

---

# Rapport de pré-étude

Pôle IA

---

*CrystalClearInspect, Traitement d'images pour détection et classification automatique de défauts dans le domaine du semiconducteur*

---

*Réalisé par :*  
Lucas AMIOT  
Aléo IBRAHIM HOUMED  
Maxence RICHARD  
Maxence ROSSIGNOL  
Thibault SOUBESTE

Nous remercions chaleureusement nos encadrants pour leur aide et leurs conseils précieux, et Adrien GOGUET pour son accompagnement dans notre découverte des plaques de silicium et de leurs défauts.

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Présentation du client . . . . .	1
1.2	Problème et bénéfices . . . . .	1
<b>2</b>	<b>État de l'art</b>	<b>3</b>
2.1	Prétraitement des images . . . . .	3
2.2	Détection . . . . .	3
2.2.1	Segmentation par contours . . . . .	3
2.2.2	Pré-traitement . . . . .	4
2.2.3	Apprentissage profond . . . . .	5
2.2.4	Segment Anything Model (SAM) . . . . .	6
2.3	Classification . . . . .	6
2.3.1	Méthode non-supervisée . . . . .	6
2.3.2	Few-Shot learning . . . . .	8
<b>3</b>	<b>Description du travail réalisé</b>	<b>9</b>
3.1	Présentation structurée des tâches . . . . .	9
3.2	Développement de deux méthodes . . . . .	12
3.3	Enjeu des données . . . . .	12
3.3.1	Logiciel de labellisation . . . . .	12
3.3.2	Les données reçues . . . . .	13
3.4	Approche supervisée . . . . .	15
3.4.1	Données : passage du Faster R-CNN au Mask R-CNN . . . . .	15
3.4.2	1 <sup>er</sup> modèle : Detectron2 . . . . .	15
3.4.2.1	Discussion préalable . . . . .	15
3.4.2.2	Entraînement d'un Detectron2 sur les gros défauts . . . . .	16
3.4.2.3	Entraînement d'un Detectron2 sur les petits défauts. . . . .	17
3.4.3	2 <sup>e</sup> modèle : Entraînement d'un res-net 50 sur PyTorch . . . . .	17
3.4.3.1	Métriques employées pour le res-net 50 . . . . .	17
3.4.3.2	Résultats et perspectives du resnet 50 . . . . .	17
3.5	Approche non supervisée . . . . .	18
3.5.1	Reprise de la détection . . . . .	18
3.5.1.1	Les petits défauts . . . . .	19
3.5.1.2	Les gros défauts . . . . .	19
3.5.2	Classification . . . . .	20
3.5.2.1	Extraction de features . . . . .	21
3.5.2.2	Clusterisation . . . . .	21
3.5.2.3	Résultats . . . . .	22
3.5.2.4	Cas des petits défauts . . . . .	25
3.5.2.5	Deep clustering . . . . .	25
3.5.3	Extraction de meilleures caractéristiques . . . . .	26
<b>4</b>	<b>Conclusion</b>	<b>27</b>

# Chapitre 1

## Introduction

Le présent document est le rapport final de notre projet de deuxième année à CentraleSupélec. Il vise à présenter en détail le problème qui nous a été confié, notre approche pour le résoudre et les étapes parcourues jusqu'à la présentation de notre solution finale. Les autres livrables ont été rassemblés dans un dépôt git privé : ici. Les codes y sont mis à jour avec l'avancée de notre projet et sont à la disposition permanente du client.

### 1.1 Présentation du client

Notre projet est réalisé avec le CEA-Leti (pour Laboratoire d'électronique et de technologie de l'information du CEA). C'est un institut de recherche technologique de CEA Tech, spécialisé dans la microélectronique, qui s'implique dans le développement de solutions novatrices pour l'industrie, la santé, l'énergie et de nombreux autres secteurs. Ses chercheurs et ingénieurs travaillent en collaboration avec des partenaires académiques, institutionnels et industriels pour innover dans la conception de circuits intégrés, dans les technologies du silicium ou entre autres exemples dans l'optique et la photonique.

Notre client dans ce laboratoire est Adrien GOGUET, ingénieur qualité et chef de projet transition numérique. Nous travaillons aussi en lien avec Bastien CHARREL, en alternance au CEA et qui développe un logiciel implémentant nos outils.

### 1.2 Problème et bénéfices

Notre projet s'intitule : "CrystalClearInspect, Traitement d'images pour détection et classification automatique de défauts dans le domaine du semiconducteur". À l'origine de ce sujet, le CEA-Leti est en partenariat avec STMicroelectronics, multinationale franco-italienne qui fabrique des puces électroniques. Ce partenariat les amène à échanger des plaques de silicium utilisées en microélectronique. Or, lors des différents processus industriels, des défauts apparaissent sur ces plaques, tels que des rayures, des poinçons, des traces blanches... (fig 1.1) Afin de pouvoir mesurer l'ampleur et l'importance de ces défauts, le CEA-Leti a mis en place une procédure d'étude en salle blanche. Ainsi, chaque lot de plaques envoyé à STMicroelectronics est pris en photo en lumière rasante lors d'un processus robotisé. Cela permet de fournir des photos en très haute résolution sur lesquelles un technicien peut capitaliser et classifier les défauts inscrits sur la plaque.

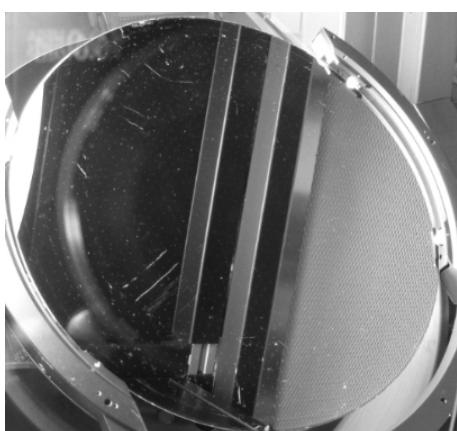


FIGURE 1.1 – Exemple de plaque de silicium présentant des rayures

Seulement, cette évaluation est non seulement pénible et compliquée, mais elle repose aussi sur une évaluation subjective de la part du technicien mobilisé. Le but de ce projet est donc de participer au développement d'une chaîne de traitement de ces images. La finalité est d'atteindre une capacité de classification automatique des défauts, mais toute avancée sur les étapes menant à ce résultat (traitement d'image, détection de défauts, ...) est valorisée par le client.

Plus qu'une aide à la classification, toute la chaîne de traitement des images pourrait être exploitée par le client. En effet, une détection efficace des défauts peut permettre dans certains cas de remonter au processus qui a causé ce défaut. Par exemple, un motif de petits points peut se retrouver sur plusieurs plaques et provenir d'un procédé précis. Ainsi, avoir un recul et une méthode statistique pour répertorier ces défauts pourrait permettre d'augmenter leurs connaissances et peut-être permettre une meilleure gestion de ces plaques, ou même une amélioration des processus qui les abîment trop.

Enfin, la diversité des défauts observés lors des contrôles sur les plaques ainsi que la possibilité de voir apparaître de nouveaux défauts encourage l'écriture d'un code modulable. Ainsi, toute piste de recherche permettant le traitement de nouveaux types de défauts pourra être utile pour notre client et pourra inspirer le développement d'outils futurs.

# Chapitre 2

## État de l'art

Un des grands enjeux dans l'industrie high-tech est la capacité à déterminer la qualité d'un produit afin de pouvoir le corriger si celle-ci n'est pas suffisante ou même pouvoir détecter les rebuts. Que ce soit à l'aide de vidéos, de photos ou même de prise son des produits, de nombreuses méthodes permettent d'analyser la qualité de celui-ci. Pour la recherche de défauts sur des plaques (de silicium dans notre cas), la prise de photos et le traitement d'images sont souvent utilisés. Plusieurs méthodes algorithmiques permettant de prétraiter les images existent. Sur la question de la détection de défauts, les dernières méthodes mises au -point et donnant les meilleurs résultats sont basées sur de l'intelligence artificielle avec de l'apprentissage supervisé ou non en fonctions des cas d'application et des possibilités.

### 2.1 Prétraitement des images

Le prétraitement des données implémenté dans le projet comporte plusieurs étapes de mise en forme de l'image, telles que la mise à plat de la plaque (la photo étant prise avec un angle), le découpage de l'image pour ne considérer que la plaque en elle-même, la rotation de l'image pour avoir un angle d'entrée constant avant la concaténation des images de deux prises de vue différentes avec un angle différent.

Tout d'abord, pour la mise à plat de l'image, dans le but de la redresser afin d'obtenir un point de vue normal au plan de la plaque, des méthodes géométriques sont généralement utilisées. La méthode classique est la transformation de perspective, qui est une transformation affine. La seule difficulté de la transformation de perspective est de trouver les paramètres de transformation nécessaires. Dans notre cas, l'image à transformer est un plan avec plusieurs angles droits : il est donc aisément de trouver quatre points formant un rectangle sur l'image sans perspective et donc un parallélogramme dans l'image d'origine.

Une autre étape importante du prétraitement de notre projet est la rotation de l'image, qui est effectuée a priori après la correction de perspective. Comme il sera détaillé dans la suite du rapport, pour pouvoir implémenter la rotation, nous avons d'abord dû détecter une encoche sur les plaques. L'utilisation des réseaux de neurones convolutifs pré-entraînés pour ce type de tâches très répétitives est fréquente dans la littérature.

Nous ne développerons pas les étapes de découpage et de concaténation de l'image qui sont très simples.

### 2.2 Détection

La première étape de détection revient à faire de la segmentation d'image et plus précisément de l'*instance segmentation*, c'est à dire de faire ressortir des instances apparaissant sur un fond. Dans notre cas, on considère que les instances sont les défauts et que le fond est la plaque sur laquelle ils se situent ainsi que les reflets. Plusieurs méthodes sont mentionnées dans la littérature pour faire cela : la segmentation par région, par classification de seuillage, ainsi que les deux méthodes que nous avons exploré ici que sont l'apprentissage profond et la segmentation par contours.

#### 2.2.1 Segmentation par contours

Il existe une multitude de filtres permettant d'extraire les contours d'une image mais leur fonctionnement est en général toujours le même, à quelques variations près. La première étape est le calcul du gradient de la valeur des pixels de l'image, qui est une dérivée en dimension 2 que l'on calcule de manière discrète. Le gradient est lié à la variation locale de la valeur des pixels, il est élevé lorsque les variations sont fortes. Une telle zone dans l'image va donc correspondre à un « bord », un des contours que l'on cherche à détecter. Le calcul se fait par convolution de l'image avec une matrice rectangle (filtre de Prewitt) ou bien triangulaire (filtre de Sobel) qui obtient de meilleurs résultats. Des méthodes d'ordre deux plus gourmandes existent aussi mais sont en général

moins intéressantes. Notons que la taille de ces matrices peut être choisie, c'est le *kernel* (un couple d'entiers impairs). Un *kernel* petit, (3, 3) par exemple, permettra d'observer chaque variation dans l'image, tandis qu'un *kernel* plus grand fera apparaître des variations de plus grandes échelles comme observés dans la figure 2.1.



FIGURE 2.1 – Comparaison des *kernels* ayant pour valeurs (3, 3) et (29, 29)

Une fois le gradient calculé, on utilise un filtre à hystérésis de manière à le seuiller tout en conservant un aspect connexe. Essentiellement, ce filtre à hystérésis dispose de deux seuils, un seuil haut et un seuil bas. Dans l'image obtenue du gradient, un pixel va être directement considéré comme faisant partie d'un bord si sa valeur est plus élevée que le seuil haut. Dans le cas où cette valeur se situe entre les deux seuils, le pixel sera conservé s'il est relié à d'autres pixels ayant déjà été considérés comme « bord ». En cela, on améliore la continuité des contours détectés et on efface les effets de bruits apparus lors du calcul de gradient.

Dans le cas du filtre de Canny, qui est le plus réputé, afin de réduire les fausses détections liées au bruit, on ajoute un flou gaussien avant le calcul du gradient. De plus, dans la version complète du filtre de Canny, dans une zone où un bord a été détecté, seuls les maxima locaux sont conservés. Cela permet d'extraire précisément un contour qui sera idéalement de largeur 1 pixel [13]. Dans notre cas nous n'appliquerons pas cette dernière étape afin de conserver plus d'information sur le défaut. Un exemple imagé du fonctionnement de l'algorithme de Canny est présenté sur la figure 2.2.



FIGURE 2.2 – Fonctionnement de l'algorithme de Canny [8]

Pour l'algorithme de Canny, les seuils utilisés sont des hyperparamètres que l'on doit déterminer pour chaque image. Le choix de ces seuils est primordial puisque de mauvais seuils peuvent donner lieu à une détection tout à fait incohérente. Au contraire, les paramètres optimaux peuvent donner d'extrêmement bons résultats. C'est donc un enjeu important pour la fonction de détection.

Une première option est d'utiliser la méthode d'Otsu[14]. Elle s'applique lors de la recherche d'une valeur de seuil qui va séparer des pixels d'une image en deux classes différentes (binarisation). Le seuil est alors celui qui minimise la variance intra-classe tout en maximisant la variance inter-classe. Nous garderons alors ce seuil comme seuil maximal du filtre à hystérésis, le seuil minimal étant choisi comme une fraction donnée de ce seuil maximal. D'autres méthodes qui utilisent la moyenne, la médiane et la variance existent mais aucune ne fait consensus dans le cas général. Le choix de la méthode dépend donc du problème et des résultats observés.

### 2.2.2 Pré-traitement

Usuellement, la détection de défaut se fait sur des pièces complexes où une simple détection de contour n'est pas suffisante. C'est pourquoi, il est nécessaire de prétraiter les images avant de procéder à la détection de contours. Lorsque la disposition du contrôle qualité est constant, une possibilité est d'utiliser un masque qui permet de supprimer un motif afin de se concentrer uniquement sur les défauts comme représenté sur la figure 2.3. Après soustraction, il ne reste plus qu'un léger bruit, ainsi que les pixels liés à un défaut.

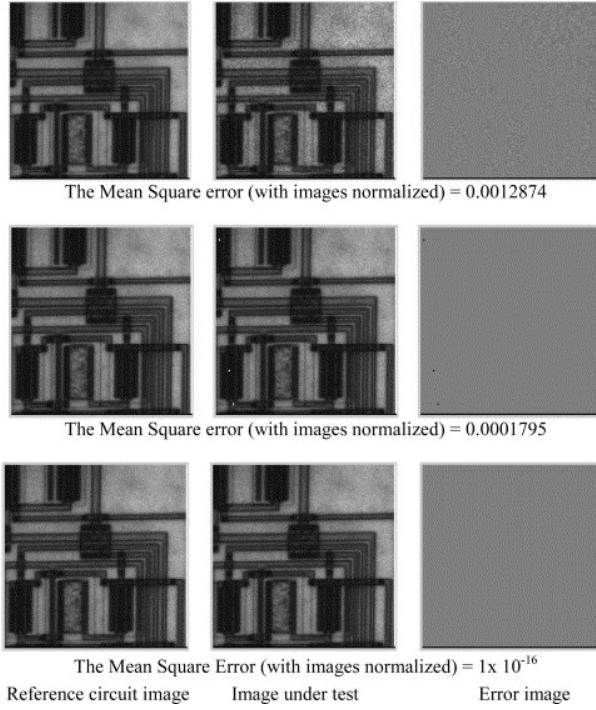


FIGURE 2.3 – Utilisation d'une image "flawless" que l'on soustrait à notre image pour la détection [10]

Un autre prétraitement que l'on peut utiliser est le débruitage de l'image. Le plus connu est le flou gaussien, mais qui a le défaut de réduire les contrastes dans l'image. Une autre méthode est l'utilisation de la transformée en ondelette, et enfin celle pour laquelle nous avons opté : "*Non-Local Means Denoising*" qui a le défaut d'avoir une complexité quadratique [1] mais qui produit d'excellents résultats. Cette dernière garde l'idée de calculer une moyenne pour obtenir la vraie couleur d'un pixel bruité, elle diffère des méthodes classiques en ce qu'elle ne se base pas sur les pixels voisins mais sur les pixels semblables qui peuvent se situer n'importe où dans l'image.

### 2.2.3 Apprentissage profond

De nombreuses approches intéressantes basées sur le deep learning ont été développées entre 2008 et aujourd'hui pour la détection et classification de défauts dans l'industrie high-tech. Elles s'appuient majoritairement sur l'apprentissage profond, parmi lesquelles on trouve entre autres des algorithmes de pixel segmentation et R-CNN (Region-based Convolutional Neural Network).

Pour la première méthode, le principe repose sur l'entraînement d'un réseau de neurones à partir d'images qu'on lui fournit (ou si celles-ci sont trop grandes, seulement des sections d'images). Le réseau retourne en sortie un tenseur de la même dimension que l'image en entrée. Pour un classifieur usuel, on aurait en sortie  $n$  composantes qui correspondent aux  $n$  classes que l'on obtient. Dans notre cas, chaque pixel est classifié, ce qui permet d'avoir en sortie une image à  $n$  canaux, avec  $n$  le nombre de classes voulu. Dans un système de détection simple de défaut, nous n'aurions que deux classes : la classe défaut (ou foreground) et la classe non-défaut (ou background). Cependant cette méthode s'étend aussi à la classification, la première classe de défaut peut être divisée en plusieurs classes, de manière à pouvoir distinguer différents types de défauts [7].

La seconde méthode repose quant à elle sur des réseaux de neurones convolutifs. Le premier algorithme de ce type a été créé en 2013 par Ross Girshick et al. [5], puis a été amélioré à deux reprises, c'est-à-dire rendu plus efficace et plus rapide, avec l'arrivée de Fast R-CNN [4] et Faster R-CNN [9]. Le modèle Faster R-CNN se divise en deux parties ; la première qui permet d'effectuer des propositions de régions, soumises à la deuxième partie qui elle détecte, classe et précise les coordonnées de chaque région d'intérêt. Le résultat de sortie est composé de "bounding boxes", c'est-à-dire des rectangles entourant chaque objet avec la classe respective, associé à un score entre 0 et 1 traduisant la probabilité que ce cadre contienne bel et bien un objet. Par la suite, de nouveaux modèles sont apparus comme Mask R-CNN [6], offrant des résultats plus rapides, plus efficaces et la création de masques entourant chaque objet, plus précis que les bounding boxes.

La méthode d'apprentissage utilisée le plus souvent est l'apprentissage supervisé. Les données sont labellisées en associant à chaque image un masque qui attribue à chaque pixel une couleur représentant la classe à laquelle il appartient, ou une bounding box entourant chaque objet avec la classe correspondante. Cependant, cette méthode demande beaucoup de travail en amont pour obtenir une grande quantité d'images labellisées et une bonne qualité et/ou précision de labellisation. Il est possible de réduire ce travail à l'aide de l'augmentation

de donnée, qui permet de générer de nouvelles données d'entraînement à partir de données déjà labellisées. Cependant, cette charge de travail liée aux données de l'apprentissage supervisé a mené la méthode non supervisée à gagner en popularité. Les réseaux de neurones à apprentissage non supervisé les plus utilisés sont les AE (AutoEncoder), ou les GAN (Generative Adversarial Network)[7].

Un AE est constitué de deux parties : un encodeur, puis un décodeur. L'encodeur utilise des données en entrée non labellisées et tente de les transformer en une représentation de dimension inférieure. Il produit donc une « représentation latente » qui est une version compressée des données qui en contient les caractéristiques importantes de manière compacte. Le décodeur a alors le rôle de reconstituer la donnée d'origine à partir de la représentation latente. Le réseau s'entraîne alors à réduire l'écart entre la donnée reconstruite et la donnée d'origine en minimisant une fonction de coût (qui quantifie donc l'écart, on peut utiliser par exemple l'erreur quadratique moyenne (MSE)). Les données ne sont pas labellisées, l'optimisation s'effectue donc en adaptant les poids du réseau pour réduire cette fonction de coût. Les caractéristiques extraites dans la représentation latente peuvent alors être utilisées pour différentes tâches comme une classification.

Un GAN a un fonctionnement un peu différent. Il a été proposé par Ian Goodfellow en 2014. Son système possède deux réseaux : un générateur et un discriminant. Le générateur doit créer une image en s'inspirant des données du set. Son objectif est de tromper le discriminant qui doit différencier une image créée d'une image du dataset. L'entraînement améliore en parallèle les capacités du générateur et du discriminant jusqu'à trouver un équilibre pour lequel le générateur crée des images que le discriminant n'est plus capable de distinguer d'une image du dataset. Si on peut ensuite exploiter la capacité de ce réseau à extraire des caractéristiques des images, il est plutôt utilisé pour sa capacité à générer des données et ne s'appliquera peut-être pas à ce projet. [12]

#### 2.2.4 Segment Anything Model (SAM)

Des systèmes ne nécessitant pas de nouvel apprentissage pour fonctionner sur des données différentes de celles d'entraînement existent. SAM[2], créé en 2023 par Meta AI se dit avoir une "zero-shot generalization". Ayant appris à reconnaître les contours, forme, perspective et autres vues de millions d'objets, il est capable de très bien généraliser à des objets qu'il n'a jamais aperçu. Basé sur de la segmentation, il prend en entrée une image et renvoie les masques des objets qu'il a réussi à extraire de l'image.

### 2.3 Classification

La classification suit dans la chaîne de traitement la détection qui est responsable de trouver les zones d'intérêts et dans certains cas de les regrouper en ensemble cohérents. Ainsi la classification des défauts peut se voir ou bien comme de la labellisation d'image, ou bien (puisque nous sommes dans un cadre spécifique), comme de la classification de vecteurs. Nous faisons la différence ici entre vecteurs et images dans le sens où une image (bien qu'étant un vecteur) ne peut être considérée comme entrée pour la plupart des méthodes de machine learning.

Dans certaines industries, il peut être utile de savoir quel type de défaut apparaît, puisque certains peuvent être traités tandis que d'autres non. Dans notre cas, l'intérêt est de pouvoir mieux déterminer le risque associé à tel ou tel manipulation puisque les défauts peuvent ou bien être gênant, ou bien rendre la plaque inutilisable. Certaines approches permettent de cumuler détection et classification en une seule étape insécable, c'est notamment ce que fait la méthode supervisée étudiée ici.

#### 2.3.1 Méthode non-supervisée

Labelliser des données (classifer) sans avoir de données d'entraînement s'apparente à de la clusterisation, c'est à dire le fait de regrouper les données en plusieurs groupes les plus cohérents possible en leur sein. Ainsi dans un cas idéal, les clusters trouvés devraient s'apparenter aux labels. Dans le cas de vecteurs (format représentant les données), la similarité entre deux objets peut être donnée par leur distance euclidienne. De multiples algorithmes existent : K-means, DBSCAN, Gaussian Mixture Model, BIRCH, etc. Chacun a ses avantages et ses défauts. K-means est extrêmement simple et est donc une très bonne première option ; en revanche celui-ci est influencé par la répartition des données et suppose que les clusters sont de forme sphérique. DBSCAN fonctionne par densité, il n'est donc pas soumis à la répartition des données mais en nécessite un nombre conséquent pour ne pas avoir des clusters trop "creux" ; de plus, il suppose que les clusters sont de densité semblable, ce qui n'est pas toujours vérifié. On peut le voir sur la troisième ligne de la figure 2.4. Une alternative est le HDBSCAN modèle qui recherche des clusters robustes aux variations du paramètre epsilon choisi dans le modèle DBSCAN. Gaussian Mixture Model résout le problème des clusters sphériques du K-means mais suppose une répartition gaussienne et gère mal les caractéristiques catégoriques... Une image assez représentative des algorithmes les plus communs se trouve ici[2.4].

Ces modèles sont sujet à la "malédiction de la dimensionnalité". En effet plus la dimension est grande et moins la distance euclidienne est efficace pour exprimer une bonne similarité. C'est pourquoi on peut utiliser

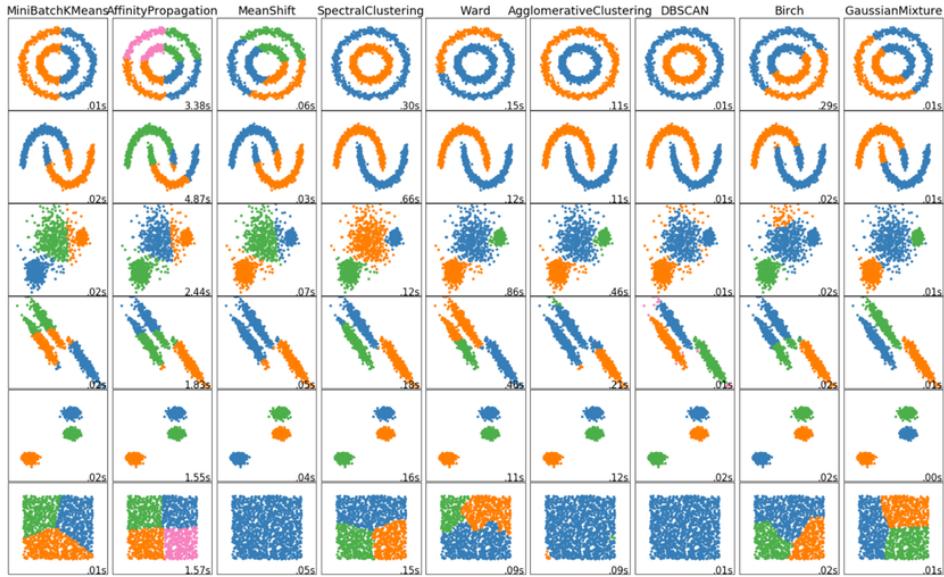


FIGURE 2.4 – Différentes méthodes de clusterisation

des outils de réduction de dimension en amont de ces algorithmes. Une première approche est la sélection de caractéristiques, c'est à dire de ne garder que quelques caractéristiques. Cette méthode a l'avantage d'apporter une très bonne « explicabilité » des résultats. Une seconde approche est l'extraction de nouvelles caractéristiques, c'est à dire de créer de nouvelles caractéristiques à partir des anciennes, de manière à pouvoir reformer les données avec le minimum de perte. Cette méthode a l'avantage de garder plus d'informations (pour un nombre de dimensions donné) que la première. Il existe une multitude de méthodes, certaines linéaires comme l'ACP, mais aussi d'autre non-linéaires comme la Kernel-ACP ou bien encore l'utilisation d'auto-encodeurs. Enfin, tous ces modèles ne sont pas tous adaptés à de la prédiction, contrairement à l'algorithme des KMeans par exemple. Ainsi ces modèles doivent être suivis d'algorithmes supervisés (basés sur les clusters détectés) comme un KNN, un réseau de neurones, un Random Forest, etc...

Dans le cas des images, une première méthode est de se ramener au cas précédent en effectuant de l'extraction de caractéristiques. Pour extraire ces caractéristiques on utilise des CNN qui ont de bonnes capacités d'extraction d'information générale sur des images. Ces CNN doivent être entraînés au préalable. Pour ce faire on peut utiliser un modèle de labellisation entraîné dans un autre contexte (Cifar, Imagenet, MNIST, etc) auquel on enlève la dernière couche, ou bien on peut entraîner un modèle spécifiquement sur ces données. On utilise pour cela une "tâche prétexte" (exemples : prévoir une rotation, puzzle, patch context, inpainting patches, détection d'artefact,...) qui ne sert qu'à apprendre au modèle de bonnes caractéristiques.

La première approche à l'avantage d'être rapide à implémenter mais est peu spécifique au problème et dispose donc de nombreuses caractéristiques qui ne sont pas pertinentes. La deuxième approche a l'inconvénient de devoir entraîner un modèle et de bien sélectionner la tâche d'entraînement. Un second inconvénient de la deuxième méthode est qu'elle nécessite d'avoir des connaissances sur son jeu de données et n'est donc pas facilement généralisable. En effet, certaines pretext-task ont plus de chance de succès dépendamment du jeu de données. Nous avons donc choisi d'étudier la détection d'artefact puisqu'elle répondait au mieux à notre dataset.

Pour entraîner un modèle qui détecte des artefacts, il faut d'abord les créer. L'idée est donc de créer deux modèles et les entraîner comme un Generative Adversarial Network. L'un devra prendre des images et générer des artefacts et l'autre devra réussir à discerner quelles images possèdent des artefacts et quelles images n'en possèdent pas. Pour générer les images, on utilise un Auto-Encodeur. Une fois entraîné, on drop un nombre aléatoire de features de l'image encodée puis on essaye de récréer les trous en intercalant avant chaque couche de décodage une couche de réparation permettant de récréer les trous. Les couches de réparations permettent d'obtenir des images avec des artefacts très similaires à celles sans d'un point de vue bas-niveau ce qui demande au discriminant d'apprendre les features des données sur une échelle plus globale, afin de mieux discerner les images avec artefacts des images sans. On peut ensuite finalement utiliser le discriminant pour récupérer les features représentant les images.

Il existe aussi d'autres méthodes qui ne dissocient pas la phase de sélection de caractéristiques et de clusterisation. Ainsi, elles apprennent en même temps à représenter les données et à les clusteriser. Une méthode parmi d'autres est le modèle SCAN [3] (un modèle de *DeepClustering*, autres exemples : DEC, DAC, Deeper-Cluster, SeCu, ...) qui présume (en utilisant un modèle pré-entraîné pour la première extraction de features) que les plus proches voisins d'un individu font partie de sa classe. Il apprend alors à classifier chaque individu et ses voisins ensemble, puis il utilise les nouvelles features apprises pour mettre à jour les voisins et ainsi de

suite jusqu'à obtenir des clusters tel que le modèle de classification soit sûr de l'appartenance des individus à ces clusters.

### 2.3.2 Few-Shot learning

Pour la partie supervisée, l'enjeu principal est la labellisation. D'une part, c'est une étape très chronophage et elle doit être effectuée directement par le client pour que le modèle corresponde au mieux aux attentes de ce dernier. D'autre part, de nouvelles données deviennent progressivement accessibles. Cela est à la fois une opportunité d'augmenter les capacités du modèle mais également une contrainte puisque si de nouveaux types de défauts apparaissent, le modèle doit être capable de les détecter et de les classifier (sans nécessiter de réentraîner tout le modèle ni de labelliser un grand nombre de fois ces nouveaux défauts), même si ces défauts n'apparaissent que quelques fois. Une méthode a été développée pour tenter de résoudre ces problèmes de généralisation. Il s'agit de l'approche Few-Shot Learning. Cette approche repose sur le fait qu'un modèle pré-entraîné est capable d'extraire les caractéristiques d'objets, en particulier ceux sur lesquels il a été entraîné. L'idée générale est la suivante : lorsqu'un nouvel objet apparaît, on lui fournit quelques exemples de labels correspondants. On appelle cela l'ensemble support, avec l'espoir que les caractéristiques extraites de cet ensemble sont suffisamment différentes des autres classes sur lesquelles le modèle était initialement entraîné. La prochaine fois que le modèle rencontrera cet objet, il essaiera de classifier l'objet en trouvant la classe qui présente les caractéristiques les plus similaires (il est possible de créer une fonction de similarité). On utilise donc ici la capacité du modèle à reconnaître et extraire des caractéristiques : il s'agit de méta-apprentissage. L'approche Few-Shot Learning peut être appliquée de différentes manières et pourrait constituer une solution à notre problème de labellisation et de généralisation, transformant ainsi la méthode supervisée en méthode semi-supervisée.

# Chapitre 3

## Description du travail réalisé

### 3.1 Présentation structurée des tâches

Notre projet en sa totalité a été divisé en quatre grandes parties. La première partie était consacrée à la compréhension du sujet et à la prise en main des outils de base. Après une recherche poussée sur les méthodes existantes, nous avons déterminé les différentes méthodes que nous pourrions mettre en place pour mener à bien les étapes de détection et classification des défauts (méthodes non-supervisée et supervisée).

La deuxième partie regroupe tous les attendus du client du point de vue pré-processing, c'est-à-dire la mise à plat des images (originellement prises avec un certain angle), la suppression des contours, la détection d'une encoche (appelée "notch") pour orienter toutes les images dans le même sens, et la concaténation de deux images.

La troisième partie désigne la détection des défauts sur les plaques, et constitue donc l'attente principale du client. Enfin, la quatrième partie est consacrée à la classification de ces défauts. Pour ces deux étapes, nous avons privilégié deux méthodes après nos recherches de l'état de l'art : une méthode non-supervisée, et une méthode supervisée par apprentissage profond (Deep Learning).

Le projet que nous avons mené ce semestre constitue l'aboutissement d'une année de coopération avec le CEA-LETI. Pour cette seconde partie, notre objectif a été de prolonger le travail réalisé en première partie d'année, en produisant une détection des défauts plus efficace et en proposant des outils de classification opérationnels.

La figure 3.1 présente de manière visuelle les étapes qui ont guidé l'ensemble de notre travail. Enfin, la représentation schématique suivante (3.2 et 3.3) présente la répartition des tâches en grandes parties. Les tâches principales seront ensuite détaillées en lien avec les tâches parallèles qui ont été nécessaires à leur avancée.

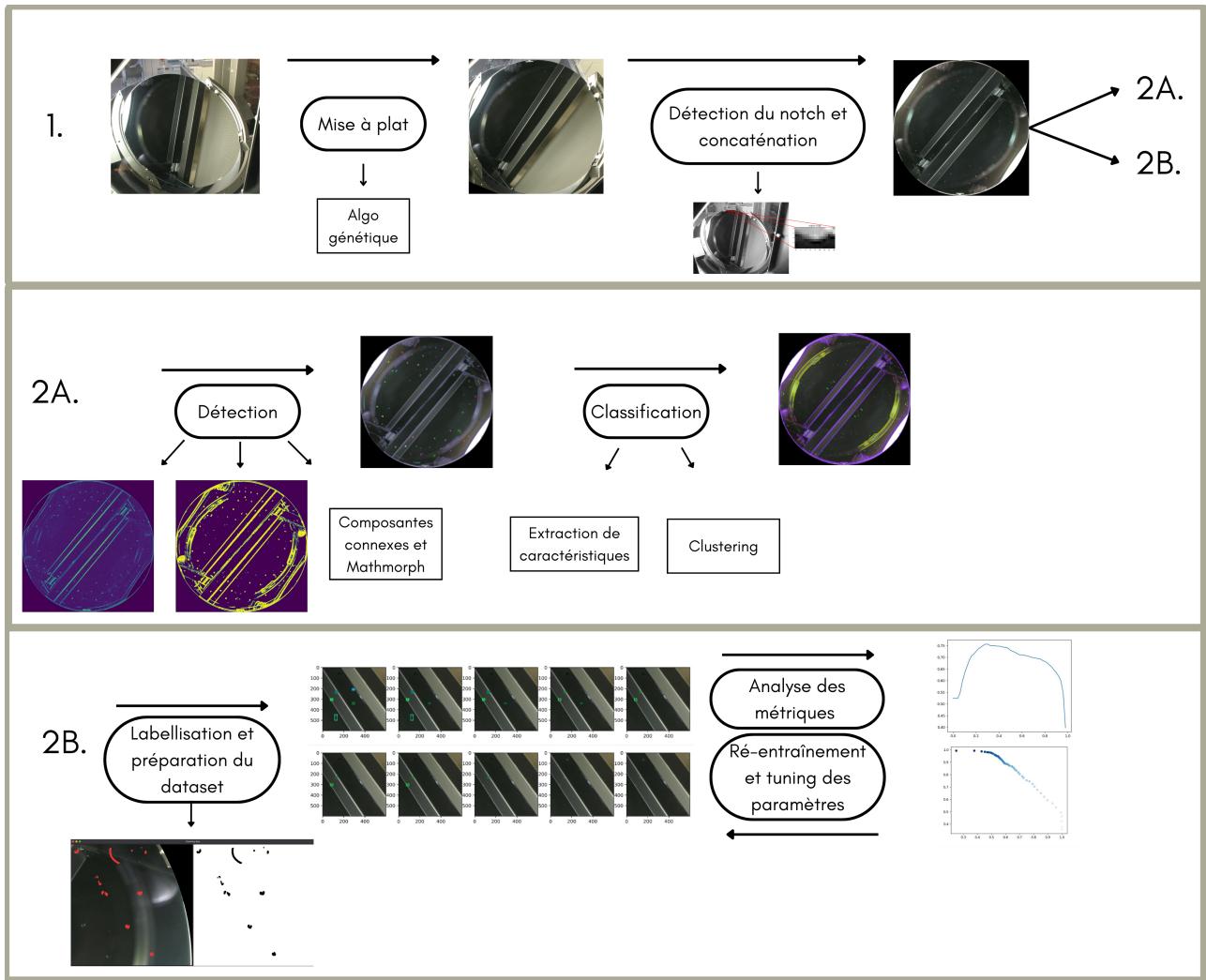


FIGURE 3.1 – Représentation schématique du processus de traitement

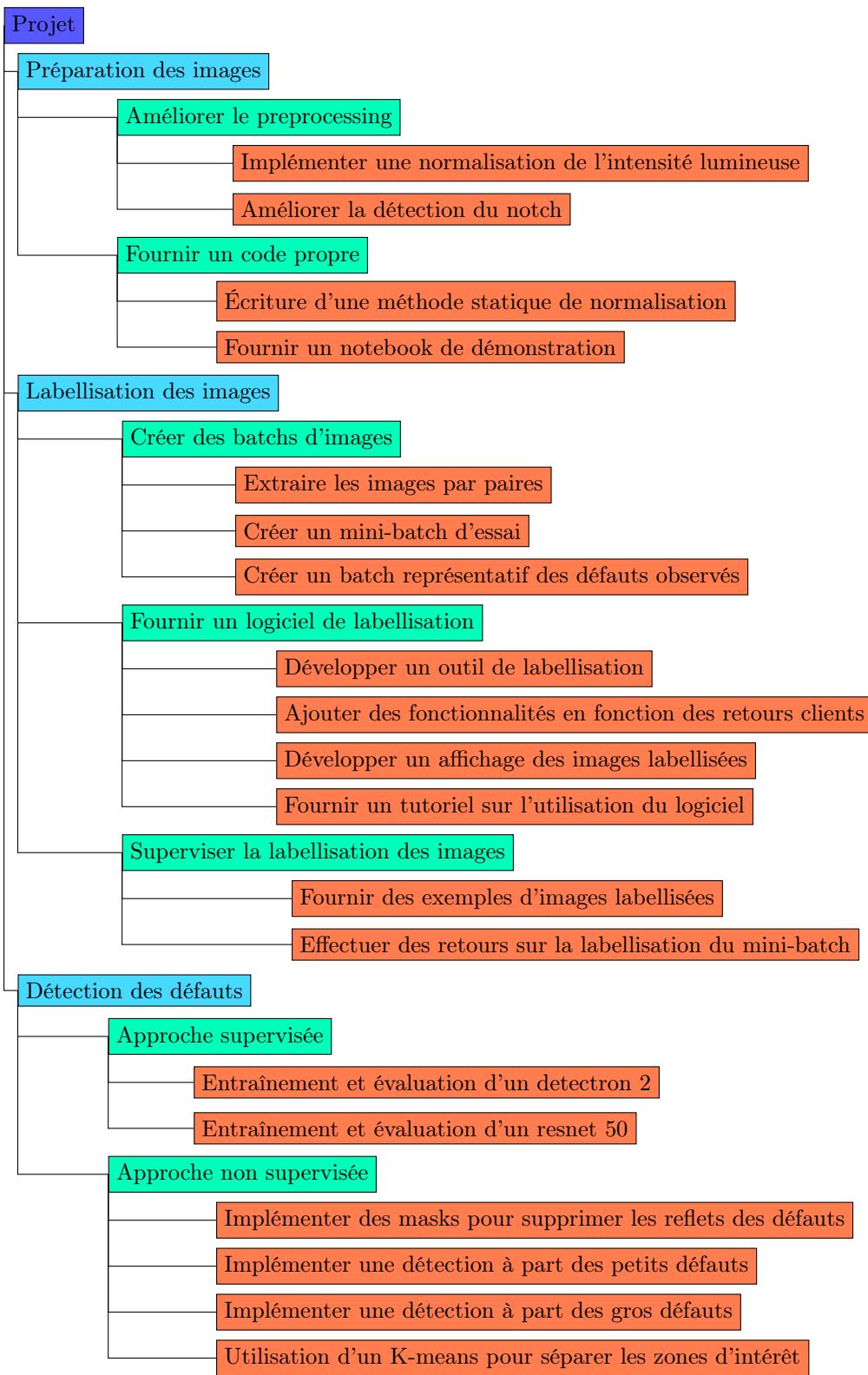


FIGURE 3.2 – Work Breakdown Structure du projet. Partie 1

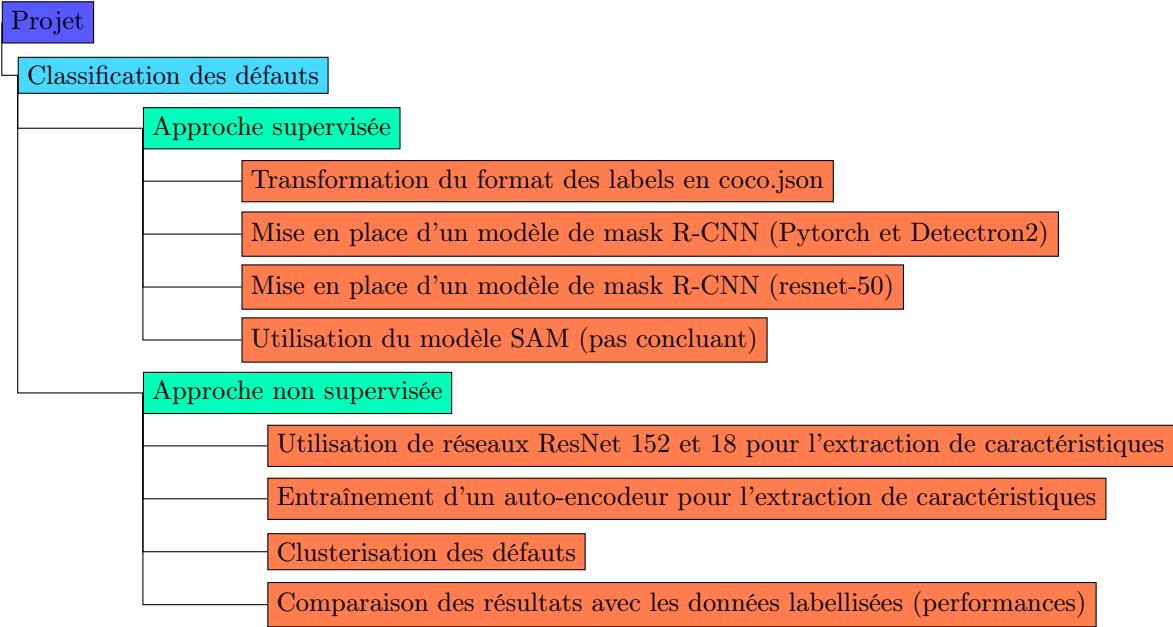


FIGURE 3.3 – Work Breakdown Structure du projet. Partie 2

## 3.2 Développement de deux méthodes

La partie que nous avons pris dès le début du projet est de développer en parallèle deux méthodes pour la détection et la classification des défauts : une méthode dite « supervisée » et une méthode « non supervisée ».

Initialement, l'objectif a été de répartir notre charge de travail entre deux tâches et de pouvoir comparer les performances des deux approches. Ensuite, nous avons eu accès à un nombre important de photos de plaques et donc de défauts, mais ces données n'étaient pas labellisées. Cela rend l'approche supervisée impossible tant qu'un travail de labellisation n'est pas fait et donc a encouragé le développement d'une approche non supervisée. Néanmoins, ce type de méthode est habituellement moins performant dans la littérature scientifique. Ainsi, nous avons en plus échangé avec le client pour lancer une labellisation des données. Enfin, une caractéristique de ces plaques est que de nouvelles classes de défauts peuvent potentiellement apparaître. Ainsi, un argument en faveur de l'approche non supervisée est qu'elle pourrait potentiellement être en capacité de les détecter puis de créer une nouvelle classe avec l'aide d'un technicien, ce que ne pourra pas permettre facilement un classifieur supervisé.

Pour ces différentes raisons et ces complémentarités, nous avons décidé de continuer dans cette deuxième partie de projet le développement de ces deux méthodes. Les tâches effectuées se répartissent donc selon ces deux approches, nous développeront enfin nos conclusions sur les cadres d'utilisation de ces deux modèles la comparaison de leurs performances.

## 3.3 Enjeu des données

Comme mentionné plus tôt, un enjeu important de cette seconde partie de projet est l'obtention de données labellisées. Nous avons échangé avec le client qui a accepté de mener cette tâche. Un logiciel de labellisation avait déjà été conçu en première partie, il a été repris afin d'intégrer de nouvelles fonctionnalités. En particulier, cette nouvelle version permettait de classer les défauts labellisés à l'aide d'une palette de couleurs qui correspondent aux classes connues.

### 3.3.1 Logiciel de labellisation

Une nouvelle version du logiciel de labellisation a donc dû être mise au point afin de mieux convenir au client. Les besoins étaient les suivants :

- Zoom/dézoom
- Labellisation d'une plaque entière et pas d'une section
- Outil gomme
- Sélection de classe de défaut

Le logiciel de labellisation a été dur à réaliser avec ces contraintes, surtout en gardant l'architecture TKinter choisie premièrement. Cependant en utilisant des astuces, il a été possible d'allier toutes ces fonctionnalités pour créer une version finale. Cette nouvelle version a été très bien reçue par le client qui a grandement apprécié les

efforts mis dans le logiciel afin qu'il possède toutes les fonctionnalités demandées et qui nous a répondu très positivement à l'envoi de la nouvelle version.

### 3.3.2 Les données reçues

La labellisation a ensuite été lancée. Pour cela, nous avons mis un point un « mini-batch » de test contenant 5 images afin de fixer avec le client ses attentes sur la labellisation. Ensuite, nous avons formé un batch de 100 images en essayant d'avoir une diversité des défauts présents, afin de garantir un entraînement efficace des modèles supervisés. Ces batchs ont été partagés au client le 22 mars.

Du côté client, la labellisation a déjà été effectuée par Samuel, en CDD au CEA. Suite à son départ au 1er avril, Adrien, notre client, a repris cette tâche. Les lots d'images labellisées que nous avons reçus sont listés dans le tableau ?? ainsi que des exemples d'images labellisées.

Le principal problème qu'il y a eu dans la première phase de labellisation a été la qualité de la labellisation. Cela a été rediscuté avec Bastien, qui encadrerait la tâche de Samuel, et nous avons du fixer que la qualité des données devait primer sur la quantité fournie. En effet, un modèle s'entraînant sur des données mal labellisées risque d'être trompé par des défauts non labellisés et donc présenter une très mauvaise performance. De fait, les données issues des trois premiers retours n'ont pas pu être exploitées pour un entraînement. Elles pourraient cependant être reprises dans une adaptation du logiciel de labellisation afin de ne pas constituer un travail perdu.

Après le départ de Samuel, la version la plus aboutie du logiciel permettant un zoom et une labellisation sur des plaques entières a été partagée au client. En ajoutant des consignes plus précises sur l'importance d'une haute qualité de labellisation, les deux derniers lots reçus ont pu constituer une base de travail pour l'entraînement des modèles de R-CNN et Mask R-CNN. Nous reviendrons dans cette partie sur l'exploitabilité des données et des choix qui ont été faits pour obtenir des performances correctes. Enfin, les données labellisées pourront aussi être utilisées pour mesurer les performances de l'approche non supervisée, augmentant encore l'importance d'avoir une labellisation de qualité.

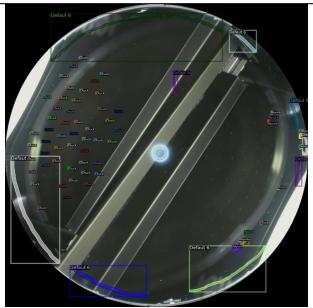
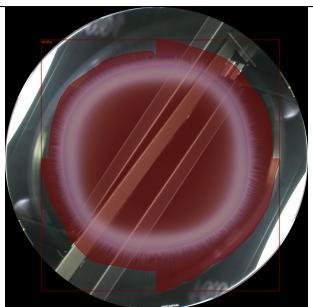
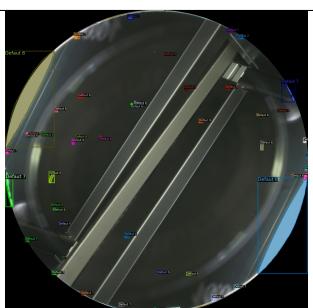
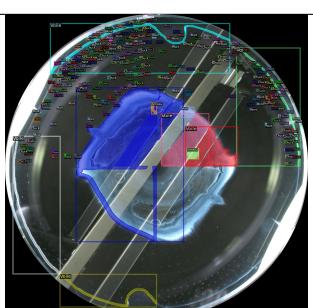
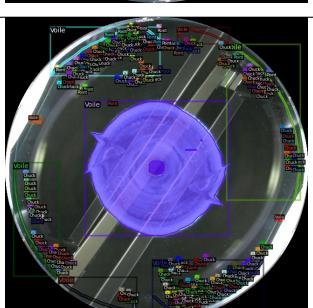
Nombre d'images	Labellisées par	Date de réception	Exemple
5 (mini-batch)	Samuel	21/03	
20	Samuel	25/03	
20	Samuel	02/04	
5	Adrien	12/04	
5	Adrien	26/04	

FIGURE 3.4 – Récapitulatif des données labellisées par le client

## 3.4 Approche supervisée

### 3.4.1 Données : passage du Faster R-CNN au Mask R-CNN

Au semestre 7, un modèle Faster R-CNN avait été entraîné. Pour rappel, les labels créés avec le logiciel de labellisation étaient donc convertis en cadres, (dites "bounding boxes"). Le modèle n'exploitait pas les contours précis des défauts. Pour prendre en compte les contours précis des défauts, il fallait donc passer à une architecture différente de R-CNN : le Mask R-CNN. Une étape importante a donc été de convertir tous les fichiers d'annotations, qui contiennent les labels et contours des défauts. En particulier, les défauts contenant parfois des "trous", il a été nécessaire d'encoder les contours des défauts selon le format RLE (Run Length Encoding), à la place du format par défaut en polygones. En effet, le format polygones, spécifie un certain nombre de sommets de chaque contour. Or il n'est pas possible pour le modèle utilisé par la suite de prendre en compte des contours intérieurs, les trous des défauts étaient donc remplis, et cela posait problème par exemple pour les défauts comme les grands voiles ou anneaux, qui peuvent occuper une grande partie de la place (le modèle autorise néanmoins les superpositions de défauts). Le format RLE, lui, est simplement un masque binaire.

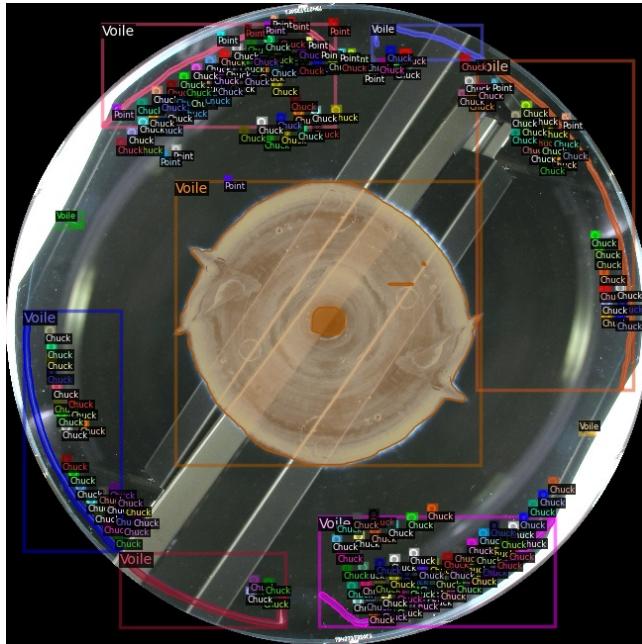


FIGURE 3.5 – Exemple de défaut contenant un contour intérieur encodé en RLE

### 3.4.2 1<sup>er</sup> modèle : Detectron2

#### 3.4.2.1 Discussion préalable

Il convient tout d'abord de discuter du dataset préparé pour l'entraînement du modèle. En effet, comme dit plus tôt, peu de données de qualité ont été labellisées. À cela s'ajoute une particularité du dataset : la fréquence d'apparition des défauts varie grandement d'une classe de défauts à l'autre. Comme vous pouvez le voir sur l'image 3.4, là où des petits points dits "chuck" apparaissent des dizaines de fois, d'autres n'apparaissent qu'une fois (comme le voile au milieu), voire pas du tout. Cela pose un problème majeur pour l'entraînement : certaines classes de défauts n'apparaissent qu'une fois dans tout le dataset, rendant la tâche périlleuse. Il n'était pas envisageable d'entraîner de manière réaliste le modèle sur le dataset pour les gros défauts. Des essais ont été menés, et le modèle n'était tout simplement pas en capacité de réaliser la détection, encore moins la classification, des gros défauts. Néanmoins, les petits défauts tels que les chuck, étaient facilement détectés. Lors d'une réunion avec le client dans laquelle a été évoqué le fait qu'il ne serait pas possible d'entraîner un modèle avec un dataset de moins de 10 images de qualité, et le fait qu'il n'était pour le moment pas possible pour le client d'entamer de nouvelles sessions de labellisation, les objectifs pour l'apprentissage supervisé ont été revus : il s'agissait maintenant de connaître l'ordre de grandeur de la quantité de données à obtenir pour permettre l'entraînement d'un tel modèle de détection.

### 3.4.2.2 Entraînement d'un Detectron2 sur les gros défauts

Afin de connaître la quantité de données à labelliser pour une détection efficace des défauts, un batch de quelques dizaines d'images contenant des gros défauts (voile, anneaux, rayures, traces de supports) a été prélevé de la base de données brute. Ces images ont ensuite été labellisées, en ne prenant en compte que les 4 gros défauts sus-cités. Cela a été relativement rapide avec le logiciel précédemment développé, puisque la partie la plus chronophage de la labellisation est celle des petits défauts qui sont présents en grand nombre à chaque image. Au final, chaque classe de défauts contenait entre 5 et 10 labels. En outre, une augmentation de données a été entreprise, avec les transformations aléatoires suivantes : changement de luminosité, brillance, contraste, saturation, rotation et symétrie axiale. L'entraînement du modèle a révélé des performances satisfaisantes, d'une

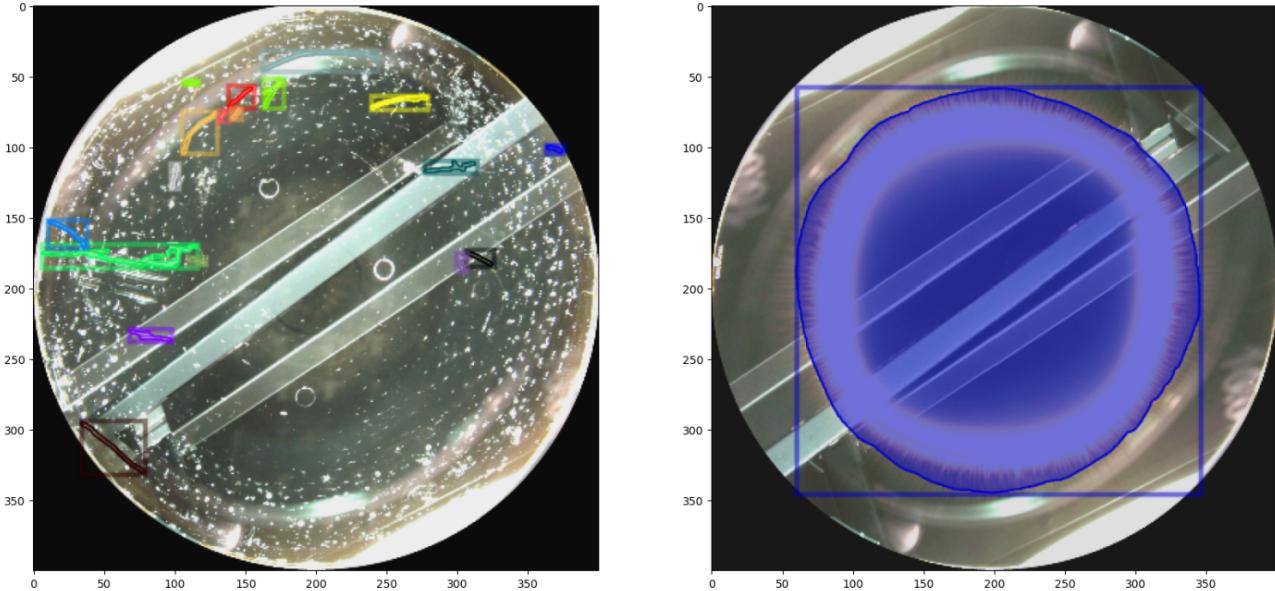


FIGURE 3.6 – Augmentation de données des gros défauts

précision de l'ordre de 80%, avec un modèle pré-entraîné basé sur un ResNet-50. Voici quelques prédictions : Une dizaine d'images par type de gros défaut devrait donc être suffisante. Concernant les petits défauts, la

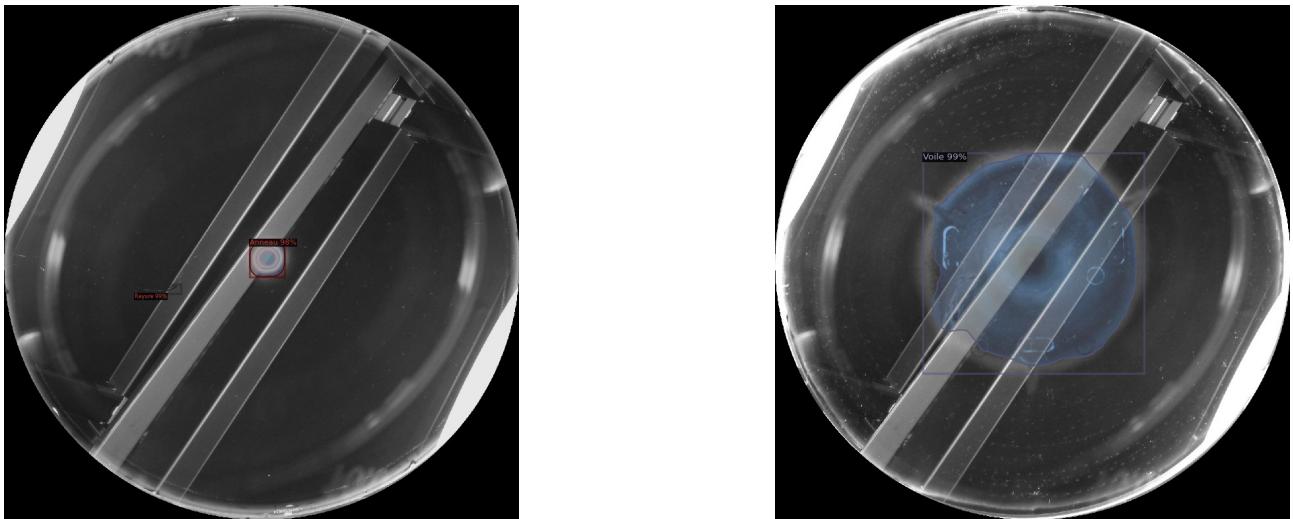


FIGURE 3.7 – Prédiction des gros défauts

quantité d'images nécessaire à l'entraînement est plus faible, et un modèle Faster R-CNN entraîné au semestre dernier avait de bonnes performances quant à la détection de ces défauts.

### 3.4.2.3 Entraînement d'un Detectron2 sur les petits défauts.

## 3.4.3 2<sup>e</sup> modèle : Entraînement d'un res-net 50 sur PyTorch

L'objectif initial était de développer un modèle Mask R-CNN autre que detectron pour pouvoir comparer les résultats. La première tentative a été réalisée en utilisant TensorFlow, mais des difficultés ont été rencontrées pour configurer CUDA, rendant impossible l'utilisation de l'accélération GPU.

Un passage à PyTorch a été effectué pour tenter de résoudre ce problème. Cependant, l'entraînement du modèle était trop lent en raison de la taille réduite du dataset (seulement une vingtaine d'image pour l'entraînement et la validation) et des contraintes matérielles. La mémoire GPU disponible était une autre limitation majeure, ne permettant pas de traiter des batchs de plus de trois images à la fois. De plus, la taille de la couche cachée du modèle a dû être limitée à 400 unités pour respecter les capacités de mémoire.

Pour surmonter ces défis, un modèle Mask R-CNN pré-entraîné a été utilisé. Ce modèle a ensuite été ajusté pour notre tâche spécifique de détection d'impuretés, en utilisant des transformations et des augmentations de données pour améliorer les performances.

L'augmentation des données a joué un rôle crucial dans l'amélioration des résultats du modèle. Un script à part a été utilisé pour générer des variations des images d'entraînement, augmentant ainsi la diversité des données et améliorant la robustesse du modèle. Les images ont été transformées par retournement horizontal et vertical, rotation aléatoire dans une plage de -25 à 25 degrés, ajustement de la luminosité pour simuler des variations d'éclairage, et zoom aléatoire. Pour chacune de ces transformations, les coordonnées des boîtes englobantes et des masques ont été recalculées pour s'adapter aux nouvelles dimensions et orientations des images. Ces transformations ont généré un ensemble de données augmenté, comprenant des images transformées avec des annotations ajustées.

### 3.4.3.1 Métriques employées pour le res-net 50

Pour évaluer les performances du modèle, plusieurs métriques ont été utilisées, spécifiquement fournies par la bibliothèque torchmetrics.detection.mean\_ap :

#### mAP (Mean Average Precision)

- mAP global : Mesure la précision moyenne pondérée en fonction de la proportion de prédictions correctes sur toutes les classes, en prenant la moyenne de l'Average Precision (AP) pour différents seuils d'Intersection over Union (IoU).
- mAP@50 et mAP@75 : Mesurent la précision moyenne à des seuils d'IoU de 0.50 et 0.75 respectivement, fournissant une vue détaillée de la performance du modèle à différents niveaux de tolérance pour les prédictions correctes.
- mAP par taille d'objet : Le mAP est également évalué pour les petits, moyens et grands objets, permettant ainsi de comprendre comment le modèle performe pour des objets de différentes tailles.

#### mAR (Mean Average Recall)

- mAR global : Évalue la capacité du modèle à détecter toutes les instances d'objets présents dans les images. Cette métrique est calculée pour différents nombres de prédictions (par exemple, les meilleures 1, 10 et 100 prédictions).
- mAR par taille d'objet : Le mAR est aussi évalué pour les petits, moyens et grands objets, ce qui aide à comprendre si le modèle manque certains objets en fonction de leur taille.

**mAP et mAR par classe** : ces métriques sont calculées pour chaque classe individuellement. Cela permet de vérifier si le modèle a des biais envers certaines classes et assure que le modèle performe de manière équilibrée sur toutes les catégories d'impuretés. Toutefois, des valeurs anormales peuvent indiquer un problème potentiel dans le calcul ou la collecte des données.

Ces métriques sont essentielles pour comprendre la précision et la fiabilité du modèle dans la détection et la segmentation des impuretés. Le mAP, en particulier, est crucial car il prend en compte à la fois la précision et le rappel, offrant ainsi une vue équilibrée des performances du modèle. En calculant ces métriques, nous pouvons évaluer de manière exhaustive et rigoureuse les capacités du modèle à détecter et segmenter les impuretés dans les images.

### 3.4.3.2 Résultats et perspectives du resnet 50

On a ici utilisé les boîtes englobantes pour la représentation des résultats mais le modèle produit surtout les masks pour détecter les objets. On peut observer une amélioration visuelle significative lorsqu'on utilise des données augmentées pour entraîner le modèle. Les métriques aussi augmentent, la mAP passant de 17% à 32%, et le mAR large passant de 56% à quasiment 100%. Les petits objets restent difficile à détecter pour le modèle, et sont une voie d'amélioration pour ce dernier.

Ground Truth  
Prediction

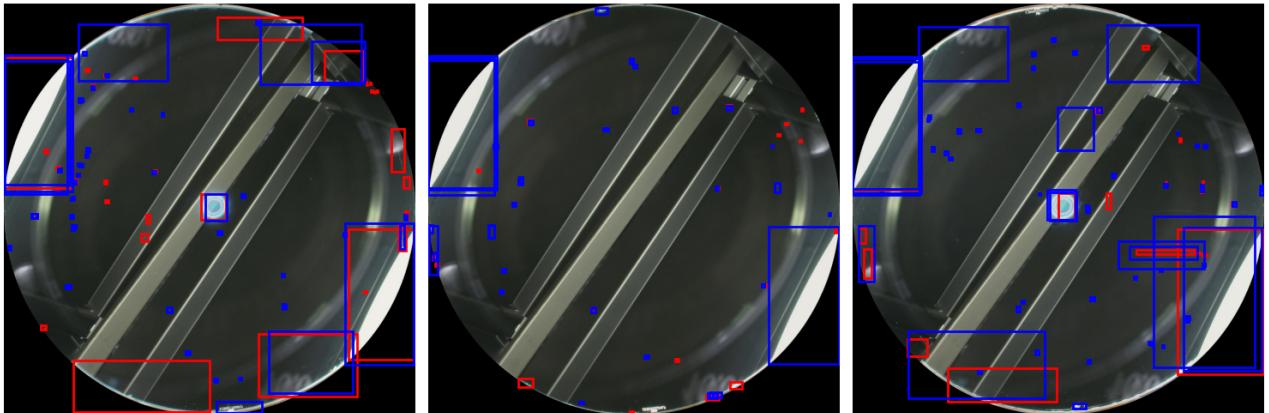


FIGURE 3.8 – Validation visuelle du modèle entre prédictions et labels sans données augmentées

Ground Truth  
Prediction

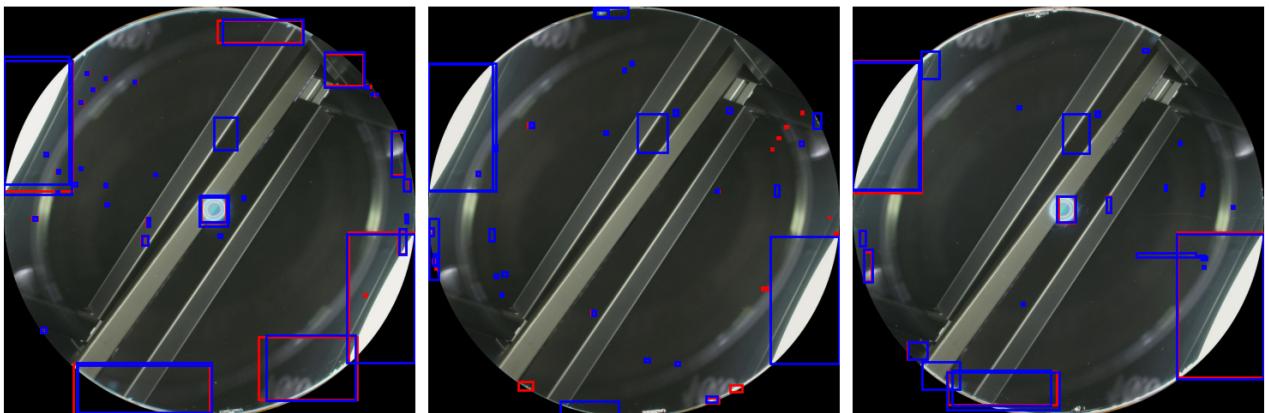


FIGURE 3.9 – Validation visuelle du modèle entre prédictions et labels avec des données augmentées

## 3.5 Approche non supervisée

### 3.5.1 Reprise de la détection

À la fin du premier semestre, nous avions réussi à mettre en place une détection qui proposait un bon compromis entre fausse détection, détection des fines rayures, mauvaise agglomération de défauts et agglomération totale de bords correspondant à un même défaut. Cela donnait le résultat visible sur la figure 3.10a. On peut voir que l'ensemble des défauts est bien détecté, mais cependant on détecte aussi un reflet qui apparaît sur la totalité des images. De plus, cela engendre un problème d'agglomération des gros défauts avec ce reflet.

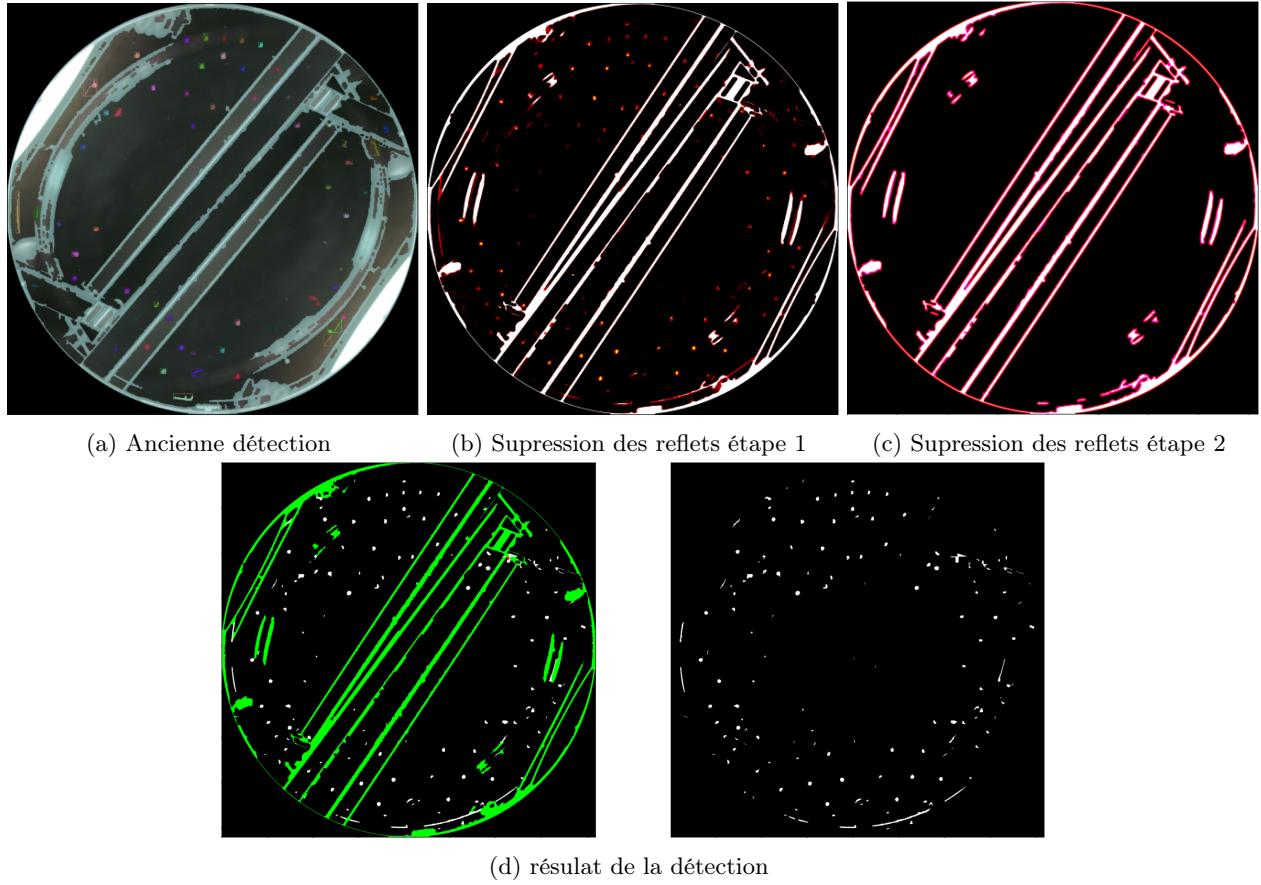


FIGURE 3.10 – La détection de petits défauts

Lors du premier semestre, nous avions essayé de supprimer les reflets en soustrayant un masque représentant la même plaque si elle n'avait pas de défauts. Cela n'avait pas été concluant puisqu'il était difficile de créer un tel masque, d'autant plus que la plupart des plaques que nous avions à l'époque possédaient une multitude de défauts. Afin d'améliorer la détection, nous l'avons découpée en deux parties : une détection des petits défauts et une détection des gros défauts.

### 3.5.1.1 Les petits défauts

Nous avions vu au semestre précédent que certains paramètres permettaient de mieux détecter ou bien les défauts de petite taille ou bien ceux de grande taille. Ainsi, en optimisant ces paramètres uniquement pour les petits défauts, la quantité de reflets détectés est bien moindre (en blanc, rouge et jaune 3.10b) comparée à notre ancienne détection. Une fois ceci fait sur une partie du jeu de données, nous concaténons les résultats pour obtenir à chaque endroit de l'image la probabilité que la zone soit détectée. On pose des seuils de manière arbitraire pour supprimer les zones qui sont rarement détectées (qui correspondent donc à des défauts) en rouge. On voit en jaune que certaines zones qui ont l'air de correspondre à des défauts sont toujours présentes. Afin de les faire disparaître on ajoute un critère nécessitant que la zone soit assez grande, ce qui donne la zone en blanche.

Une fois ce premier masque obtenu, on va ajouter une certaine sécurité : une augmentation de la taille des bords. Cela présente l'inconvénient de supprimer des zones de détection potentielles. Pour ce faire, on procède en deux étapes (voir figure 3.10c). Dans un premier temps, on augmente la taille des bords si la probabilité calculée localement dépasse un certain seuil, puis l'on ajoute un bord supplémentaire de manière uniforme. Cela a l'intérêt d'ajouter une marge de sécurité plus importante là où le risque est plus élevé. On aurait pu passer plus de temps sur cette mise en place du masque et plus précisément sur l'élaboration des seuils ainsi que des marges de sécurité mais ce temps a été, selon nous, mieux investi dans le reste du projet. Une fois ceci fait nous obtenons la détection 3.10d en blanc et en vert ce qui a été supprimé.

### 3.5.1.2 Les gros défauts

Nous nous référons dans ce paragraphe à la figure 3.11. Pour ce qui est des gros défauts, leur caractéristique principale est qu'ils sont de taille importante et donc que l'on a trop d'informations, on peut donc réduire

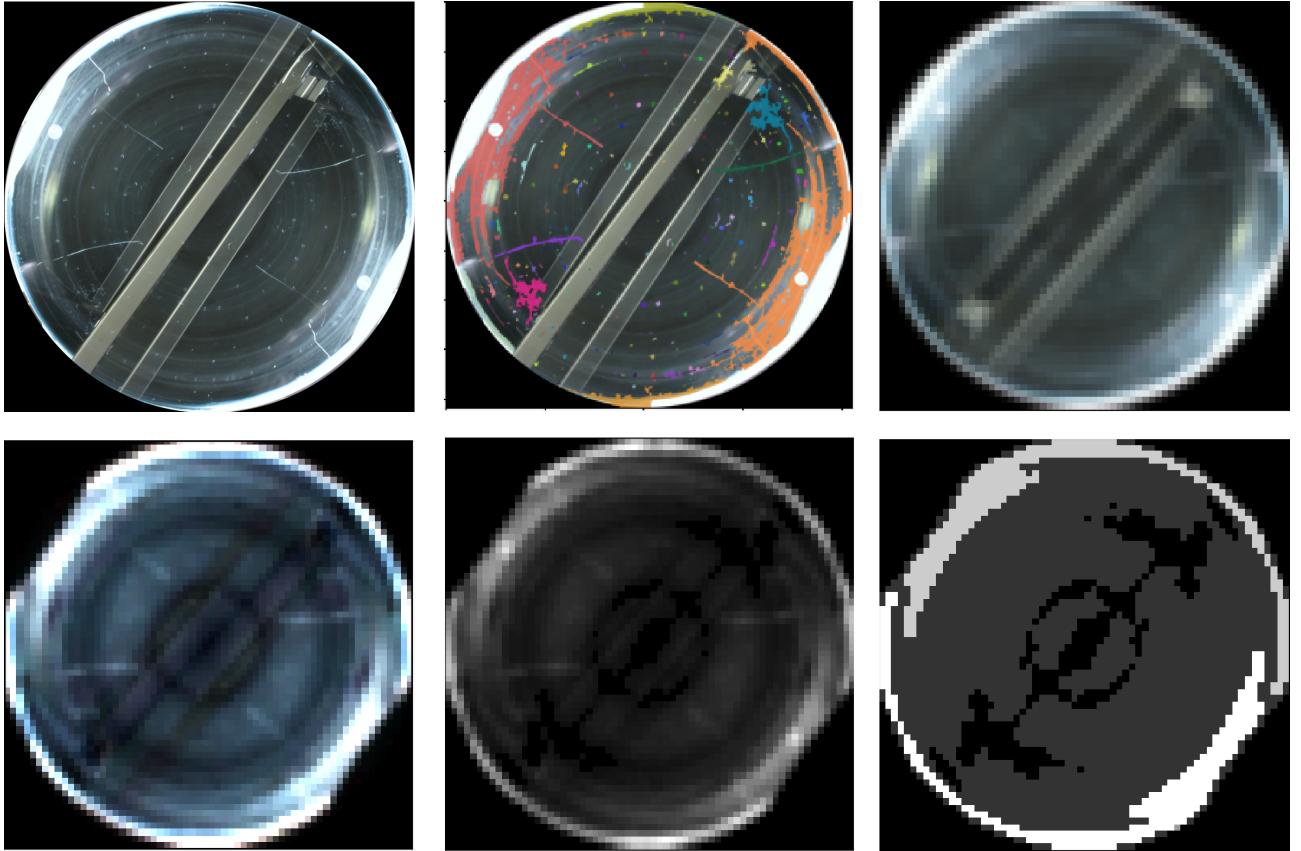


FIGURE 3.11 – Détection des gros défauts

drastiquement la qualité de l'image. Pour ce faire, nous avons d'abord réduit la taille de l'image de (4096,4096) à (128,128) à l'aide d'une interpolation linéaire suivie d'un flou gaussien pour effacer au mieux toute trace des petits défauts (qui sont bien détectés par l'autre algorithme image 2) et enfin d'une interpolation de type "area" pour passer à (64,64). On voit sur que cette compression (image 3) permet de mettre en évidence le gros défaut sur quelques pixels.

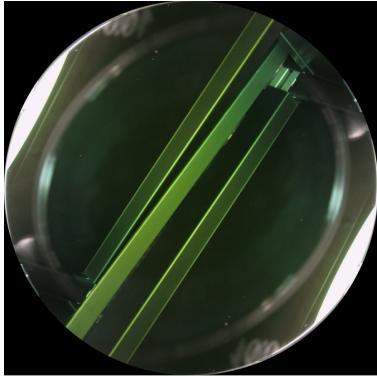
De plus, puisqu'on a réussi à gommer les défauts de petite taille, on peut former une image moyenne qui sera alors dépourvue de défaut. Ainsi, en prenant la valeur absolue de la différence, nous obtenons l'ensemble des défauts sur la plaque (image 4). Cela marche extrêmement bien sur les plaques habituelles (grises) puisque l'on a normalisé le contraste et la luminosité à partir du reste de l'image qui est invariant (hormis au réglage de l'appareil photo). Cependant nous avons découvert trop tard qu'il existait des traitements modifiant la couleur des plaques utilisées au CEA, ce qui rend cette méthode bien moins bonne, apportant des fausses détections comme ici 3.12a. Nous n'avons pas pris le temps de régler ce problème afin de passer à la suite. Des pistes de solutions pourraient être : traiter l'image en noir et blanc, chercher les paramètres optimaux pour transformer la plaque de couleur en plaque habituelle afin de minimiser la quantité de zone détectée.

Une fois l'ensemble des gros défauts détectés, nous souhaitons les séparer. En effet, il peut y avoir plusieurs gros défauts sur une même plaque. Ici par exemple 3.11, la plaque présente un voile et son bord très bleuté. Cette partie de l'algorithme offre des résultats satisfaisants mais pourrait être améliorée puisque son fonctionnement ne raisonne que sur l'intensité et ne prend pas en compte la localisation. En effet, on effectue plusieurs *KMeans* sur l'intensité et on compare les résultats à l'aide du silhouette score. Cela marche globalement bien pour distinguer des grosses traces de chuck du voile créé, de même pour les anneaux. Cependant si le voile (ou la trace de chuck) est de même intensité que la bord, alors les deux défauts ne sont pas séparés 3.12b.

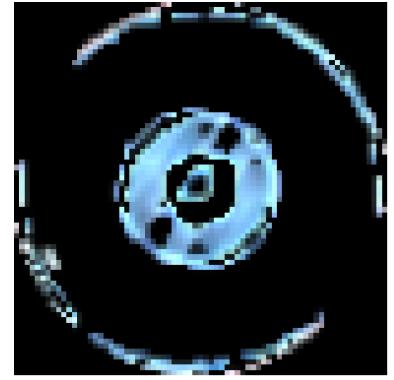
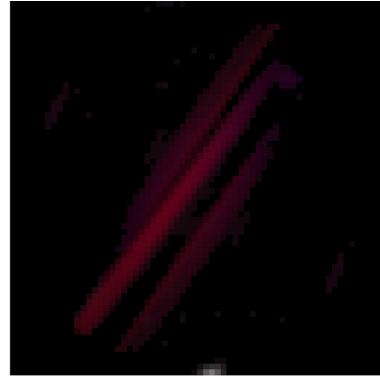
Pour ce qui est des fausses détections, autant pour les gros défauts que pour les petits défauts, nous laissons cette tâche à l'algorithme de classification.

### 3.5.2 Classification

Pour la classification, le client était plus intéressé par les gros défauts plutôt que les petits qui viendraient alors dans un second temps. Pour ce faire nous avons décomposé le travail en une extraction de features de qualité et une clusterisation de ces features.



(a) Problème des plaques de couleurs



(b) Non séparation du bord et du chuck

FIGURE 3.12 – Problème détection de gros défauts

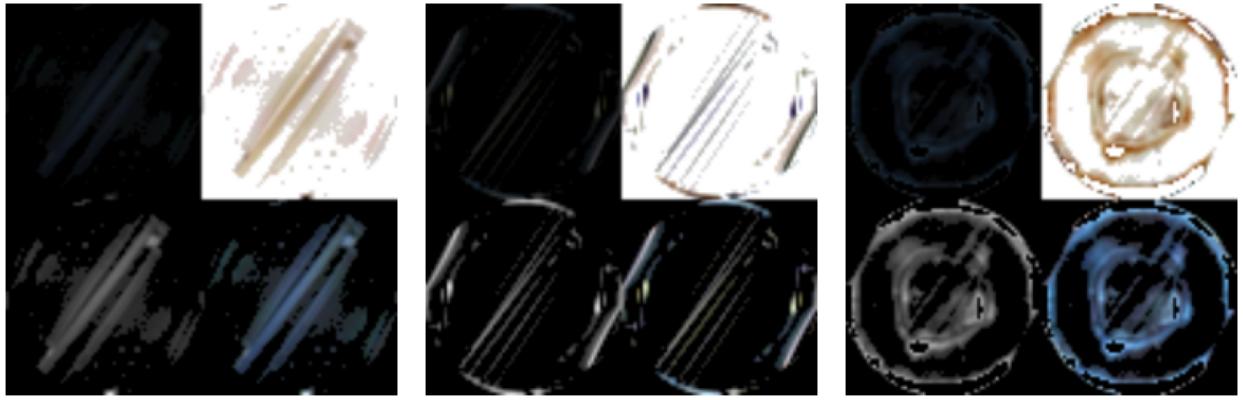


FIGURE 3.13 – Entrée du ResNet18

### 3.5.2.1 Extraction de features

Pour extraire les features, nous avons dans un premier temps utilisé un réseau de neurones pré-entraîné puisque cela pouvait s'avérer suffisant. Nous avons essayé un resnet 18 et un resnet 152 et avons opté pour le 18. En effet, ses 512 features nous semblaient largement suffisantes, d'autant plus que nous allons devoir passer à un nombre de features inférieur à environ 50 pour éviter le *curse of dimensionality*. Dans un second temps, nous avons essayé d'utiliser une pretext task afin d'avoir de meilleures features, ce qui est développé dans la sous-partie 3.5.3. Cependant l'un des problèmes soulevé est que ou bien on normalisait les couleurs et l'intensité et dans ce cas les features extraites ne dépendaient plus du tout des couleurs ni de l'intensité (ce qui est dommage car ces paramètres sont souvent pertinents pour classifier un défaut) ou bien les features extraites ne permettaient pas de rapprocher un même défaut présent sous deux couleurs différentes. Finalement, nous avons résolu ce problème en donnant une image concaténée composée de l'image originelle, une image d'intensité normalisée, une en noir et blanc et enfin une avec les couleurs inversées (en effet un défaut vert en fonction de sa position dans l'image pourra apparaître vert ou rouge selon la couleur normale de la plaque à cette endroit, conséquence directe de la valeur absolue) ce qui donne l'entrée suivant du modèle 3.21.

### 3.5.2.2 Clusterisation

Après classification nous savons que notre dataset composé d'environ 3800 détections possède 3000 fausses détections. Ainsi, un premier enjeu va être d'éliminer ces fausses détections qui, de par leur sur-représentation nous empêcheront d'obtenir de bon résultats. Pour ce faire nous allons utiliser un modèle DBSCAN où l'on va chercher à optimiser à la fois le score de silhouette, un nombre de classe raisonnable (ni trop grand ni trop faible) et une proportion de défauts classés la plus grande possible. Ce dernier point est justifié car le silhouette score n'est pas impacté par les données classées comme outlier. Ainsi il est normal d'obtenir un plus grand score de silhouette si l'on ne prend que les données qui nous arrange. Pour ce qui est du nombre de classe, on se sert en réalité de la clusterisation comme d'une méthode rapide de labellisation. En effet, il n'est pas raisonnable de penser que le modèle va pouvoir rassembler de lui-même toutes les mauvaises détections puisque celle-ci ne sont pas forcément proche dans l'espace des caractéristiques ; même de bonnes features ne peuvent garantir

cela. De même pour les éléments d'une même classe : deux défauts de type voile peuvent être de nature très différente. Pour optimiser ces trois critères nous pouvons jouer sur le nombre de composantes gardées lors de l'ACP (ou *PCA*) et le rayon  $\epsilon$ . Ce qui nous donne le résultat suivant : ???. Nous avons aussi tracé ces graphiques lorsque les données ne sont pas labellisées avant l'ACP et nous avons observé que la normalisation permettait une plus grande proportion d'individus classés à score égal. Ainsi, nous avons choisi 15 dimensions et un epsilon (recalcué de manière plus précise) de 3.36. Cela donne 105 classes, 20% de non assignés et un silhouette score de  $0.395 \in [0.25, 0.5]$  correspondant donc à un cluster de qualité moyenne. Les vrais défauts se trouvaient parmi 21 classes (ne contenant aucune fausse détection) ainsi que dans les non détection. Les fausses détections représentaient donc toutes les autres classes et 10% d'entre eux se trouvaient dans la classe des outliers. Il a donc fallu labelliser 21 classes comme défaut potentiel (ainsi que la classe des outliers) ce qui a pris 2 minutes, pour référence une classe ressemble à ceci???. Ensuite puisque les défauts existent souvent en quantité très limitée on peut supposer que les densités associées sont très variables (notamment la classe des défauts : bord lumineux possède 346 défauts contre 23 pour les anneaux) c'est pourquoi on a opté pour l'algorithme HDBSCAN qui résout ce problème. Nous avons optimisé les paramètres de celui-ci afin d'avoir de même le moins d'outliers, un nombre de classe raisonnable et un silhouette score maximal??. Ce qui nous a poussé à choisir un alpha = 0.7 et un epsilon = 8.3. On a donc labellisé les 80 classes ce qui a été plus fastidieux (de l'ordre de 10 minutes). Il reste donc des données non labellisées que l'on va donc ignorer pour la suite. Il est aussi à noter que quelques fausses labellisations ce sont glissées dans le jeu de données mais cela reste négligeable. Enfin pour avoir notre modèle de prédiction on va utiliser un K-NN légèrement modifié, c'est à dire que l'on ne va considérer que les voisins à distance suffisante.

### 3.5.2.3 Résultats

Ici on va pouvoir évaluer à quel point nos features sont bonnes. En effet on a utilisé depuis le début le silhouette score qui indique à quel point les clusters sont cohérents, maintenant que nous avons les vrais labels (du moins pour 95% du dataset) nous pouvons voir qu'en considérant la classe des fausses détection on a silhouette score de -0.027 qui traduit principalement à quel point les fausses détections peuvent prendre plusieurs formes. Si l'on ne considère que les vraies détections, on trouve 0.2753 ce qui traduit de même qu'un même défaut a différente manifestation (c'est notamment le cas des voiles), mais aussi que les features ne sont pas excellente d'où l'intérêt de la partie ??ur ce qui est des résultats, il est à noter que les paramètres du classifieur ont été optimisés sur les données sur lesquelles il a été testé. Ce n'est pas une bonne chose en soit mais les gains observé sont relativement faibles comparés à des valeurs arbitraires (faisant sens). On a donc opté pour un KNN et pour l'évaluer nous avons fait du leave-one-out ainsi que "leave-batch-out". En effet les données arrivent par batchs qui correspondent à un même traitement, ce qui implique généralement les mêmes type de défauts, qui sont de plus très semblables pour un même batch. C'est cette deuxième méthode que nous avons étudié plus en profondeur. On obtient donc la matrice de confusion suivante 3.17 sur laquelle on peut observer que la ligne des -1 ne peut être interprétée : Défauts qui n'ont pas été labellisés. On voit aussi que la classe -1 est l'un des plus gros second choix, le classifieur est donc prudent. En revanche on voit que la classe n°6 3.18 est souvent confondue avec une fausse détection ce qui est une mauvaise chose. De même 3 et 4 sont confondus mais ce n'est pas étonnant puisque les classes se ressemblent beaucoup. Cela donne le classification report suivant 3.1 qui montre de plutôt bon résultats. Ces résultats on été montrés en "leave-batch-out", cependant si l'opérateur est en train d'opérer les données, il aura ces résultats là pour la première plaque d'un batch. Au fur et à mesure qu'il opère le batch, il aura de bien meilleurs résultats comme on peut voir ici 3.19, rendant donc très évolutif le modèle.

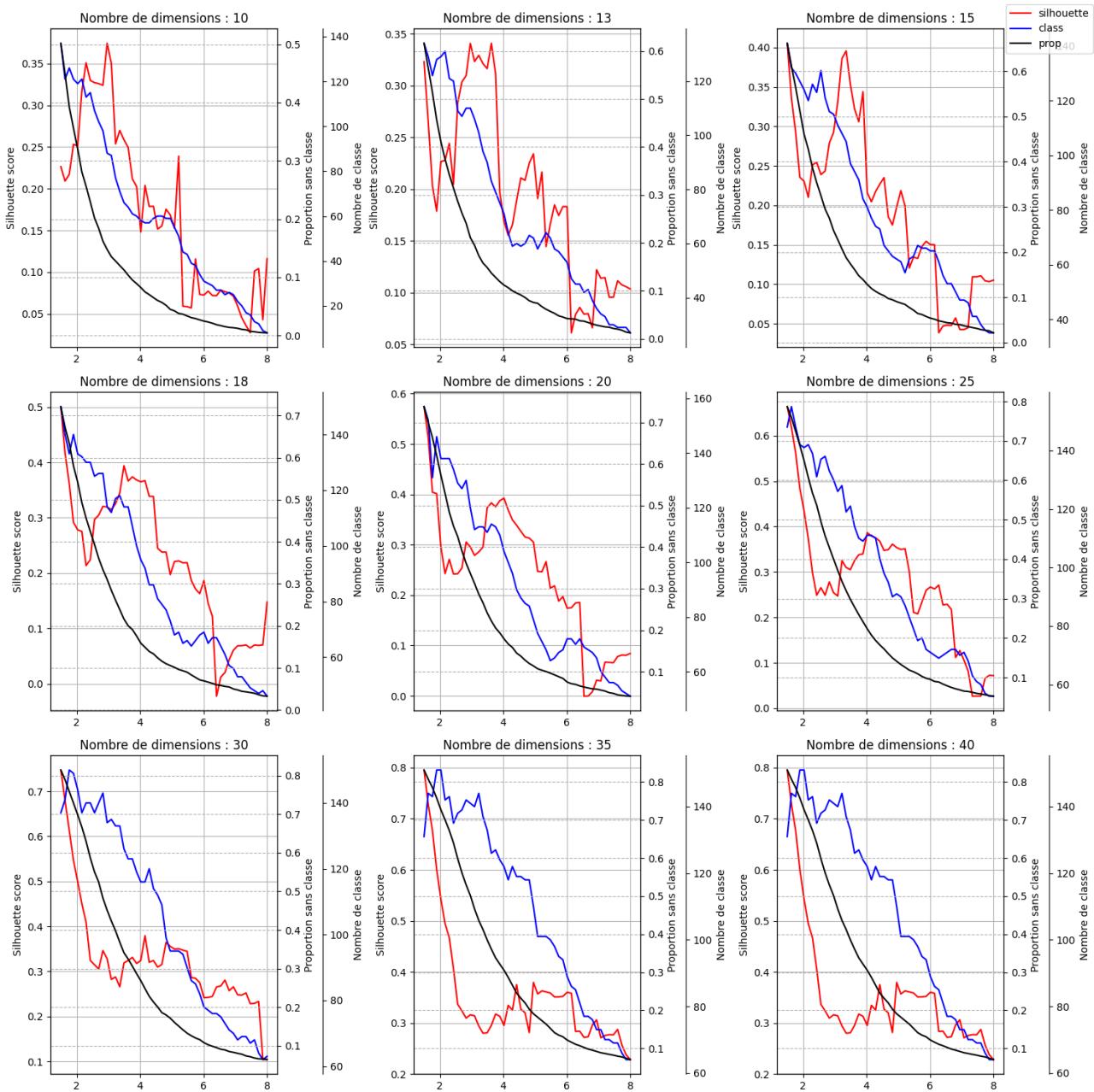


FIGURE 3.14 – Choix de  $\epsilon$  et du nombre de composantes

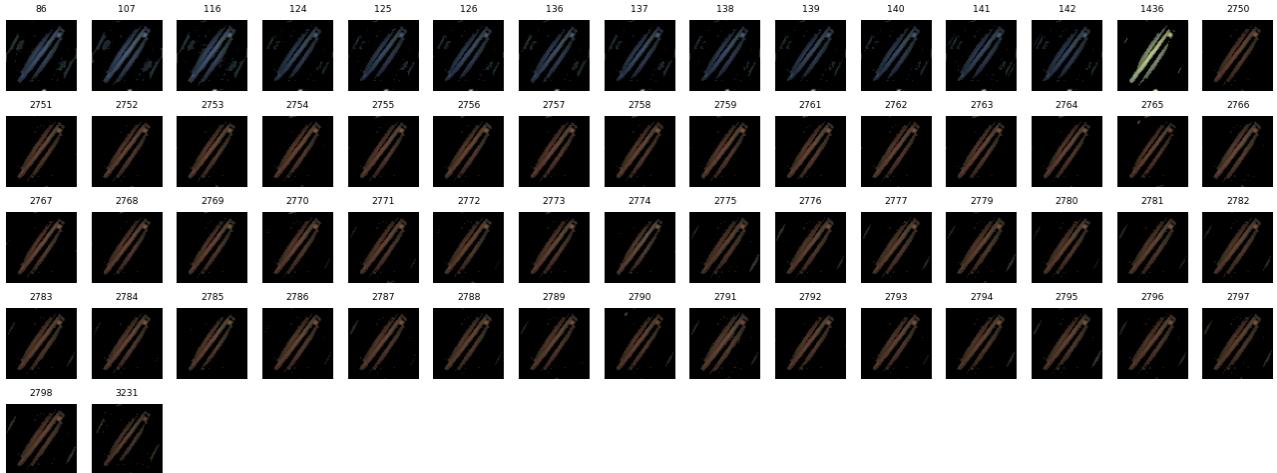


FIGURE 3.15 – Exemple de classe de fausse détection

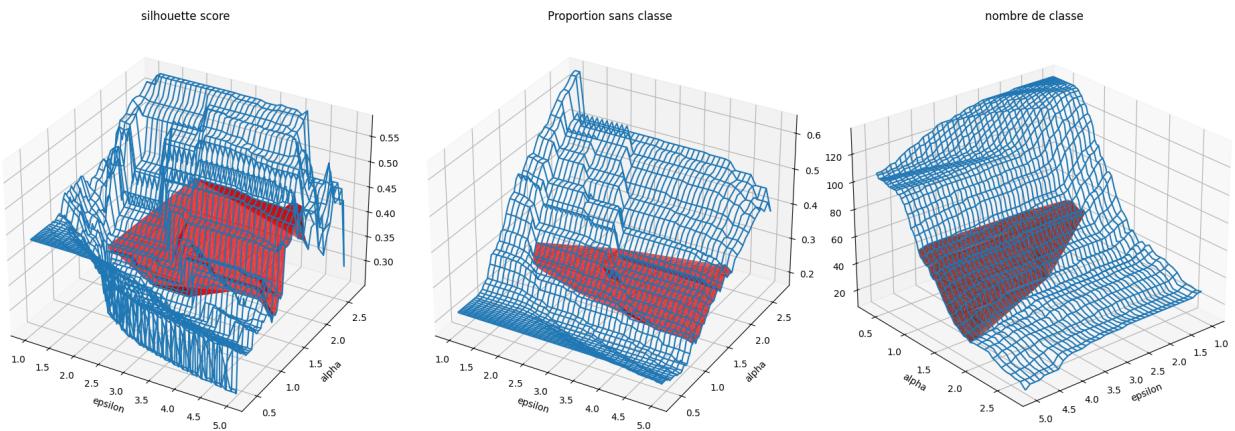


FIGURE 3.16 – Choix optimal de epsilon et alpha (en rouge zone acceptable : Silhouette score  $> 0.3$ , proportion de non classés  $\in [0.1, 0.25]$  et nombre de classes  $\in [30, 70]$ )

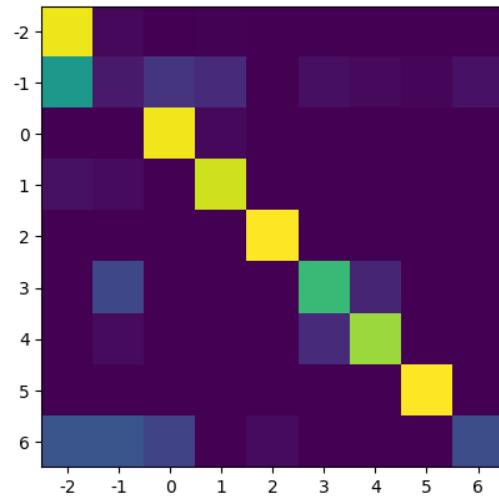


FIGURE 3.17 – Matrice de confusion pour les gros défauts

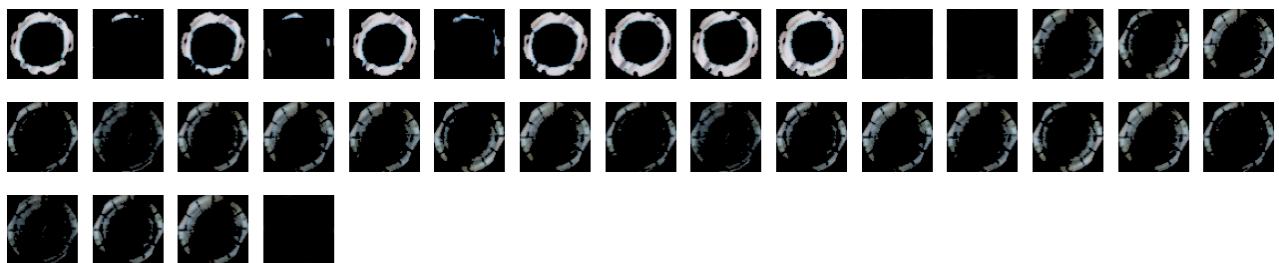


FIGURE 3.18 – Classe n°6

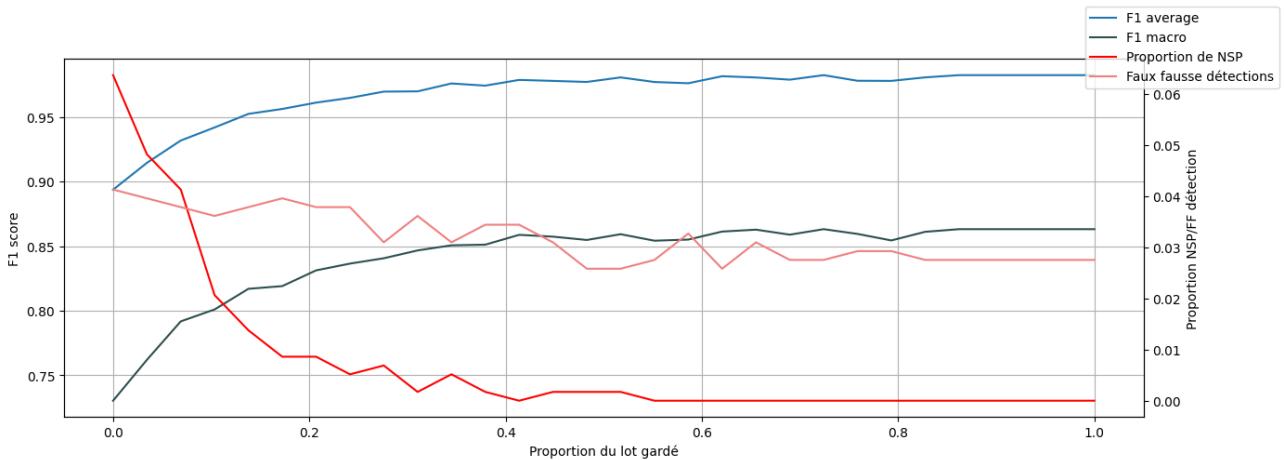


FIGURE 3.19 – Impact de la connaissance du batch sur la détection d'un nouvel individu

	Precision	Recall	F1-Score	Support
<b>-2</b>	0.97	0.97	0.97	3162
<b>-1</b>	0.08	0.07	0.08	126
<b>0</b>	0.65	0.98	0.78	50
<b>1</b>	0.91	0.93	0.92	346
<b>2</b>	0.96	1.00	0.98	23
<b>3</b>	0.85	0.68	0.76	78
<b>4</b>	0.72	0.85	0.78	33
<b>5</b>	0.89	1.00	0.94	17
<b>6</b>	0.53	0.24	0.33	34
<b>Accuracy</b>			0.93	3869
<b>Macro Avg</b>	0.73	0.75	0.73	3869
<b>Weighted Avg</b>	0.92	0.93	0.92	3869

TABLE 3.1 – Precision, Recall, F1-Score, and Support for different classes

### 3.5.2.4 Cas des petits défauts

Forts de notre apprentissage sur les gros défauts, nous allons essayer de transposer la même méthode sur les petits défauts. Ici notre data-set est composé de 177000 défauts mais les défauts rencontrés sont beaucoup moins variés. En effet on recense principalement des fausses détections, des traces de chucks et des rayures. Ainsi on va pouvoir s'éviter de séparer la partie suppression des fausses détections puisque ceux là ne représentent qu'une partie modérée des données.

Même si les défauts sont très peu variés, ils sont très semblables. C'est pourquoi on va utiliser plutôt un ResNet-152 (qui est plus gros, mais cela n'est pas un problème puisque nous faisons seulement de l'inférence). On ne va pas réutiliser la concaténation pour l'entrée du modèle puisque les arguments précédents ne tiennent pas ici. Le data set étant beaucoup plus gros nous n'allons pas pouvoir faire tourner plusieurs fois les modèles non-supervisés pour optimiser les paramètres. Nous allons donc pour le nombre de dimensions utiliser la règle du pouce des 80% qui donne donc 60 dimensions. De même nous allons principalement nous servir de la partie non-supervisée comme d'un accélérateur de labellisation. Ainsi, nous allons utiliser un KMeans (algorithme suffisamment rapide) avec 150 classes qu'il faut labelliser à la main (10 minutes).

### 3.5.2.5 Deep clustering

Étant donné que le code du modèle SCAN[3] était fourni, nous l'avons adapté afin d'entraîner le modèle sur nos données. Il était difficile de monitorer les trois étapes car que nous n'avions pas de label. Cela a donné des

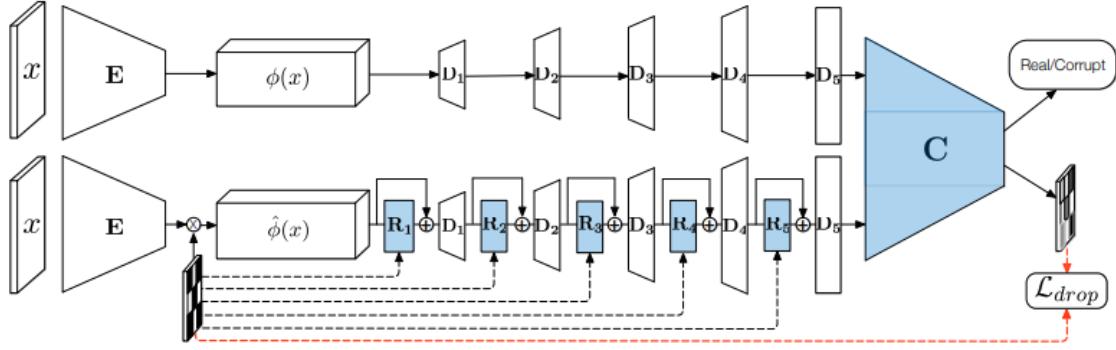


FIGURE 3.20 – Architecture du GAN

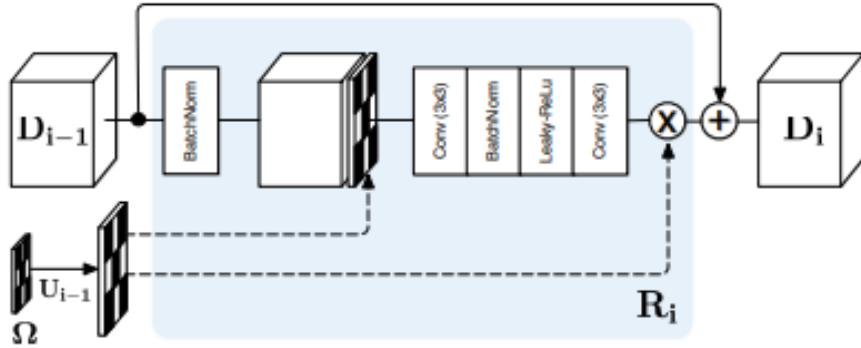


FIGURE 3.21 – Architecture d'une couche de repair

résultats dépourvus de sens malgré une pretext task qui semblait concluante (les proches voisins d'un individu étaient bien de la même classe). Nous n'avons pas consacré plus de temps à cette méthode.

### 3.5.3 Extraction de meilleures caractéristiques

L'idée est donc d'obtenir de meilleures features en utilisant une pretext-task (cf état de l'art) afin de pouvoir mieux classifier nos défauts. La méthode choisie est celle de cet article [11]. On choisit donc cette architecture pour le GAN :

L'idée est donc d'entraîner un auto-encodeur. On le constitue de 4 couches d'encoding et 4 couches de decoding (de simples BatchNorm, Transpose, ConvTranspose2D et UpSample). On ne modifie ensuite plus ces couches là. On duplique cet auto-encodeur pour ajouter au niveau du bottleneck un masque aléatoire qui perd environ la moitié des features compressées. On intercale ensuite des couches de repair avant chaque couche du decodeur afin de reconstruire l'information perdue par le drop.

Chaque couche de repair est constituée ainsi :

On entraîne finalement, en suivant un modèle de type GAN, en même temps un Resnet18 avec deux classes en sortie qui servira de discriminant et les 4 blocs de repairs qui génèrent les artéfacts. On met aussi en sortie du discriminant la prédiction du masque utilisé, qui correspond donc aux endroits où les artéfacts sont dans l'image.

On récupère ensuite les features en utilisant l'auto-encodeur et le modèle Resnet18 en retirant sa dernière couche qui ne sert qu'à classifier. On a alors une représentation plus précise des caractéristiques des défauts permettant de mieux les classifier.

## Chapitre 4

# Conclusion

La particularité de notre projet est la grande liberté que le client nous laisse quant aux solutions apportées. Ainsi, nous devons mener un gros travail de recherche et de tests pour trouver la méthode qui fonctionnera le mieux sur le jeu de données qui nous est fourni. Cela se traduit dans notre travail par l'essai de nombreuses approches à chaque étape du projet. Ensuite, chaque possibilité envisagée doit être testée sur les images que nous fournit le CEA-Leti. Cette étape révèle souvent de nouveaux problèmes dans notre solution, ou bien les résultats sont décevants et nous devons affiner notre approche pour en améliorer les performances. L'ajout que constitue cette poursuite du sujet est la gestion de la labellisation des images. C'est une tâche laborieuse, mais donc la qualité est essentielle pour assurer une bonne performance des modèles. Cet enjeu a nécessité une coopération et une communication accrue avec le client. Cela a permis d'obtenir des images labellisées, mais leur qualité puis leur quantité ont fait défaut pour l'entraînement du modèle supervisé. Notre client a insisté sur la penibilité de la tâche de labellisation, cela a encouragé l'adaptation de la méthode non supervisée pour assister une future labellisation. En effet, les résultats prometteurs obtenus avec une labellisation spécifique encouragent à terme l'utilisation d'une méthode supervisée avec une constitution d'une base de données complète grâce à l'approche non supervisée.

La complexité de ce projet est double. D'abord, la multitude des variétés de défauts possibles sur une image nous force à trouver des approches qui soient les plus générales possibles, ou bien de définir plusieurs modes de traitement de l'image pour recouvrir toutes ces possibilités. Cela a motivé en première partie de projet la conservation des deux méthodes en parallèles. Pour la suite du projet, cela nous a obligé à développer des outils (en particulier pour la labellisation) qui pourront se généraliser à de nouveaux défauts. Plus précisément, le choix d'entraîner des modèles pour rendre la labellisation plus efficace permettra aussi au client de ré-entraîner les modèles si le besoin apparaît à l'avenir. La volonté de garder ces deux approches a porté ses fruits en fin de projet. En proposant d'assister la labellisation avec l'approche non supervisée, nous nous appuyons sur la complémentarité des deux méthodes pour proposer une solution riche et adaptable. Ensuite, une deuxième source de complexité est liée à la précision que nous voulons obtenir avec nos algorithmes. Il est en effet assez aisément de détecter grossièrement des défauts (souvent confondus avec des reflets), mais plus nos attentes sont hautes, plus la complexité de nos méthodes est élevée et plus la qualité de la labellisation devient prépondérante. C'est donc quelque chose à prendre en compte dans notre travail, afin de savoir jusqu'où nous voulons pousser notre chaîne de traitement automatique. Dans les faits, notre travail doit rester guidé par l'idée d'assister (voire de remplacer) une tâche humaine. Les outils devront être alors suffisamment autonomes tout en produisant une prédiction à la hauteur d'une analyse de technicien.

Ce projet a été particulièrement stimulant car nous avons été libres dans l'approche que nous allions employer. S'il n'y avait pas a priori une unique approche valide, nous devions être capables de bien cerner les limites de notre travail afin de prendre en compte la variable temporelle. Notre projet a une échéance fixée et a pour objectif de produire des outils fonctionnels dont les fondamentaux ont été déjà posés en première partie de projet. Ainsi, la qualité du code rendu a été un enjeu central, tout comme le développement d'une utilisation conjointe des deux approches pour fournir une solution complète en fin de projet. La quantité de données labellisées a été un facteur limitant notre avancé. Nous avons pu finalement tirer parti de l'ensemble des travaux de recherche et des outils développés pour obtenir une solution aboutie et applicable par le client. Toutes ces contraintes ont demandé donc une capacité à se fixer des limites et à organiser le travail de groupe dans la durée afin de respecter les échéances tout en fournissant un travail à la hauteur de nos ambitions.

## **Annexes**

# Annexe 1 : Matrices de confusion et mesure de précision

Les matrices de confusion sont un outil qui mesure la capacité d'un algorithme de Deep Learning à prédire un résultat. Elles sont adaptées aux problèmes de classification : elles permettent de mettre en valeur les prédictions correctes et incorrectes, et donnent également un indice sur le type d'erreurs commises.

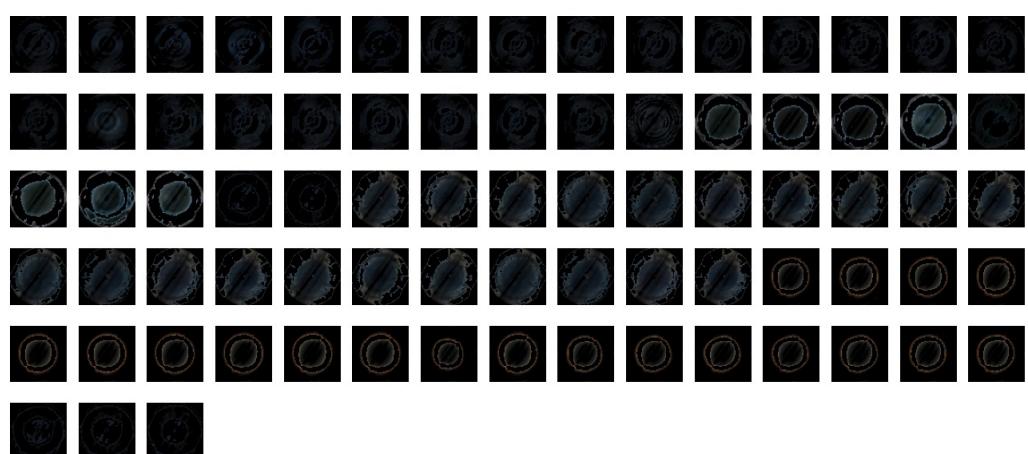
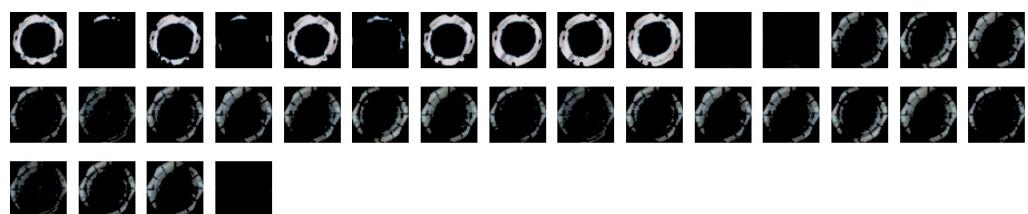
On classe dans un tableau (la matrice de confusion) les résultats obtenus en faisant tourner l'algorithme sur un ensemble de données de test. Chaque colonne du tableau contient la classe prédite par l'algorithme, et les lignes les classes réelles. Ainsi, on retrouve toujours les bonnes prédictions sur la diagonale. Les cases en dehors de la diagonale permettent d'identifier les classes qui sont souvent confondues par l'algorithme.

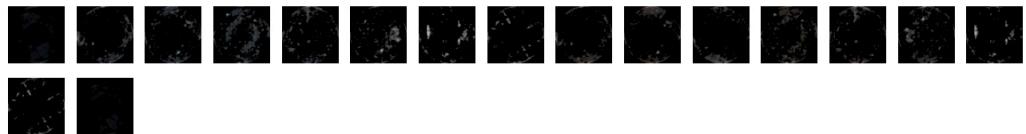
Une matrice de confusion dans le cas d'un problème à deux classes est donné en exemple sur la figure 4.1.

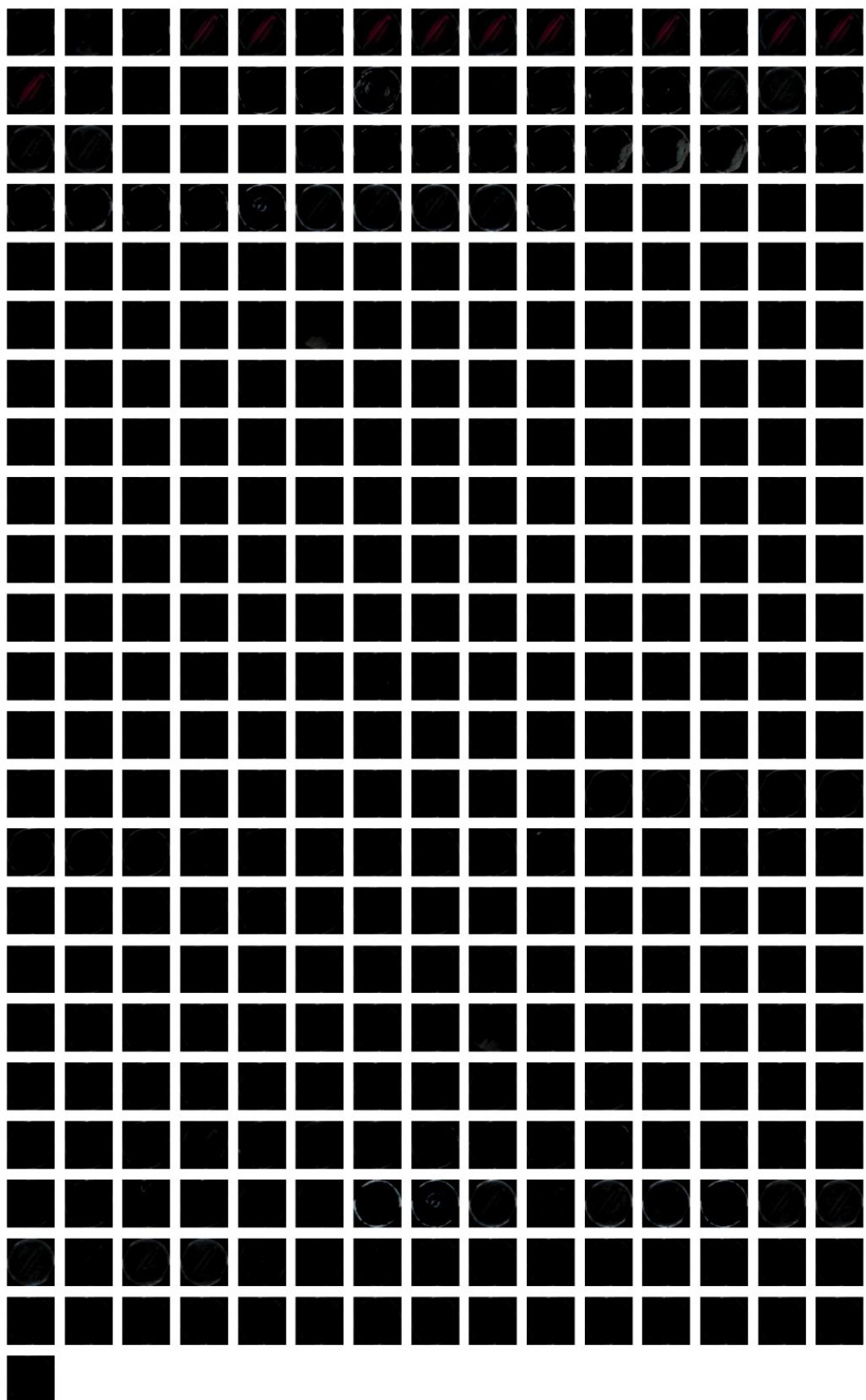
		Réponse de l'expert	
		<b>p</b>	<b>n</b>
Réponse du classifieur	Y	Vrai Positif	Faux Positif
	N	Faux Négatif	Vrai Négatif

FIGURE 4.1 – Exemple de matrice de confusion

## Annexe 2 : Exemples de classes obtenues par l'approche non supervisée







# Bibliographie

- [1] Bartomeu Coll and Jean-Michel Morel. Non-local means denoising. *Image Processing On Line*, 1, 09 2011.
- [2] Alexander Kirillov et al. Segment anything. *Meta AI*, 2023.
- [3] Wouter Van Gansbeke, Simon Vandenhende, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Scan : Learning to classify images without labels, 2020.
- [4] Ross Girshick. Fast r-cnn, 2015.
- [5] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.
- [6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn, 2018.
- [7] Shashi Bhushan Jha and Radu F. Babiceanu. Deep cnn-based visual defect detection : Survey of current literature. *Computers in Industry*, 148 :103911, 2023.
- [8] C. Leverger. Détection des contours d'une image : le filtre de canny. *ENSSAT Lannion*, 2016.
- [9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn : Towards real-time object detection with region proposal networks, 2016.
- [10] N.G. Shankar and Z.W. Zhong. Defect detection on semiconductor wafer surfaces. *Microelectronic Engineering*, 77(3) :337–346, 2005.
- [11] Paolo Favaro Simon Jenni. Self-supervised feature learning by learning to spot artifacts.
- [12] Wikipedia contributors. Generative adversarial network — Wikipedia, the free encyclopedia, 2024. [Online ; accessed 29-May-2024].
- [13] Wikipédia. Détection de contours — wikipédia, l'encyclopédie libre, 2023. [En ligne ; Page disponible le 20-juin-2023].
- [14] Wikipédia. Méthode d'otsu — wikipédia, l'encyclopédie libre, 2023. [En ligne ; Page disponible le 20-novembre-2023].