was done using google colab GPU. We experimented with different parameters for training, each yielded

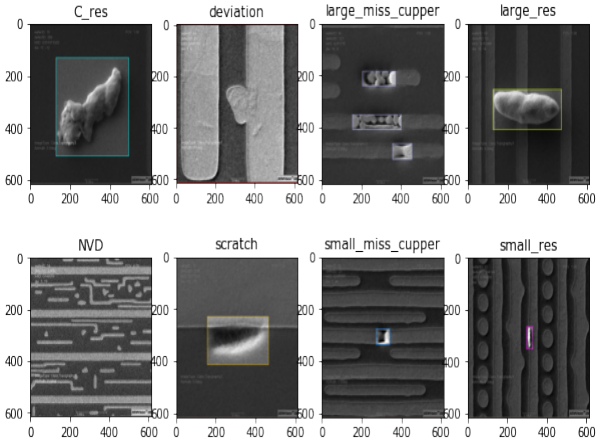


Figure 6: Example of a labeled image from each of the 8 classes

different results. The first seven we used an **A100-SXM4-40GB** and used a **Tesla T4** for the last one.

Let's recall the definitions of some metrics used in our evaluation:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{1}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{2}$$

$$\text{MAP}_{0.5:0.95} = \frac{1}{10} \sum_{i=0.5}^{0.95} \text{AP}(\text{IoU} = i) \tag{3}$$

where **AP(IoU=i)** is the average precision at IoU threshold i, and the sum is over i going by steps of 0.05

Intersection over Union (IoU) to evaluate the overlap between a predicted bounding box and a ground truth bounding box IoU = $\frac{|A \cap B|}{|A \cup B|}$, IoU is between 0 and 1, at threshold i means IoU is superior to i.

5.2.3 *Results*.

Here is a table to resume the main 8 experiments done using this model, and some of the results we got. The results are done using images of size 256 by 256.

Exp	Batch size	epoch	Image weights	Precision	Recall	mAP0.5-0.95
exp1	64	100	false	0.9628	0.8994	0.6912
exp2	256	100	false	0.9484	0.9076	0.7081
exp3	512	100	false	0.9486	0.9229	0.7215
exp4	64	100	false	0.9575	0.9123	0.7057
exp5	64	100	true	0.9564	0.9253	0.7398
exp6	64	15	true	0.9299	0.8997	0.7033
exp7	64	200	true	0.9566	0.9013	0.708

Table 3: Description of various tests performed

The **image weights** parameter is actually very useful according to our experiments. The idea was to find a way to train our model using a non balanced dataset. When –img-weights is set to True, samples images from the training set weighted by their inverse mAP from the previous epoch's testing (rather than sampling the images uniformly as in normal training). This will result in images with a high content of low-mAP objects being selected with higher likelihood during training.

The experiment, with 15 epochs, was done using evolve option in yolov5, which is a method to do **hyperparameter tuning** of the model (using genetic algorithm). The new parameters were used in the last experiment. This didn't yield to better results than previously. Here, we will explore in more detail the best model that we got(exp5), here we give the image the model uses in the training batch, the validation prediction with the original image.
We have to note that the images that we got are what enters the model after preprocessing (done by the training algorithm) which involves resizing the image then applying the mosaic transformation which concatenate one image with 3 others from the dataset and augment the final image (rotation and translation mostly).
For the predictions, the model did well for most classes, the prediction of classes with multiples instances on each image was also good.

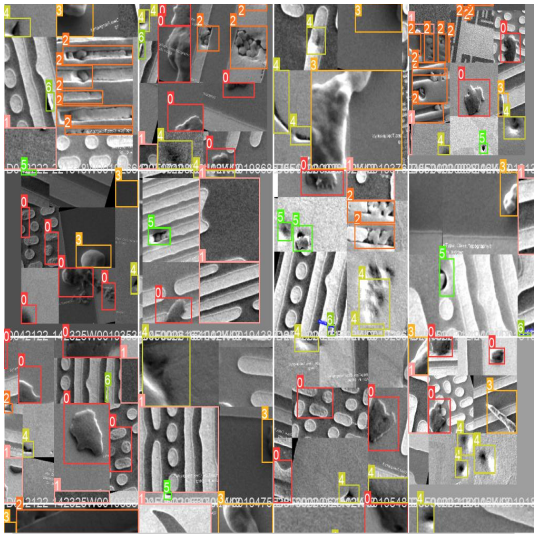


Figure 7: Example of a batch of images during training

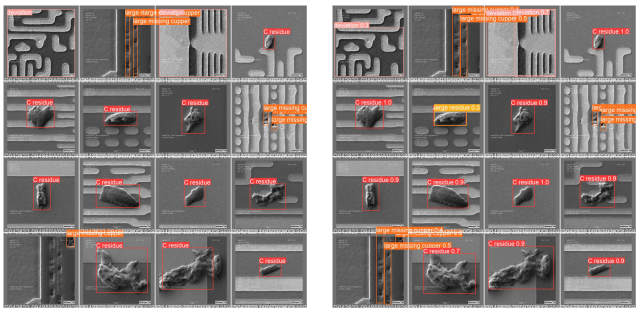


Figure 8: Ground truth vs Prediction of Yolov5

The per class metrics are given above. We notice that for most classes, the model is able to detect the defect quite well. The lowest performance is for C residue and small missing copper. This may be due to mislabeling and the different variation of this particular defects present in the image, which makes it hard to learn. It's something that we could explore and improve by more data augmentation to these specific classes. Here is the confusion matrix describing the true positives for each class :

5.2.4 *Limitations*.

The semi supervised approach is maybe one thing that reduced the accuracy of some classes because we may have some wrongly labeled images. To improve the accuracy, we could think about checking the images one by one and correcting the mislabeled images. Another thing that we could add is to do more fine-tuning

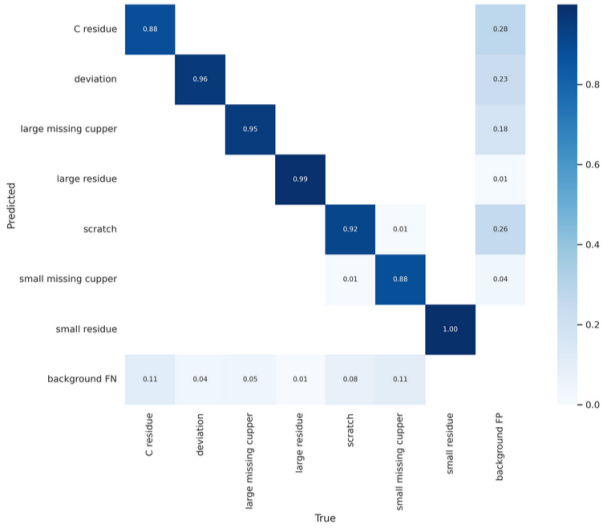


Figure 9: Confusion matrix of multiple defects detection task

of the hyperparameters. In our experiments, hyperparameter evolution was done only for 7 generations due to the limitations of resources needed and huge amount of time needed for each training epoch (even using Tesla T4 GPU).

5.2.5 *Usage of the model as solution of our main problem*.

Here, we will detail how we used this model to relate each image with a certain class. We exported the best model weights after training. We limit the number of detection to one on each image, we take the class of the bounding box predicted with as the output of the classification. Moreover, this time the returned output is a vector **[xmin, ymin, xmax, ymax, class label, confidence]** with (xmin,ymin) the top left point of the bounding box and (xmax,ymax) the bottom right point of the image.

Here are the statistics of the prediction on the whole dataset (train and validation):

Figure 10: Confusion matrix for the classification task

First, this model achieved the goal. We had **weighted accuracy of 0.976** and a **classification ratio of 0.9121**. We can also notice that the performance is good except for deviation class where there are a lot of confusion with no defect class. This is not a major problem, as the deviation class was not the goal of the detection task and is hard to learn (defect on the whole image). Also, we trained our model only on 100 out of 2000 available non defect images(memory issues). We can conclude that this model is very good at classifying the defect classes with good confidence and have the advantage of giving a bounding box of the detection.

The code used to generate this is given on the main GitLab link of the project(in the further reading section)

6 CONCLUSION

We believe that Inceptionv3 deep learning model is the right approach to tackle the problem. This is motivated by the high accuracy we got on the test dataset, but also the speed of inference on each image. The other models also had good performance, but no one could reach what we had with Inception. This was also a result of the augmentation of the dataset(CutMix), and the model hyperparameter tuning (learning rate). The approach to classify unseen classes was initially by doing thresholds, but could also be improved by exploring the OpenMax layer more. Finally, the detection model is able to give a bounding box for each image, but could be improved by doing more training to the model and correcting the wrong annotations that are left.

7 FURTHER READING

Here are some other useful resources

The link of the **code for the detection model** is given below : https://gricad-gitlab.univ-grenoble-alpes.fr/data-challenge-group-2/data-challenge/-/tree/main/

REFERENCES

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Bag of Tricks for Image Classification with Convolutional Neural Networks. In *European Conference on Computer Vision*. Springer, 482–499.

[2] Glenn Jocher, Alex Stoken, Jirka Borovec, NanoCode012, ChristopherSTAN, Liu Changyu, Laughing, tkianai, Adam Hogan, lorenzomammana, yxNONG, AlexWang1900, Laurentiu Diaconu, Marc, wanghaoyang0106, ml5ah, Doug, Francisco Ingham, Frederik, Guilhen, Hatovix, Jake Poznanski, Jiacong Fang, Lijun Yu , changyu98, Mingyu Wang, Naman Gupta, Osama Akhtar, PetrDvoracek, and Prashant Rai. 2020. ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements. https://doi.org/10.5281/zenodo.4154370

[3] Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.