

DÉTECTION D'ANOMALIES INTELLIGENTE AU COURS D'UNE IMPRESSION 3D

Rapport de Stage

soutenu le 1er Septembre 2022

pour l'obtention du

Master 2 Intelligence Artificielle de l'Université d'Artois

par

Duc Viet NGUYEN

Encadrant entreprise : Cédric Bobenrieth

Encadrant universitaire : Salem Benferhat

Remerciements

Tout d'abord, je tiens à remercier très chaleureusement Monsieur Cédric BOBEN-RIETH - mon encadrant durant toute la durée de ce stage. Je tiens particulièrement à le remercier d'avoir proposé ce sujet de stage, pour son soutien, sa disponibilité, sa persévérance sans faille, et ses précieux conseils qui m'ont permis de progresser et de rédiger ce rapport.

Je tiens également à remercier Monsieur Salem BENFERHAT - mon responsable universitaire pour sa gentillesse, ses conseils et son soutien tout au long de l'année scolaire et du stage. Je tiens également à remercier les membres du Jury qui ont accepté l'évaluation et pris le temps d'examiner ce projet.

Je tiens aussi à remercier les enseignants du parcours Informatique de l'Université d'Artois pour leur formation de qualité, ainsi que les connaissances apportées qui m'ont permis d'effectuer ce stage dans de bonnes conditions.

Je tiens à remercier Madame Hyewon SEO - la responsable de l'équipe MLMS Strasbourg de m'avoir fourni le matériel nécessaire, Killian VUILLEMOT - mes collègues, Theo GAYANT - mon ami, qui m'ont soutenu dans la préparation de ce rapport de stage.

Enfin, à toutes les personnes, famille, amis, connaissances, qui m'auront soutenu, de près ou de loin, dans l'élaboration de ce mémoire. Je tiens à leur exprimer toute ma gratitude et ma reconnaissance.

Table des matières

Table des figures	vi
Résumé	1
Chapitre 1 Introduction	2
1.1 Introduction générale	2
1.2 Organisme d'accueil	3
1.3 Présentation du problème	4
1.3.1 Imprimante 3D	5
1.3.2 Anomalies	5
1.4 Conclusion	6
Chapitre 2 L'approche proposée	8
2.1 Introduction	8
2.2 Description	8
2.2.1 Générateur	8
2.2.2 Traducteur	9
2.2.3 Comparateur	9
2.2.4 Détecteur	9
2.3 Étapes de mise en œuvre	10
2.4 Intuition	10
Chapitre 3 État de l'art	11
3.1 Introduction	12
3.2 Réseaux de neurones convolutifs (Traducteur)	12
3.2.1 Introduction	12
3.2.2 Couche de convolution	13
3.2.3 Stride	14

3.2.4	Rembourrage (Padding)	14
3.2.5	Unité linéaire rectifiée - ReLU	15
3.2.6	Couche de Pooling	15
3.2.7	Couche entièrement connectée	16
3.2.8	Conclusion	17
3.3	Réseau de neurones Siamois (Comparateur)	17
3.3.1	Introduction	17
3.3.2	Architecture	19
3.3.3	Principales caractéristiques du réseau siamois	20
3.3.4	Entraînement au réseau de neurones siamois	20
3.3.5	Avantages et inconvénients	21
3.3.6	Fonctions de perte	22
3.3.7	Conclusion	24
3.4	Yolov4 (DéTECTEUR)	24
3.4.1	Introduction	24
3.4.2	La backbone CSPDarknet53	25
3.4.3	Cou (agrégation de feature)	26
3.4.3.1	SPP	26
3.4.3.2	PAN	27
3.4.3.3	SAM : Modèle d'Attention Spatiale	28
3.4.4	Tête	29
3.4.5	Fonction de perte	30
3.4.5.1	Perte IoU	31
3.4.5.2	Perte CIoU	31
3.4.6	Activation Mish	32
3.4.7	Conclusion	32
3.5	Générateur	32
3.5.1	Introduction	32
3.5.2	Étalonnage	33
3.5.3	Simulateur GCode	33
3.6	Conclusion	33
Chapitre 4 Base de données		35
4.1	Introduction	35
4.2	Distribution et étiquetage des classes	37

4.2.1	Classes	37
4.2.2	Étiquetage	38
4.2.3	Distribution	39
Chapitre 5 Expérimentations et résultats		42
5.1	Introduction	42
5.2	Outils utilisés	43
5.3	Générateur	44
5.3.1	Étalonnage	44
5.3.2	Simulation	44
5.4	Comparateur	45
5.4.1	Introduction	45
5.4.2	Résultat	47
5.4.3	Méthode alternative.	48
5.5	Détecteur	49
5.5.1	Data Augmentation	49
5.5.1.1	CutMix	49
5.5.1.2	Augmentation des données en Mosaic	50
5.5.2	Méthode de régularisation	50
5.5.2.1	Introduction	50
5.5.2.2	DropBlock	50
5.5.2.3	Lissage des étiquettes de classe	51
5.5.3	Entraînement	51
5.5.3.1	Évaluation	51
5.5.4	Expériences	52
5.5.4.1	Introduction	52
5.5.4.2	Réglages et Hyperparamètres	53
5.5.4.3	Résultats	54
5.5.4.4	Précision du modèle sous Cinq Angles	61
5.6	Problèmes rencontrés	63
5.7	Conclusion	64
Conclusion		65
Bibliographie		67

Annexe A Programmes Python	71
Annexe B Capture d'écran	84

Table des figures

1.1	Photos des équipes MLMS du laboratoire ICube.	3
1.2	Les équipes de recherche du laboratoire.	4
1.3	Imprimante 3D avec 5 caméras.	5
1.4	Modèle d'imprimante 3D.	6
1.5	Anomalies qui peuvent être détectées sans connaître le modèle 3D.	6
1.6	le décalage	7
2.1	Architecture hybride.	9
3.1	Une image RVB de taille 6x6x3 (3 correspond ici aux valeurs RVB) [1].	12
3.2	Ensemble du flux CNN pour traiter l'image d'entrée [2].	13
3.3	Matrice d'image multiplie par le noyau ou la matrice de filtre [3].	13
3.4	Matrice d'image multiplie le noyau [3].	14
3.5	3 x 3 matrice de sortie [3].	14
3.6	Quelques filtres courants [3].	15
3.7	Stride de 2 pixels [3].	16
3.8	Fonctionnement ReLU [3].	16
3.9	Types de Pooling [3].	17
3.10	Aplation de la matrice en vecteur et l'avons livré dans une couche entièrement connectée comme un réseau de neurones [3].	18
3.11	Modèle de réseau de neurones siamois [4].	19
3.12	la sortie est la distance entre 2 images.	21
3.13	Structure du modèle de perte contrastive [5].	23
3.14	Structure du modèle de Perte de triplet [5].	23
3.15	La perte triplet minimise la distance entre une ancre et un positif, qui ont tous deux la même identité, et maximise la distance entre l'ancre et un négatif d'une identité différente [6].	24
3.16	Architecture du détecteur à une étape Yolov4 [7].	25
3.17	DenseNet [8].	25
3.18	Bloc DenseNet et bloc CSPDenseNet [9].	26
3.19	SSP d'origine et SPP modifié [10].	27
3.20	PANet [7].	28

3.21 Module d'Attention Spatiale Original [11].	28
3.22 SAM modifié [7].	29
3.23 Tête de Yolov3[12].	30
3.24 Fonction d'activation Mish [13].	32
4.1 Défauts générés.	36
4.2 Classes Fil, Extrudeuse et Objet.	38
4.3 Programme LabelImg [14].	39
4.4 Pourcentage de chaque classe.	40
4.5 Pourcentage de chaque anomalie	41
5.1 Calibrage de la caméra de Matlab.	44
5.2 Simulation d'impression d'un modèle 3D à partir de deux perspectives (avant, dessus).	45
5.3 L'espace simulateur correspond à l'espace réel.	46
5.4 Image est subdivisée en plusieurs patchs de 32x32 pixels.	46
5.5 Calibrage de la caméra de Matlab.	47
5.6 Encadrer les parties anormales du fichier image réel.	48
5.7 Aperçu des résultats de Mixup, Cutout et CutMix [15].	49
5.8 Mosaic représente une nouvelle méthode d'augmentation des données [7].	50
5.9 Les régions vertes dans (b) et (c) incluent les unités d'activation qui contiennent des informations sémantiques dans l'image d'entrée.	51
5.10 <i>mAP</i> 0.5 et fonction de perte de la première expérience.	55
5.11 <i>mAP</i> 0.5 et fonction de perte de la Deuxième expérience.	58
5.12 <i>mAP</i> 0.5 et fonction de perte de la Troisième expérience.	60
5.13 Application du modèle Yolov4 à la prédiction des classes d'objets anormaux.	62
B.1 La matrice de comparaison est générée après l'utilisation du programme A.2.	84
B.2 PhotoFiltre permet de recadrer et de comparer la position d'objet de 2 images.	85
B.3 Le diagramme de Gantt représente l'avancement des tâches du projet que nous avons effectuées.	85

Résumé

Aujourd’hui, les imprimantes 3D sont considérées comme une révolution dans la fabrication additive (FA), recevant beaucoup d’attention de la part des chercheurs et de l’industrie, notamment dans le domaine du contrôle de la qualité. Il est nécessaire qu’une imprimante 3D puisse fonctionner de manière autonome en permanence tout en assurant une production de pièces de bonnes qualités, c’est-à-dire dépourvues de quelconques défauts d’impression. Ceci est garanti si nous créons une imprimante 3D intelligente capable de détecter les anomalies dans le processus d’impression, en temps réel. Cela évitera de perdre du temps et du matériel. Alors que les fabricants utilisent l’apprentissage automatique depuis des décennies, les récents modèles d’apprentissage en profondeur ont poussé le domaine du contrôle qualité vers un nouveau niveau.

Dans ce rapport, nous proposons une nouvelle approche basée sur l’état de l’art des architectures de réseaux de neurones convolutifs (CNN) prenant en compte des images de l’objet imprimé venant de tous les côtés (haut, avant, arrière, gauche et droite), ainsi que l’utilisation du réseau de neurones siamois pour créer une matrice de comparaison contenant des informations sur le modèle 3D de l’objet imprimé. Enfin, cette approche utilise le modèle Yolov4 avec en entrée la matrice de comparaison et l’image réelle de l’objet pour détecter les anomalies dans le processus d’impression.

1

Introduction

Sommaire

1.1	Introduction générale	2
1.2	Organisme d'accueil	3
1.3	Présentation du problème	4
1.3.1	Imprimante 3D	5
1.3.2	Anomalies	5
1.4	Conclusion	6

1.1 Introduction générale

Les technologies de fabrication additive (FA) ont progressé au cours des dernières décennies, et c'est grâce à la technologie de la numérisation que les formes peuvent être virtuellement conçues, imprimées sur place puis testées, le tout dans un laps de temps très court. L'objet 3D est imprimé couche par couche selon un fichier GCode qui contient toutes les informations relatives au processus d'impression, ces objets 3D sont fortement utilisés dans le domaine de la fabrication comme l'automobile ou l'aéronautique. La qualité et la fiabilité des pièces sont essentielles pour maintenir la compatibilité et l'intégrité structurelle nécessaires à la construction de constructions complexes. Par conséquent, il est essentiel de détecter toute anomalie pouvant survenir au cours d'un processus d'impression, susceptible de provoquer un décalage entre les conceptions 3D souhaitées et les formes imprimées. Pour atteindre cet objectif, un système de détection d'anomalies est nécessaire, basé sur le traitement d'images, capturées à partir de cinq caméras chacune dans un emplacement spécifique pour entourer l'imprimante 3D et des algorithmes d'apprentissage en profondeur, dans le but d'identifier et de distinguer les imperfections et les défauts d'un objet imprimé.

L'apprentissage en profondeur a prouvé à maintes reprises son efficacité dans de nombreux domaines, la surveillance de la qualité ne fait pas exception. En effet, avec le CNN

(Convolutional Neural Network), les chercheurs du monde entier ont été dotés d'un nouvel outil qui surpasse de loin les autres approches en termes de précision et d'efficacité.

Le reste de ce rapport est organisé comme suit :

1. Dans la suite de ce chapitre nous introduisons le contexte général de mon stage.
2. Le chapitre 2 décrit notre approche proposée.
3. Le chapitre 3 décrit les propositions d'état de l'art que nous avons mises en œuvre à notre approche jusqu'à présent.
4. Le chapitre 4 décrit l'ensemble de données.
5. Le chapitre 5 présente les résultats et les discussions de nos expériences.
6. Le dernier chapitre présente les conclusions et les travaux futurs.

Mes travaux dans ce stage comprend : rapport de stage, rapport de conférence, code source, les articles scientifiques seront mis à jour sur le lien [github](#).

1.2 Organisme d'accueil

Créé en 2013, le Laboratoire ICube rassemble l'ensemble des ressources de recherche strasbourgeoises dans les domaines de l'Ingénierie et des Sciences de l'Information, avec l'Image comme fil conducteur. De ce fait, une communauté d'intérêts similaires s'est constituée, permettant la mise en place de projets ambitieux dans de nombreux domaines : Informatique, Robotique, Biophysique, Microélectronique, Photonique, Génie mécanique, Apprentissage et Traitement d'Image. Avec plus de 500 personnes, c'est une force de recherche majeure à Strasbourg.

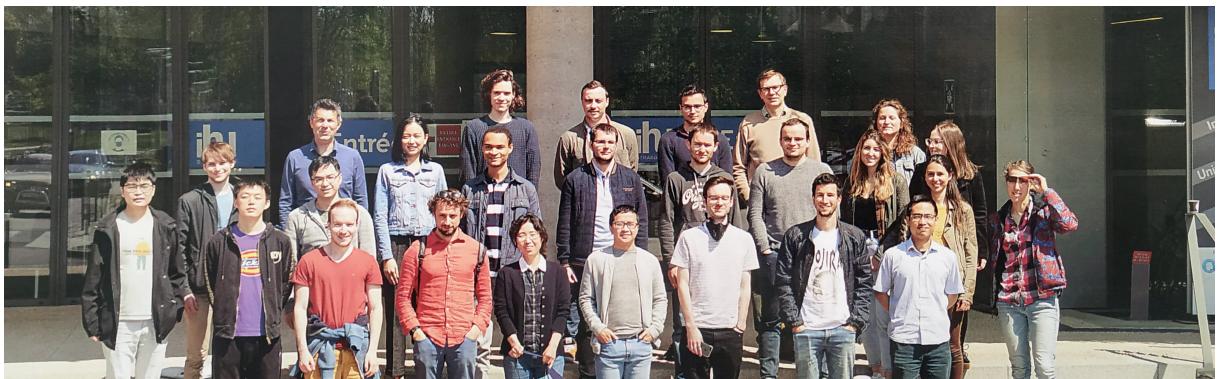


FIGURE 1.1 – Photos des équipes MLMS du laboratoire ICube.

ICube est issu de la fusion de quatre laboratoires de recherche affiliés au CNRS (Centre National de la Recherche Scientifique) : LSIIT UMR 7005, spécialisée en Informatique, Traitement d'Images, Robotique et Télédétection, InESS UMR 7163, spécialisée en Microélectronique et Photonique, IMFS FRE 3240, spécialisée en Biomécanique et Mécanique

des Fluides Réactifs, et l'équipe Imagerie In Vivo du LINC UMR 7237, spécialisée en Biophysique. Le laboratoire compte près de 230 salariés permanents dont environ 200 chercheurs, enseignants-chercheurs et ingénieurs de recherche. Les principaux domaines d'application du laboratoire ICube sont l'ingénierie médicale et la durabilité.

ICube dépend de plusieurs institutions différentes : l'Université de Strasbourg, le CNRS (INSIS et INS2I), l'ENGEES (École Nationale du Génie de l'Eau et de l'Environnement de Strasbourg) et l'INSA de Strasbourg. Le Laboratoire ICube, réparti dans 3 antennes, comprend quatre départements, et chaque département regroupe plusieurs équipes de recherche (voir figure 1.2 ci-dessous).

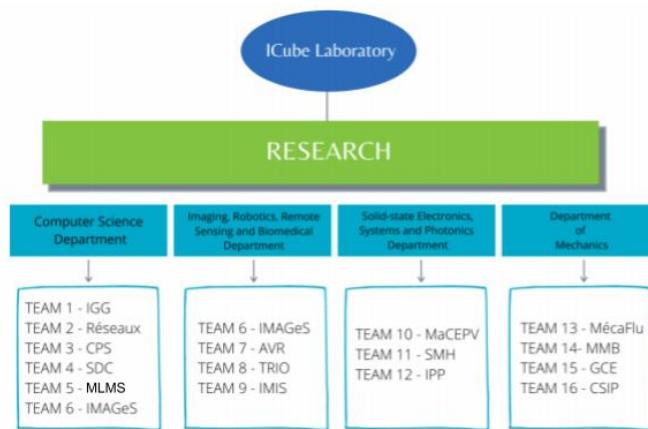


FIGURE 1.2 – Les équipes de recherche du laboratoire.

1.3 Présentation du problème

Tout au long de l'année, étudiants, professeurs et chercheurs impriment une variété d'objets 3D pour leurs projets professionnels et académiques. Cela fait que l'imprimante 3D fonctionne sans arrêt, en particulier pendant les heures de pointe ou vers la fin du semestre, car certains objets 3D ont besoin de plus de 24 heures pour s'imprimer. Malheureusement, de nombreux défauts surviennent, et sans surveillance, l'impression continue d'imprimer, gaspillant beaucoup de matériel et de temps, pour finalement devoir recommencer le lendemain.

Mon sujet de stage est de réaliser un modèle de réseau de neurones profond capable de détecter des défauts en temps réel, afin d'arrêter l'impression et d'avertir le superviseur dès qu'un problème est identifié. Pour cette raison, nous avons installé 5 caméras simples **forcasm** (faible budget) autour de l'imprimante (voir figure 1.3), connectées au serveur de l'école, nous permettant de recevoir des données en temps réel. La raison pour laquelle nous avons choisi cinq caméras est que nous craignions qu'une ou deux caméras ne puissent

pas capturer tous les côtés de l'objet 3D. La figure 1.3 nous donne une vue complète de l'objet (Haut, Avant, Arrière, Gauche, Droite).

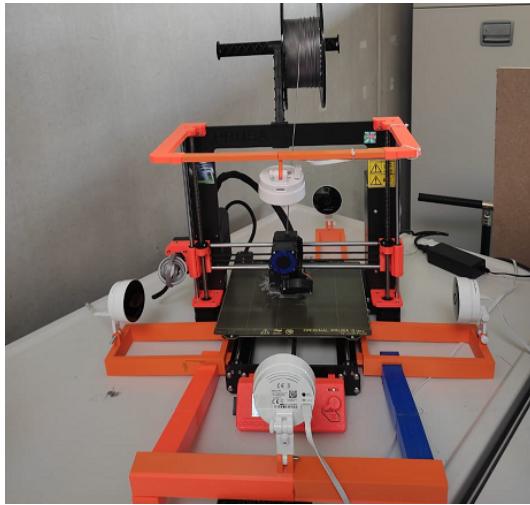


FIGURE 1.3 – Imprimante 3D avec 5 caméras.

1.3.1 Imprimante 3D

Le sujet de cette étude est la célèbre imprimante **Prusa i3 mk3** (voir figure 1.4), qui est considérée comme l'une des meilleures imprimantes 3D pour un usage quotidien. Il y a trois choses importantes que nous devons savoir pour comprendre le processus d'impression :

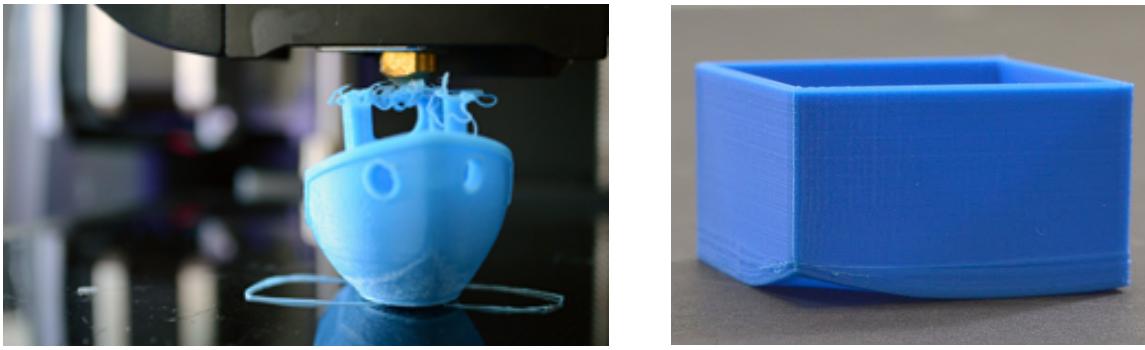
- **Lit** : Où l'objet 3D sera imprimé. Il se déplace le long de l'axe (O, Y) (voir figure 1.4).
- **Extrudeuse** : pièce permettant de faire couler le filament et ainsi d'imprimer l'objet 3D. Elle se déplace le long du plan (O, X, Z) (voir figure 1.4).
- **Fichier GCode** : C'est un fichier que nous connectons à l'imprimante pour imprimer ce que nous voulons. Il contient les coordonnées 3D (nuage 3D) des objets 3D ainsi que d'autres informations liées au processus d'impression telles que la température de l'extrudeuse, la température du lit, la vitesse, la quantité de fil à couler entre chaque point, etc.

1.3.2 Anomalies

Autant d'anomalies qu'il y en a, chacune appartient à l'une des deux familles d'anomalies suivantes : celles pour lesquelles on peut détecter sans connaissance préalable du modèle 3D et celles pour lesquelles on ne peut décider s'il s'agit d'une anomalie (ou non) sans avoir une connaissance préalable du modèle 3D. La première famille est indépendante de la forme et de la taille de l'objet 3D, par exemple *spaghetti* ou *warping* (voir figures 1.5a, 1.5b).



FIGURE 1.4 – Modèle d'imprimante 3D.



(a) Spaghetti.

(b) Warping.

FIGURE 1.5 – Anomalies qui peuvent être détectées sans connaître le modèle 3D.

Les anomalies de la seconde famille dépendent de la forme de la taille et de la texture de l'objet. Par exemple si l'imprimante a commencé à imprimer l'objet 3D, mais avec une taille plus petite ou si par erreur on branche un fichier GCode différent qui imprimera autre chose. Un autre exemple est les défauts de décalage (voir figure 1.6), qui en le regardant avec les yeux, on ne peut pas savoir si le modèle 3D a ce inteface, ou que l'imprimante 3D qui a fait l'erreur.

1.4 Conclusion

Pour résumer ce qui a été discuté dans ce chapitre, notre objectif est de construire un système intelligent utilisant des réseaux de neurones profonds, capable de détecter toutes sortes de défauts pouvant survenir dans le processus d'impression. Notre système doit s'adapter à tout type d'imprimante 3D (lit mobile ou fixe) et il doit également être adapté à la détection en temps réel (pas de pause pendant l'impression).

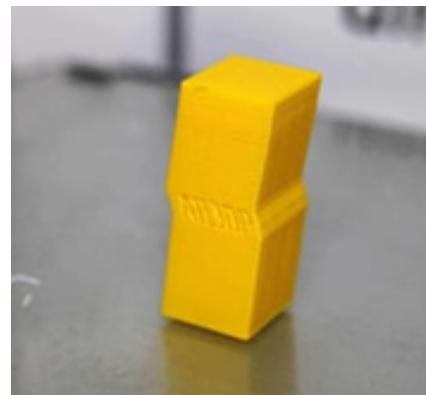


FIGURE 1.6 – le décalage

2

L'approche proposée

Sommaire

2.1	Introduction	8
2.2	Description	8
2.2.1	Générateur	8
2.2.2	Traducteur	9
2.2.3	Comparateur	9
2.2.4	Détecteur	9
2.3	Étapes de mise en œuvre	10
2.4	Intuition	10

2.1 Introduction

Après avoir bien compris le problème ainsi que les dernières approches états de l'art. Nous avons essayé d'utiliser toutes ces informations pour trouver une solution afin de résoudre notre problème. Cela nécessite que le modèle prenne en compte l'objet 3D en cours d'impression ainsi que les informations sur le modèle 3D (fichier GCode). Sur la base de ce principe, nous proposons une approche hybride (voir figure 2.1), qui examinera le modèle 3D de l'objet ainsi que cinq images pour estimer la qualité d'impression. Notre approche est divisée en quatre parties : **Détecteur**, **Traducteur**, **Générateur** et **Comparateur**.

2.2 Description

2.2.1 Générateur

Le générateur est un simulateur d'imprimante 3D, ce composant prend l'angle de vue de la caméra (paramètres externes et internes), accompagnant l'imprimante d'un fichier GCode permettant de modéliser en 3D l'objet couche par couche.

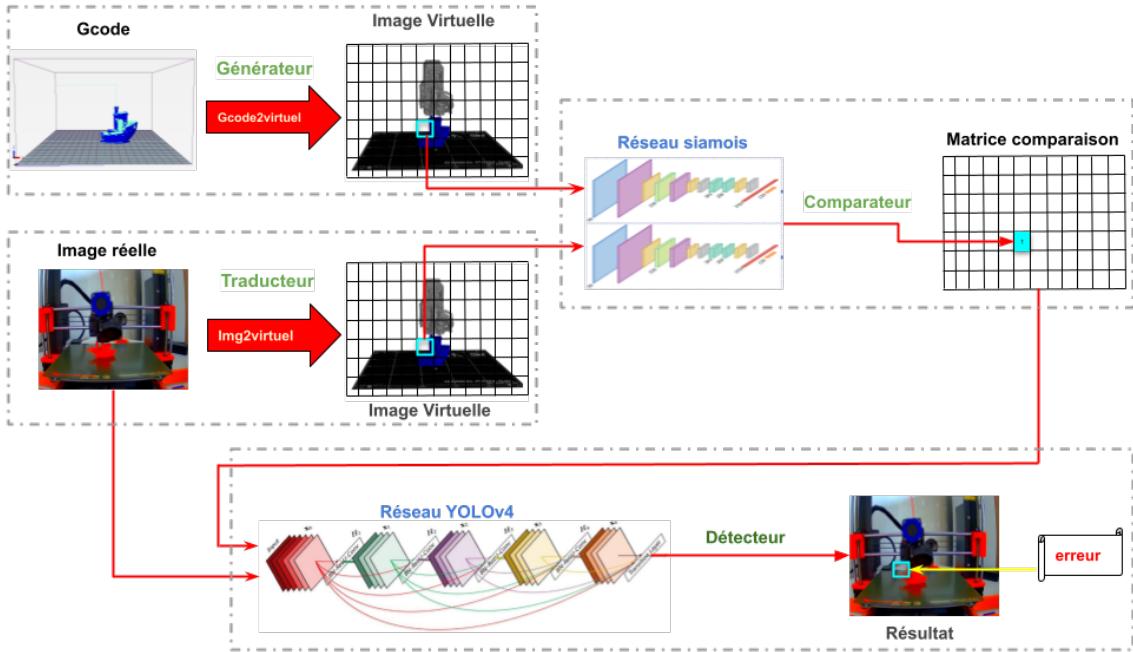


FIGURE 2.1 – Architecture hybride.

2.2.2 Traducteur

Le traducteur est également un réseau convolutif [16] dont la fonction est de prendre une image d’entrée de l’espace du monde réel (cinq caméras) et de la traduire dans l’espace du simulateur (espace du générateur). Pour ce modèle, nous avons cherché à utiliser une traduction image à image non appariée inspirée des modèles génératifs, basée soit sur la famille de réseaux antagonistes génératifs **GAN** [17], soit sur la famille d’encodeurs automatiques **AU** [18].

2.2.3 Comparateur

Le comparateur est également un réseau convolutif [16] qui prend deux images en entrée puis sort la différence entre elles en utilisant des hyperparamètres.

2.2.4 DéTECTEUR

Le détecteur est un réseau neuronal convolutif [16] dont l’objectif principal est de détecter toutes sortes d’anomalies. Ce réseau sera basé sur un modèle de détection à une seule étape, considérant que nous voulons que notre approche soit adaptée à la détection en temps réel. De plus, le réseau de détecteurs sera conçu de manière à prendre deux images en entrée.

2.3 Étapes de mise en œuvre

Notre approche est la suivante : premièrement, l'image passe par un modèle de traduction pré-entraîné, celui-ci transforme les images de l'espace du monde réel à l'espace du simulateur (espace modèle 3D). Le générateur, d'autre part, a déjà traité le fichier GCode de l'objet imprimé actuel et rendu les images du modèle 3D avec leur angle de vue respectif dans le temps correspondant (voir figure 2.1). Ensuite, chaque image avec sa traduction correspondante passe par le comparateur, qui produit une matrice de comparaison. L'image d'entrée ainsi que la matrice de comparaison seront transmises à notre détecteur personnalisé.

2.4 Intuition

Le modèle de traducteur sera entraîné uniquement sur de bonnes données (bonnes impressions), de sorte qu'il apprend à capturer la taille et la forme de l'objet dans son ensemble et non sur des anomalies. En d'autres termes, nous voulons avoir une mauvaise transformation, un "Shift" par exemple, dans les régions qui contiennent des anomalies (le modèle d'indice n'est entraîné que sur de bonnes données).

La matrice de comparaison indique la corrélation entre la vérité terrain (image générée à partir du fichier GCode) et l'image traduite du monde réel. Si les deux images sont similaires, l'objet imprimé correspond au modèle 3D, sinon il y a un risque élevé de défaut. Le détecteur prendra la matrice de comparaison avec l'image pour identifier les anomalies, ce qui donnera des informations sur des anomalies indépendantes (première famille).

3

État de l'art

Sommaire

3.1	Introduction	12
3.2	Réseaux de neurones convolutifs (Traducteur)	12
3.2.1	Introduction	12
3.2.2	Couche de convolution	13
3.2.3	Stride	14
3.2.4	Rembourrage (Padding)	14
3.2.5	Unité linéaire rectifiée - ReLU	15
3.2.6	Couche de Pooling	15
3.2.7	Couche entièrement connectée	16
3.2.8	Conclusion	17
3.3	Réseau de neurones Siamois (Comparateur)	17
3.3.1	Introduction	17
3.3.2	Architecture	19
3.3.3	Principales caractéristiques du réseau siamois	20
3.3.4	Entraînement au réseau de neurones siamois	20
3.3.5	Avantages et inconvénients	21
3.3.6	Fonctions de perte	22
3.3.7	Conclusion	24
3.4	Yolov4 (DéTECTEUR)	24
3.4.1	Introduction	24
3.4.2	La backbone CSPDarknet53	25
3.4.3	Cou (agrégation de feature)	26
3.4.4	Tête	29
3.4.5	Fonction de perte	30
3.4.6	Activation Mish	32
3.4.7	Conclusion	32

3.5 Générateur	32
3.5.1 Introduction	32
3.5.2 Étalonnage	33
3.5.3 Simulateur GCode	33
3.6 Conclusion	33

3.1 Introduction

Ce projet est complexe pour trouver des solutions. En outre, cela demande également beaucoup d'efforts et de temps pour la collecte de données. Mon rôle dans ce stage est de rechercher, d'analyser l'état de l'art et d'établir une approche solide que mon superviseur continuera de mettre en oeuvre. Jusqu'à présent, nous avons implémenté avec succès le **générateur** et le **traducteur**, et avons découvert le réseau *siamois* pour le **comparateur** et le réseau *Yolov4* pour le **détecteur**.

Par conséquent, ce chapitre est consacré à la description des méthodes utilisées pour ces sections.

3.2 Réseaux de neurones convolutifs (Traducteur)

3.2.1 Introduction

Dans le réseau de neurones [19], le modèle Convolutional Neural Network (CNN) [16] est l'un des modèles de reconnaissance et de classification d'images. En particulier, l'identification d'objets et la reconnaissance faciale sont des domaines dans lesquels CNN est largement utilisé.

CNN catégorise une image en prenant une image d'entrée, en la traitant et en la catégorisant dans certaines catégories (par exemple Chien, Chat, Tigre, etc). L'ordinateur considère l'image d'entrée comme un tableau de pixels et cela dépend de la résolution de l'image. Selon la résolution de l'image, l'ordinateur doit afficher $h \times l \times p$ (h : hauteur, l : largeur, p : profondeur).

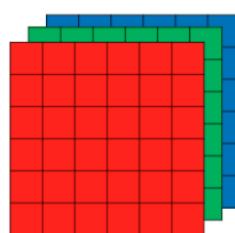


FIGURE 3.1 – Une image RVB de taille 6x6x3 (3 correspond ici aux valeurs RVB) [1].

Techniquement, lors de l'entraînement et des tests à l'aide de modèles CNN, chaque image d'entrée passera par une série de couches de convolution avec des filtres (noyaux) [16], agrégeant les couches entièrement connectées [16]. La fonction *Softmax* [19] est ensuite appliquée pour classer les objets avec une valeur de probabilité comprise entre 0 et 1. La figure 3.2 montre l'ensemble du flux CNN pour traiter l'image d'entrée et classer les objets en fonction de la valeur.

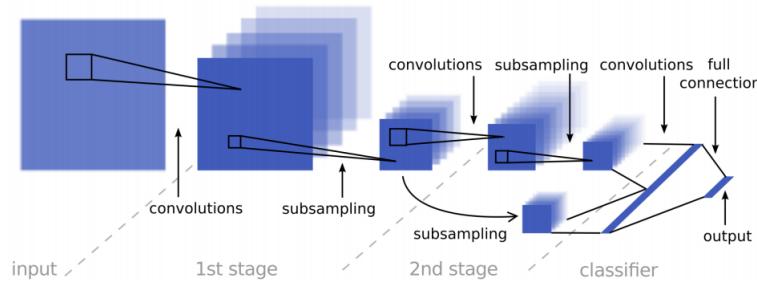


FIGURE 3.2 – Ensemble du flux CNN pour traiter l'image d'entrée [2].

3.2.2 Couche de convolution

Convolution [3] est la première couche à extraire des entités de l'image d'entrée. La convolution maintient les relations entre les pixels en explorant les caractéristiques de l'image à l'aide de petits carrés des données d'entrée. C'est une opération mathématique qui a deux entrées telles que la matrice d'image et un filtre ou noyau.



FIGURE 3.3 – Matrice d'image multiplie par le noyau ou la matrice de filtre [3].

L'image 3.3 comprend :

- Une matrice d'image de dimension $(h \times l \times p)$.
- Un filtre $(f_h \times f_w \times d)$.
- Une sortie du volume de dimension $(h - f_h + 1) \times (w - f_w + 1) \times 1$.

Considérons une matrice 5×5 avec des valeurs de pixel de 0 et 1. La matrice de filtre 3×3 est illustrée sur la figure 3.4.

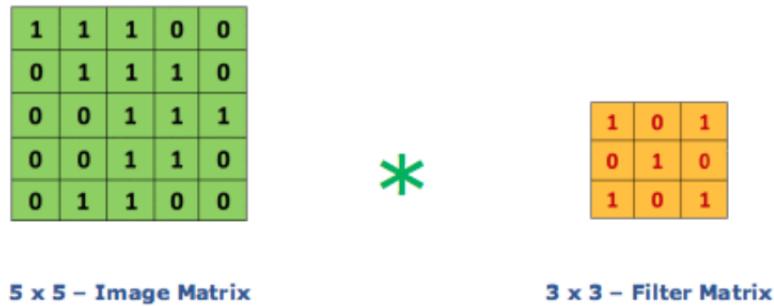
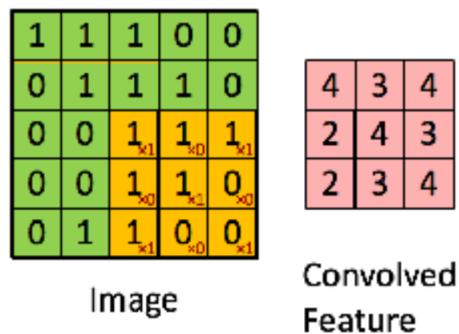


FIGURE 3.4 – Matrice d'image multiplie le noyau [3].

Ensuite, la couche convolutionnelle de la matrice d'image 5×5 multipliée par la matrice de filtre 3×3 appelée 'Feature Map' comme indiqué sur la figure 3.5.

La combinaison d'une image avec différents filtres peut effectuer des opérations telles

FIGURE 3.5 – 3×3 matrice de sortie [3].

que la détection des contours, le flou et la netteté. La figure 3.6 montre une image de convolution différente après l'application de différents noyaux.

3.2.3 Stride

Le stride est le nombre de pixels qui changent sur la matrice d'entrée. Lorsque le stride est de 1, nous déplaçons les filtres de 1 pixel à la fois. Lorsque le stride est de 2, nous déplaçons les filtres de 2 pixels à la fois et ainsi de suite. La figure 3.7 montre que la convolution fonctionnera avec un stride de 2.

3.2.4 Rembourrage (Padding)

Parfois, le filtre (noyau) ne correspond pas complètement à l'image d'entrée. Nous avons deux options :

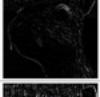
	Operation	Filter	Convolved Image
Identity		$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection		$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
		$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
		$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen		$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)		$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)		$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

FIGURE 3.6 – Quelques filtres courants [3].

- Ajouter des zéros aux quatre bordures de l'image (padding).
- Supprimer la partie de l'image qui ne correspond pas au filtre. C'est ce qu'on appelle un rembourrage valide qui ne conserve qu'une partie valide de l'image.

3.2.5 Unité linéaire rectifiée - ReLU

ReLU[20] est une fonction non linéaire donnée par :

$$f(x) = \max(0, x).$$

Pourquoi ReLU est importante : ReLU introduit la non-linéarité (voir figure 3.8) dans ConvNet [3]. Parce que les données du monde que nous étudions sont des valeurs linéaires non négatives.

Il existe d'autres fonctions non linéaires telles que tanh [21] ou *sigmoid* [21] qui peuvent également être utilisées comme alternatives à ReLU. La plupart des data scientists utilisent ReLU car ses performances sont meilleures que les deux autres.

3.2.6 Couche de Pooling

Le pooling réduira le nombre de paramètres lorsque l'image est trop grande. L'espace pooling réduit la taille de chaque map tout en conservant des informations importantes.

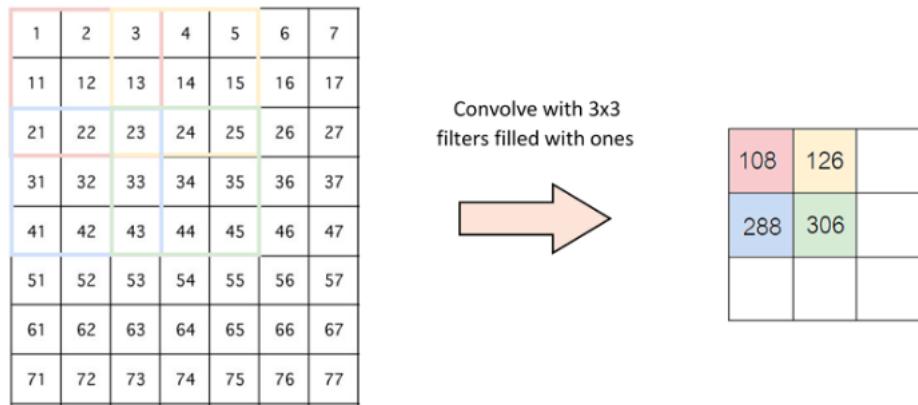


FIGURE 3.7 – Stride de 2 pixels [3].

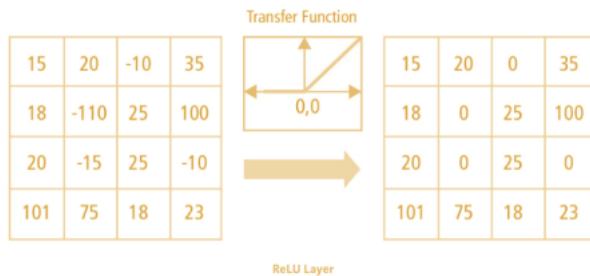


FIGURE 3.8 – Fonctionnement ReLU [3].

Les poolings peuvent être de différents types :

- Max Pooling (voir figure 3.9).
- Average Pooling (voir figure 3.9).
- Sum Pooling.

Max pooling obtient le plus grand élément de la matrice d'objets. La somme de tous les éléments de la map est appelée *sum pooling*.

3.2.7 Couche entièrement connectée

L'utilisation d'un réseau de neurones entièrement connecté est le moyen le plus courant d'apprendre des combinaisons non linéaires à partir de caractéristiques extraites des résultats de la matrice convective de sortie. Des réseaux de neurones entièrement connectés peuvent apprendre des fonctionnalités dans cet espace non linéaire.

Dans le diagramme 3.10, la matrice de la carte des caractéristiques sera convertie en vecteur (x_1, x_2, x_3, \dots) . Avec les couches entièrement connectées, nous avons combiné ces fonctionnalités pour créer un modèle. Enfin, nous avons une fonction d'activation telle que *softmax* [21] ou *sigmoïde* [21] pour classer les sorties comme chat, chien, voiture, camion, etc.

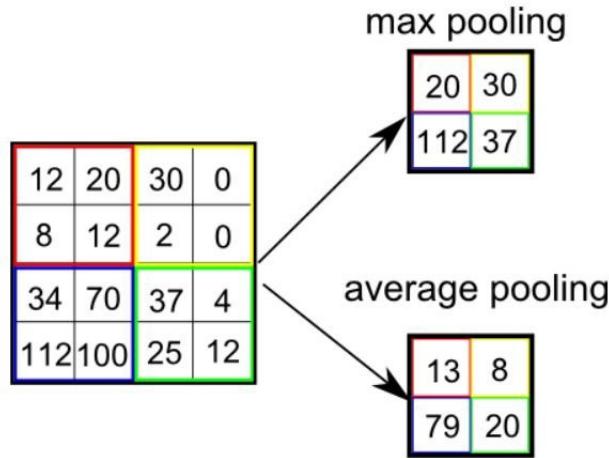


FIGURE 3.9 – Types de Pooling [3].

3.2.8 Conclusion

L'utilisation du modèle CNN pour l'application se fait selon les étapes suivantes :

- Fournir une image d'entrée dans la couche de convolution.
- Choisir des paramètres, appliquer des filtres avec des strides, rembourrage si nécessaire. Effectuer une convolution sur l'image et appliquer l'activation ReLU [20] à la matrice.
- Effectuer un pooling pour réduire la taille de la dimensionnalité.
- Ajouter autant de couches convolutives jusqu'à satisfaction.
- Aplatir la sortie et alimenter dans une couche entièrement connectée (couche FC).
- Sortie de la classe à l'aide d'une fonction d'activation (régression logistique avec fonctions de coût) et classe les images.

3.3 Réseau de neurones Siamois (Comparateur)

3.3.1 Introduction

La comparaison des correctifs entre les images est l'une des tâches les plus élémentaires de la vision par ordinateur et de l'analyse d'images. Elle est souvent utilisée comme un sous-programme qui joue un rôle important dans de nombreuses tâches visuelles.

Bien sûr, le problème de décider si deux images sont compatibles est assez difficile, car il y a beaucoup de facteurs qui affectent l'apparence finale d'une image [22]. Ceux-ci peuvent inclure des changements de perspective, des différences dans l'éclairage général d'une scène, une occlusion, un ombragé, des différences dans les réglages de l'appareil photo, etc. En fait, ce besoin de comparer les correctifs a donné lieu au développement de

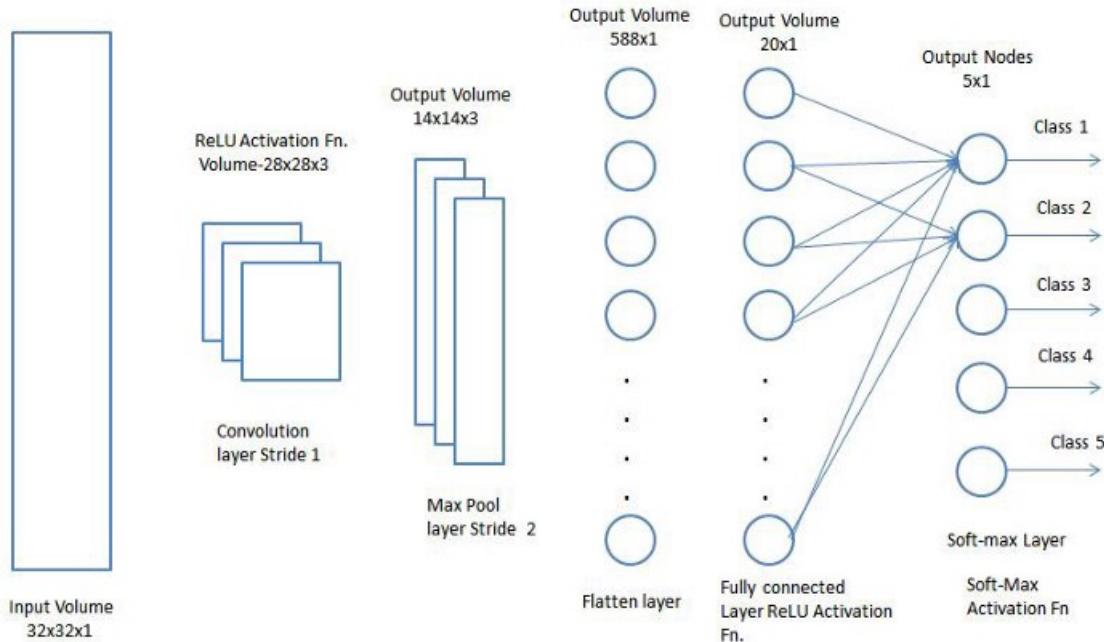


FIGURE 3.10 – Aplation de la matrice en vecteur et l'avons livré dans une couche entièrement connectée comme un réseau de neurones [3].

nombreux descripteurs de fonctionnalités conçus à la main au cours des dernières années, y compris *SIFT* [23], qui ont eu un impact énorme dans la communauté de la vision par ordinateur. Or, de tels descripteurs conçus manuellement peuvent ne pas être en mesure de prendre en compte de manière optimale tous les facteurs précités qui déterminent l'apparence d'un patch. D'autre part, de nos jours, on peut facilement accéder à de grands ensembles de données contenant des correspondances de patchs entre les images.

Notre objectif est donc d'utiliser une fonction de similarité générale pour les patchs d'image. Pour coder une telle fonction, nous utilisons et explorons ici des architectures de réseaux de neurones convolutifs *Siamois* (SNN) [24]. SNN n'utilise qu'un très petit nombre d'images pour obtenir de bien meilleures prédictions que les réseaux de neurones traditionnels. Il est donc également connu sous des noms tels que One-shot Learning, Few-shot Learning, etc. La capacité d'apprendre à partir de très peu de données a rendu le SNN plus populaire ces dernières années.

Nous prévoyons de décomposer l'image réelle et l'image GCode correspondante en petites paires de patchs. Nous comparons ensuite ces mêmes paires de patchs entre eux via le réseau de neurones SNN. La sortie de ces paires de patchs créera une matrice contenant les résultats de comparaison des paires de patchs que nous appelons matrice de comparaison.

3.3.2 Architecture

Un réseau de neurones *Siamois* (SNN) [24] (parfois appelé réseau de neurones jumeaux) est un réseau de neurones artificiel qui contient deux ou plusieurs sous-réseaux identiques, ce qui signifie qu'ils ont la même configuration avec les mêmes paramètres et poids. Les mises à jour des paramètres sont répercutées sur les deux sous-réseaux. Il est utilisé pour trouver la similarité des entrées en comparant ses vecteurs de caractéristiques, de sorte que ces réseaux sont utilisés dans de nombreuses applications. Habituellement, nous ne formons qu'un seul des sous-réseaux et utilisons la même configuration pour les autres sous-réseaux. Ces réseaux sont utilisés pour trouver la similarité des entrées en comparant leurs vecteurs de caractéristiques. La figure 3.11 illustre le modèle siamois.

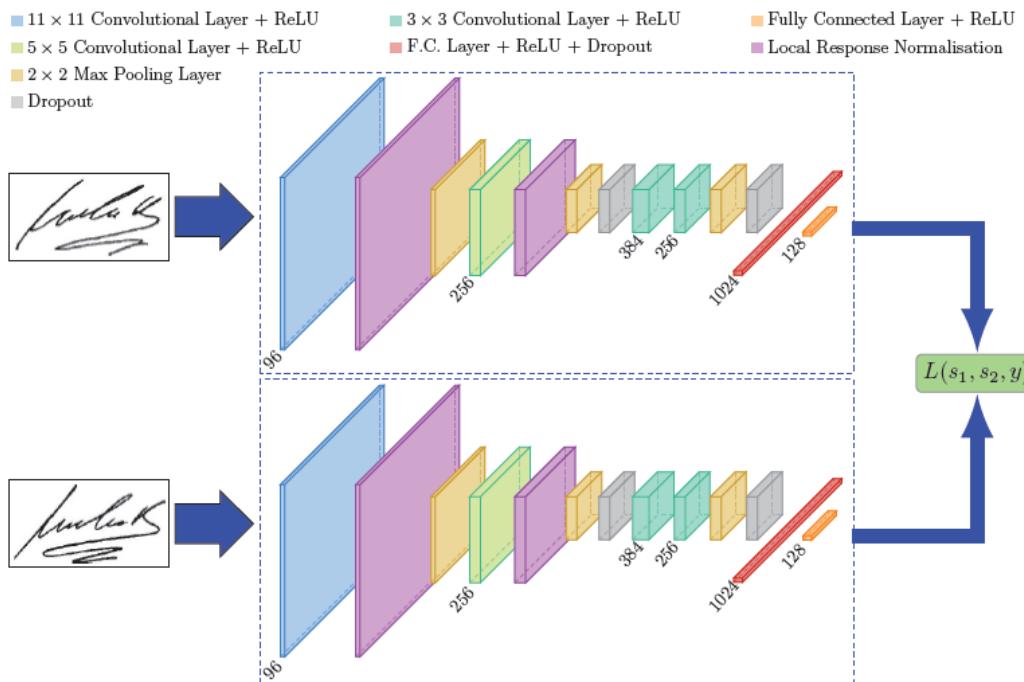


FIGURE 3.11 – Modèle de réseau de neurones siamois [4].

Habituellement, un réseau de neurones apprend à prédire plusieurs classes. Cela pose un problème lorsque nous devons ajouter/supprimer de nouvelles classes aux données. Dans ce cas, nous devons mettre à jour le réseau de neurones et le recycler sur l'ensemble du jeu de données. De plus, les réseaux de neurones profonds ont besoin d'un grand volume de données pour s'entraîner. Les SNN, quant à eux, apprennent une fonction de similarité. Ainsi, nous pouvons l'entraîner pour voir si les deux images sont identiques. Cela nous permet de classer de nouvelles classes de données sans entraîner à nouveau le réseau.

Non seulement l'architecture des sous-réseaux doit être identique, mais les poids doivent également être partagés entre eux pour que le réseau soit appelé "siamois". L'idée principale derrière les réseaux siamois est qu'ils peuvent apprendre des descripteurs de données utiles

qui peuvent ensuite être utilisés pour comparer les entrées des sous-réseaux respectifs. Ainsi, les entrées peuvent être de différents types, comme des données numériques, des données d'image, etc.

3.3.3 Principales caractéristiques du réseau siamois

Les principales caractéristiques d'un réseau siamois sont les suivantes :

- Le réseau siamois prend deux entrées différentes transmises par deux sous-réseaux similaires avec la même architecture, les mêmes paramètres et les mêmes poids.
- Les deux sous-réseaux sont des images miroirs l'une de l'autre, comme des jumeaux siamois. Par conséquent, toutes les modifications apportées à l'architecture, aux paramètres ou aux poids d'un sous-réseau sont appliqués à l'autre sous-réseau.
- Les deux sous-réseaux délivrent un codage pour calculer la différence entre les deux entrées.
- L'objectif d'un réseau siamois est de classer si deux entrées sont identiques ou différentes à l'aide du score de similarité. Le score de similarité peut être calculé à l'aide de l'entropie croisée binaire, de la fonction contrastive ou de la perte de triplet(Triplet loss), qui sont des techniques d'apprentissage métrique à distance commune.
- Le réseau siamois est un classificateur unique qui utilise des caractéristiques discriminantes pour généraliser des catégories inconnues à partir d'une distribution inconnue.

3.3.4 Entraînement au réseau de neurones siamois

Pour entraîner un réseau de neurones siamois, nous effectuons les étapes suivantes dans l'ordre :

- Charger le dataset contenant les différentes classes.
- Créer des paires de données positives et négatives. Une paire de données positive est lorsque les deux entrées sont identiques et une paire négative est lorsque les deux entrées sont différentes.
- Construire le réseau neuronal convolutif, qui produit le codage des caractéristiques à l'aide d'une couche entièrement connectée. Il s'agit de la sœur CNN à travers laquelle nous ferons passer les deux entrées. Les CNN sœurs devraient avoir la même architecture, les mêmes hyperparamètres et les mêmes poids.
- Construire la couche de différenciation pour calculer la distance euclidienne entre les deux réseaux sœurs CNN encodant la sortie.
- La couche finale est une couche entièrement connectée avec un seul nœud utilisant la fonction d'activation sigmoïde pour générer le score de similarité.
- Compiler le modèle en utilisant l'entropie croisée binaire.
- Tester le réseau de neurones siamois.

- Envoyer deux entrées au modèle formé pour générer le score de similarité.
- Lorsque la dernière couche utilise la fonction d'activation ReLU[20], elle produit une valeur supérieure à 0. Un score de similarité proche de zéro implique que les deux entrées sont similaires. Un score de similarité proche supérieur à 1 implique que les deux entrées ne sont pas identiques. La figure 3.12 illustre la sortie par la distance entre 2 images.

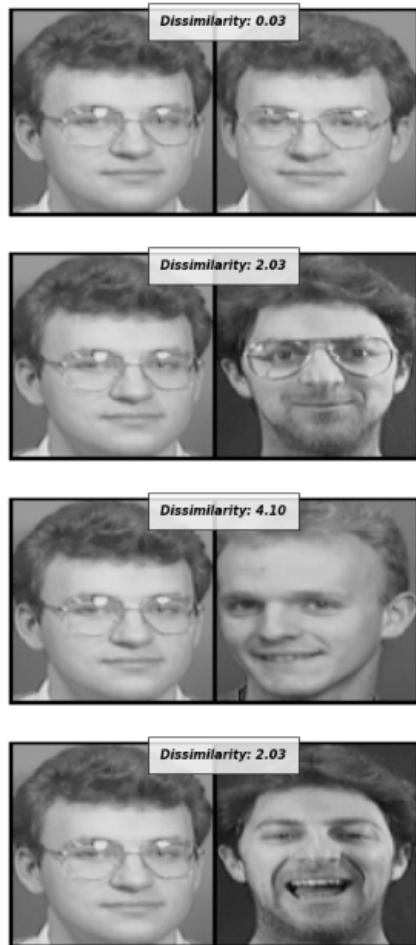


FIGURE 3.12 – la sortie est la distance entre 2 images.

3.3.5 Avantages et inconvénients

Les principaux avantages des réseaux siamois sont :

- Plus robuste au déséquilibre de classes : avec l'aide de l'apprentissage ponctuel (One-shot learning), quelques images par classe sont suffisantes pour que les réseaux siamois reconnaissent ces images à l'avenir.
- Combiné avec le meilleur classificateur : étant donné que son mécanisme d'apprentissage est quelque peu différent de la classification, une simple moyenne de celui-ci

avec un classificateur peut faire beaucoup mieux que la moyenne de deux modèles supervisés corrélés.

- Apprendre à partir de la similarité sémantique : le siamois se concentre sur l'intégration d'un apprentissage qui rassemble des classes/concepts similaires. Par conséquent, il est possible d'apprendre la similarité sémantique.

Les inconvénients des réseaux siamois peuvent être :

- Plus de temps d'entraînement est nécessaire que les réseaux normaux : étant donné que les réseaux siamois impliquent des paires quadratiques pour apprendre (pour voir toutes les informations disponibles), il est plus lent que l'apprentissage normal du classificateur (direction du point d'apprentissage).
- Ne donne pas de probabilités : étant donné que l'entraînement comprend un apprentissage par paires, elle ne donnera pas la probabilité de la prédiction, mais la distance par rapport à chaque classe.

3.3.6 Fonctions de perte

Le réseau siamois utilise le score de similarité pour prédire si les deux entrées sont similaires ou différentes en utilisant une approche d'apprentissage des métriques, qui est la distance relative entre ses entrées.

Le score de similarité peut être calculé à l'aide de *l'entropie croisée binaire* [25], de la fonction *contrastive* ou de *la perte de triplet*.

Le réseau siamois effectue une classification binaire pour classer les entrées comme similaires ou différentes ; par conséquent, l'utilisation de la fonction de perte d'*entropie croisée binaire* est le choix par défaut.

1. Fonction de perte contrastive :

La fonction de perte contrastive [25] différencie les images similaires et dissemblables en contrastant les deux entrées. Cela aide lorsque nous ne connaissons pas toutes les classes au moment de l'entraînement et que nous disposons de données d'entraînement limitées. Elle crée un codage de données qui peut être utilisé lorsque nous aurons de nouvelles classes à l'avenir.

La perte contrastive (voir figure 3.13) nécessite une paire de données d'entraînement positives et négatives. La paire positive contient un échantillon d'ancrage et un échantillon positif, et une paire négative contient un échantillon d'ancrage et un échantillon négatif.

L'objectif de la fonction de perte contrastive est d'avoir une petite distance pour les paires positives et une plus grande distance pour les paires négatives.

$$\mathcal{L}_{contrastive} = (1 - Y) \frac{1}{2} (D_w)^2 + (Y) \frac{1}{2} \max(0, m - D_w)^2. \quad (3.1)$$

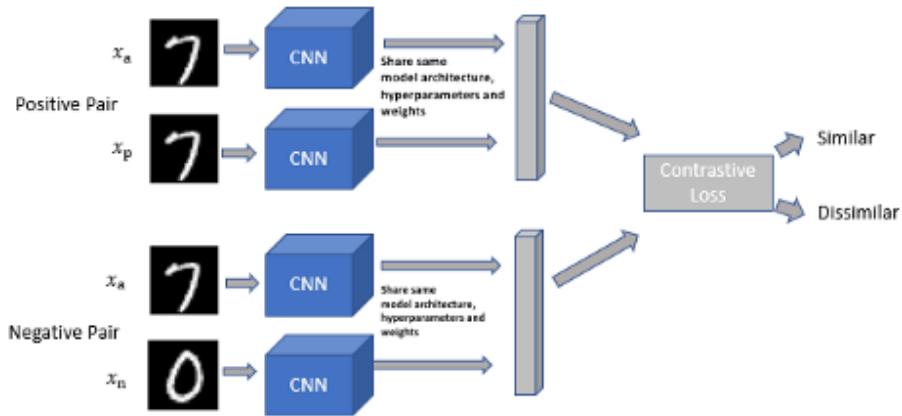


FIGURE 3.13 – Structure du modèle de perte contrastive [5].

Dans l'équation (3.1) ci-dessus, Y vaut 0 lorsque les entrées appartiennent à la même classe et 1 sinon. m est la marge qui définit le rayon pour indiquer que les paires dissemblables au-delà de cette marge ne contribueront pas à la perte et est toujours supérieure à 0. D_w est la distance euclidienne entre les sorties des réseaux siamois frères.

2. Perte de triplet (Triplet Loss) :

Dans Triplet loss [25] (figure 3.14), nous utilisons des triplets de données au lieu de paires. Le triplet est formé d'une ancre, d'un échantillon positif et d'un échantillon négatif et est principalement appliquée pour la reconnaissance du sujet.

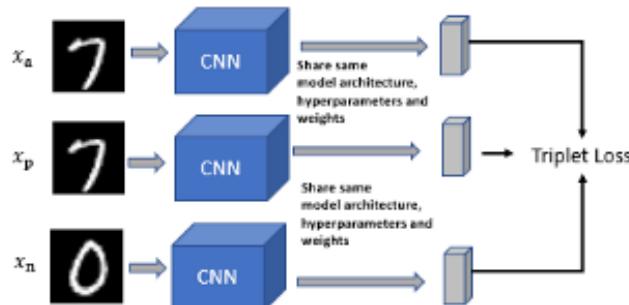


FIGURE 3.14 – Structure du modèle de Perte de triplet [5].

La distance entre l'ancre et le codage de l'échantillon positif est minimisée, et la distance entre l'ancre et les codages de l'échantillon négatif est maximisée.

$$\mathcal{L}_{triplet} = \max((d(a, b) - d(a, n) + marge, 0)). \quad (3.2)$$

La perte de triplet pousse $d(a, p)$ vers 0 et $d(a, n)$ pour être supérieure à $d(a, p) + marge$.

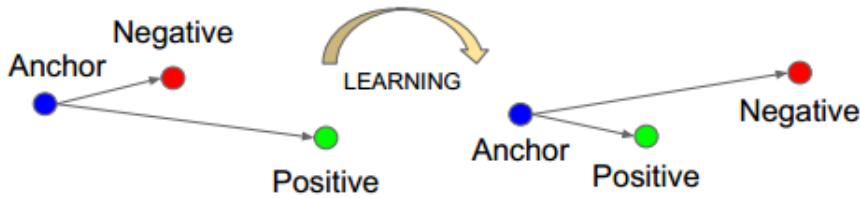


FIGURE 3.15 – La perte triplet minimise la distance entre une ancre et un positif, qui ont tous deux la même identité, et maximise la distance entre l'ancre et un négatif d'une identité différente [6].

3.3.7 Conclusion

Le réseau siamois inspiré des jumeaux siamois est une classification unique pour différencier les images similaires et différentes. Il peut être appliqué même si nous ne connaissons pas toutes nos classes de temps d'entraînement et avons des données d'entraînement limitées. Le réseau siamois est basé sur une approche d'apprentissage des métriques qui trouve la distance relative entre ses entrées en utilisant *l'entropie croisée binaire*, la *perte contrastive* ou la *triple perte*. Dans cette étude, nous utiliserons la *perte contrastive (One-shot)* pour calculer la distance entre les images afin de comparer la différence entre les deux images et créer une matrice de comparaison.

3.4 Yolov4 (DéTECTEUR)

3.4.1 Introduction

Dans un premier temps, nous voulions voir comment un détecteur non modifié fonctionnerait pour détecter les anomalies sous les cinq angles. Nous avons décidé de choisir Yolov4 [7] pour la simple raison qu'il surpassé les méthodes existantes en termes de "performance de détection" et de "vitesse exceptionnelle". L'architecture de Yolov4 contient trois grands composants : la colonne vertébrale (backbone), le cou et la tête (voir figure 3.16).

Dans ce projet, nous prévoyons d'utiliser le modèle de réseau Yolov4 pour prendre en entrée l'image réelle de l'objet imprimé à un instant donné et la matrice de comparaison qui a été générée à partir du comparateur. Yolov4 détecte alors si l'image a des anomalies. S'il y a une anomalie, à quelle anomalie appartient-il ?

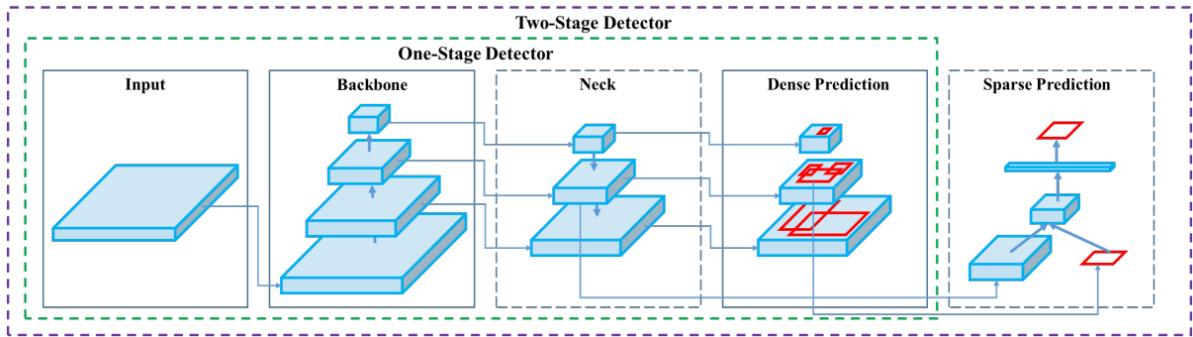


FIGURE 3.16 – Architecture du détecteur à une étape Yolov4 [7].

3.4.2 La backbone CSPDarknet53

La backbone est un réseau de neurones profonds composé principalement de couches de convolution dont l'objectif principal est d'extraire les caractéristiques essentielles. La backbone est généralement recyclée sur *ImageNet* [26], bien que les poids du réseau soient ajustés pour correspondre à une nouvelle tâche de détection d'objets. Dans Yolov4 [7], l'auteur a choisi *CSPDarknet53* [9] comme réseau de la partie backbone après avoir prouvé sa supériorité sur une architecture similaire telle que *ResNet50* [27] ou *DenseNet50* [28].

Avant de passer par l'architecture de CSPDarknet53, nous devons d'abord comprendre l'architecture de *DenseNet* [28]. DenseNet a été proposé comme une amélioration de ResNet, donc au lieu du mappage de connexion qui omet l'élément par élément pour favoriser la propagation du gradient (ResNet [27]), chaque couche de DenseNet [28] prend en entrée toutes les couches précédentes et passe par elle-même cartes de feature à toutes les couches suivantes (voir figure 3.17). En d'autres termes, chaque couche reçoit une connaissance collective (par concaténation) de toutes les couches précédentes, cela permet au réseau d'avoir un meilleur flux de gradient puisque toutes les couches sont interconnectées.

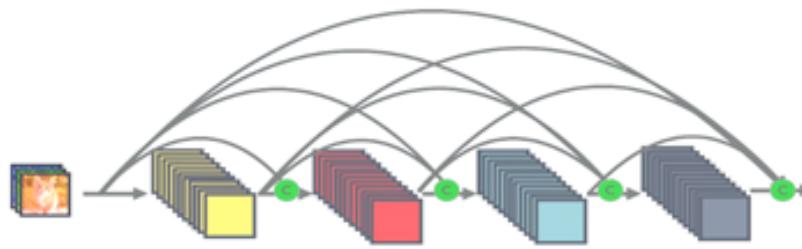


FIGURE 3.17 – DenseNet [8].

Dans l'article du *CSPNet* [9], les auteurs affirment qu'ils ont découvert qu'à l'intérieur du réseau DenseNet [8] [28], il existe un flux d'informations en gradient, c'est-à-dire aux interconnexions entre chaque couche avec toutes les autres couches. Ainsi, pour résoudre

ce problème, les auteurs proposent CSPNet [9] qui signifie Cross-Stage Partial Connection, l'idée en général est de diviser la couche de base en deux parties, une partie passera par le bloc Dense habituel tandis que l'autre partie passera directement à la partie Transition (voir figure 3.18).

Les auteurs de l'article CSPNet [9] ont prouvé qu'en adaptant la Cross Stage Partial Connection, le réseau aura plus d'efficacité et de vitesse. Cela réduisant beaucoup de calculs (puisque nous nous débarrassons des informations de gradient en double). Pour ces raisons, les auteurs choisissent CSP comme backbone du darknet53.

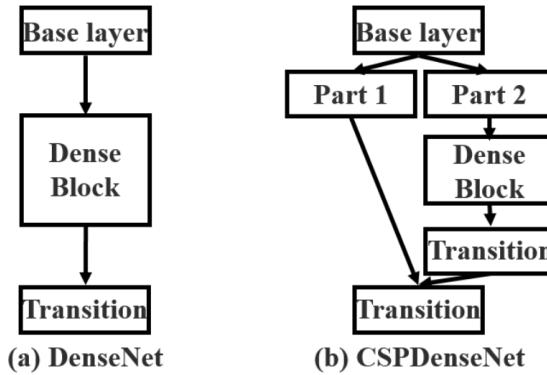


FIGURE 3.18 – Bloc DenseNet et bloc CSPDenseNet [9].

3.4.3 Cou (agrégation de feature)

Le cou est le composant intermédiaire qui est placé entre la colonne vertébrale et la tête, son rôle essentiel est de collecter des cartes de features à partir de différentes étapes de la backbone. Dans cette partie, nous verrons quelques architectures bien connues qui sont généralement utilisées pour l'agrégation de feature ainsi que les modifications et améliorations que Yolov4 [7] leur a apportées.

3.4.3.1 SPP

Pool Pyramidal Spécial SPP [29] a été proposé en 2014, l'idée originale était d'avoir une approche qui peut prendre en entrée des images de tailles différentes. Cette approche a été proposée avant le FCN [30] dans lequel la couche de convolution a été introduite pour la première fois. L'approche était au lieu de la couche entièrement connectée qui force l'entrée à avoir une taille fixe, d'utiliser les couches de Maxpooling en faisant un Max pooling multi-échelle sur chaque canal (voir figure 3.19a), en d'autres termes ils utilisent différents filtres pour chaque carte de features. Par exemple, dans la figure 3.19a, trois filtres de Max pooling de taille (4, 4), (2, 2) et (1, 1) sont appliqués à toutes les 256 cartes de features. Après cela, ils sont concaténés pour donner une sortie de longueur fixe de

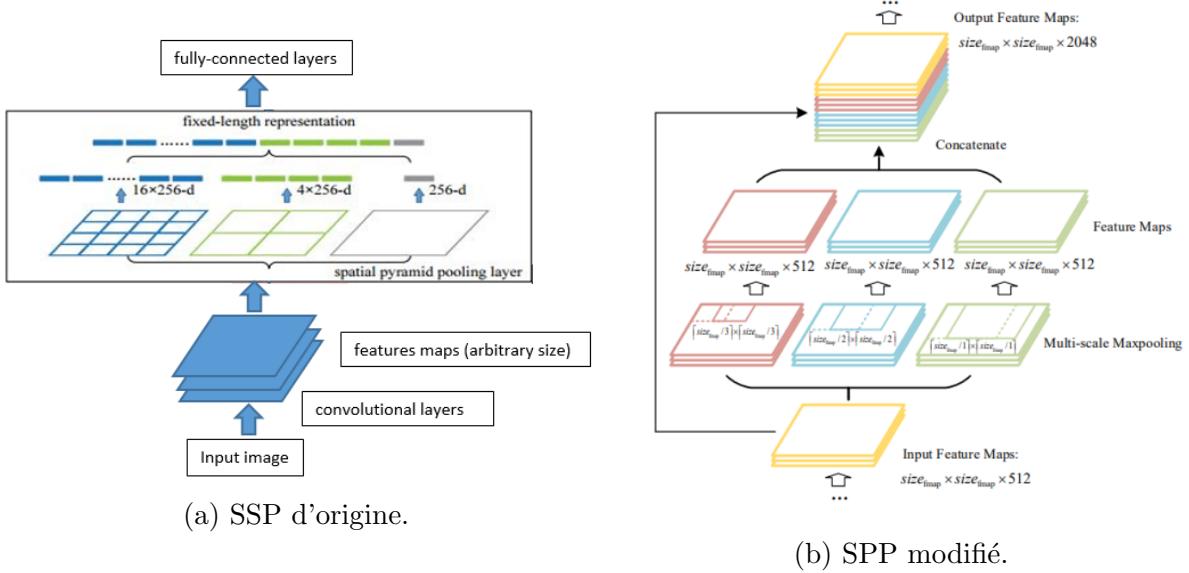


FIGURE 3.19 – SSP d'origine et SPP modifié [10].

taille égale à $(16 + 4 + 1) \times 256$ en sortie. Cela permet au réseau de prendre n'importe quelle taille d'image en entrée.

Pour le SPP modifié qui a été proposé dans Yolov4, max pooling est utilisée avec la stride 1 et le padding, ce qui signifie qu'il n'y a pas de réduction des informations spéciales, après quoi elles sont concaténées dans l'entrée d'origine (voir figure 3.19b), les auteurs , qu'en faisant cela, le réseau a une taille de champ respective beaucoup plus grande, ce qui est très bénéfique pour les composants de tête.

3.4.3.2 PAN

Au début de l'apprentissage en profondeur, des réseaux simples étaient utilisés où une entrée traversait une succession de couches. Chaque couche prend l'entrée de la couche précédente et, comme nous le savons, la couche inférieure contient des informations sur les informations de localisation qui aident le détecteur à localiser correctement les objets. Mais à mesure que nous progressons plus profondément dans le réseau, les informations spatiales qui peuvent être nécessaires pour affiner la prédiction peuvent être perdues. Pour résoudre ce problème, PANet [31] a introduit une architecture qui permet une meilleure propagation des informations de couche de bas en haut ou de haut en bas.

Dans le PANet original (voir figure 3.20a), la couche actuelle et les informations d'une couche précédente sont additionnées pour former un nouveau vecteur. Alors que dans le Yolov4 [7], un nouveau vecteur est créé en concaténant l'entrée et le vecteur d'une couche précédente. Les auteurs ont fait valoir que cette modification permet aux cartes de feature voisines provenant du flux ascendant et du flux descendant de s'écouler facilement vers la tête.

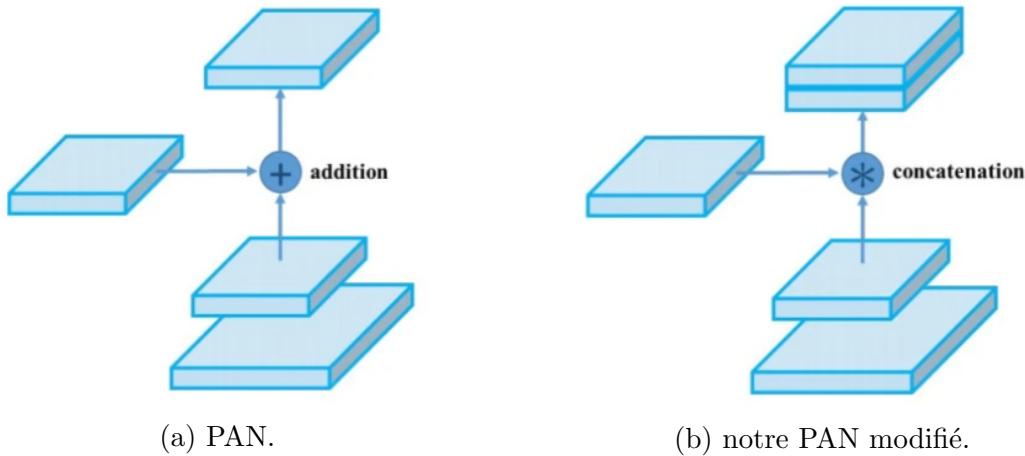


FIGURE 3.20 – PANet [7].

3.4.3.3 SAM : Modèle d’Attention Spatiale

Les modèles d’attention sont largement utilisés dans les réseaux de détection d’objets, car ils ont prouvé leur efficacité dans plusieurs réseaux tels que RCNN [32]. Le concept est simple : au lieu d’apprendre des informations inutiles (arrière-plan), le modèle d’attention utilise en général une simple couche de convolution 2D suivie d’une fonction d’activation (Sigmoïde dans la plupart des cas), pour forcer le modèle à concentrer son attention sur l’apprentissage de features importantes.

Dans **CBAM** (Module d’attention de bloc convolutif) [11], SAM est défini comme suit. Étant donné une carte de features $C_x H_x W$ en entrée, deux transformations distinctes vers la carte de features de sortie d’une couche convective, un Max Pooling et un Avg Pooling sont appliqués. Ces deux éléments sont ensuite concaténés puis introduits dans une couche de convolution suivie d’une fonction sigmoïde pour créer une attention spatiale $1 \times H \times W$ (voir figure 3.21). Cette carte d’attention est ensuite multipliée élément par élément avec la carte de features d’entrée pour obtenir une sortie plus raffinée et mise en évidence.

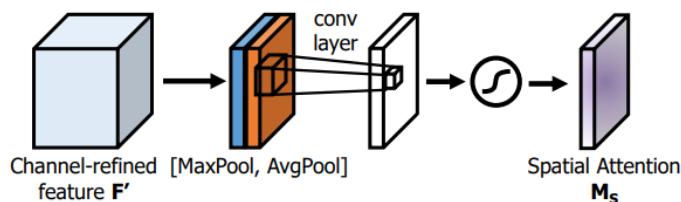


FIGURE 3.21 – Module d’Attention Spatiale Original [11].

Yolov4 [7] applique la même stratégie que SAM, mais avec une petite modification : ils ont supprimé Max Pooling et un Avg Pooling (voir figure 3.22). La carte des features entre dans une couche de convolution 2D ayant sigmoïde comme fonction d’activation, qui est ensuite multipliée par l’entrée initiale.

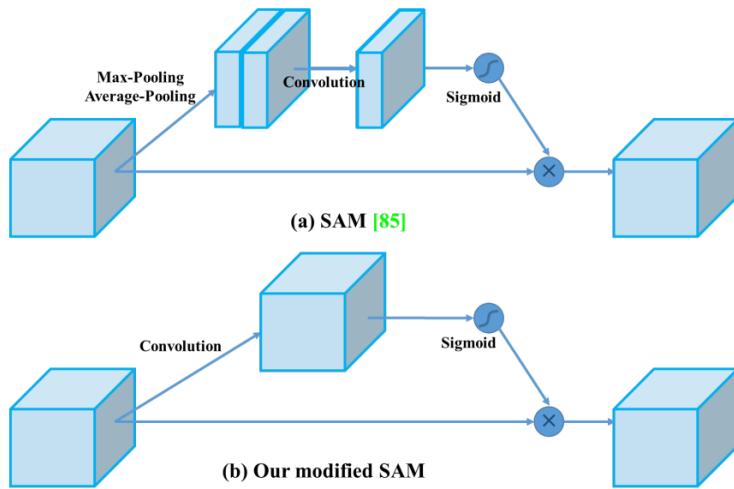


FIGURE 3.22 – SAM modifié [7].

3.4.4 Tête

La tête est un réseau chargé d'effectuer une prédiction dense. C'est la même que la tête de Yolov3 [33]. Les prédictions ont la forme de boîtes englobantes, elles représentent la région d'intérêt de l'objet candidat, ainsi que la probabilité de la classe à laquelle appartient l'objet. La tête utilise des cartes de features de l'image entière pour prédire chaque boîte englobante. Il prédit également toutes les boîtes englobantes dans toutes les classes pour une image simultanément. Grâce à cela, Yolov4 permet un entraînement de bout en bout et des vitesses en temps réel tout en maintenant une précision moyenne élevée.

Pour clarifier le mécanisme de la tête, nous devons passer par ce qui suit :

- **Cellules de grille**

Comme nous le voyons sur la figure 3.23, une image d'entrée est divisée en $S \times S$ cellules de grille. La détection est définie comme si le centre d'un objet tombe dans une cellule de grille, cette cellule de grille est responsable de la détection de cet objet. Chacune de ces cellules de grille prédit B boîtes englobantes et des scores de confiance pour ces boîtes.

- **Boîtes englobantes**

Chaque boîte englobante se compose de 5 prédictions : x, y, w, h et la confiance. Les coordonnées (x, y) représentent le centre de la boîte par rapport aux limites de la cellule de la grille. La largeur et la hauteur sont prédites par rapport à l'ensemble de l'image.

- **Confiance**

Les scores de confiance [34] reflètent la confiance du modèle dans le fait que la boîte contient un objet, ainsi que la précision avec laquelle il pense que la boîte

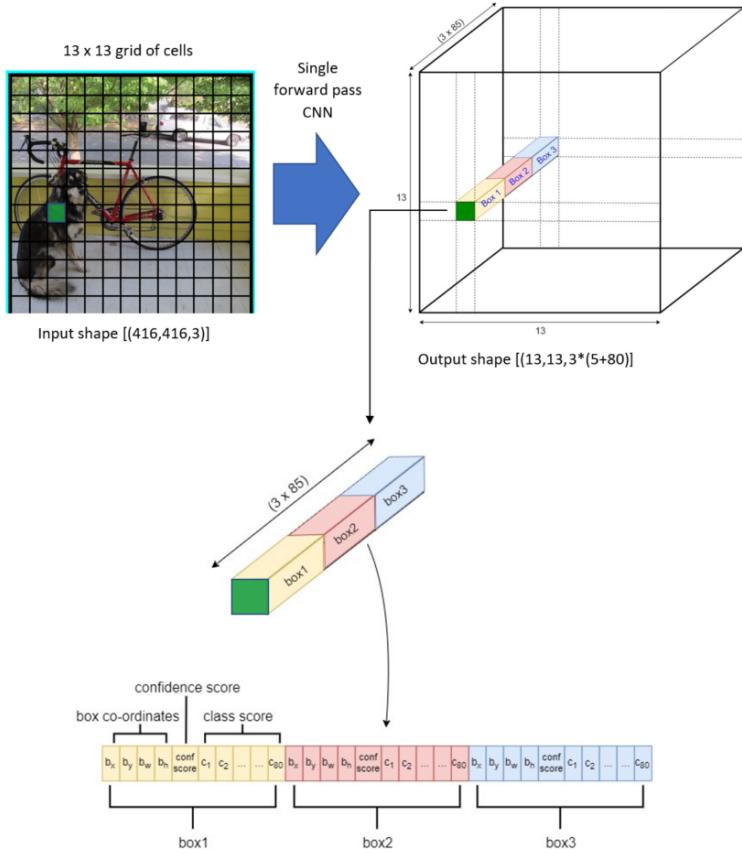


FIGURE 3.23 – Tête de Yolov3[12].

correspond à l'objet. La confiance est définie comme suit :

$$C_i^j = P_r(\text{object}) * IoU_{pred}^{truth}, \quad (3.3)$$

où C_i^j [34] fait référence à la i ^j boîte englobante de la i cellule de grille, $P_r(\text{objet})$ est la probabilité d'existence de l'objet. L'intersection sur l'union (IoU)[34] est définie comme suit :

$$IoU = \frac{\text{overlap_zone}}{\text{union_zone}} = \frac{B \cap B^{gt}}{B \cup B^{gt}}, \quad (3.4)$$

où $B^{gt} = (x^{gt}, y^{gt}, w^{gt}, h^{gt})$ représente la position de la vérité terrain (ground-truth), et $B = (x, y, w, h)$ représente la position de la boîte de prédiction.

3.4.5 Fonction de perte

Les fonctions de perte sont la principale force motrice dans l'entraînement d'un bon modèle. Spécialement dans la détection d'objets car il est difficile de mesurer la précision des boîtes englobantes. La fonction de perte la plus utilisée était Intersection over Union

(IoU), et nous avons vu dans la section principale que Yolov4 utilisait une nouvelle fonction de perte appelée $CIoU(IoU_{pred}^{truth})$ [35]. Pour comprendre comment cela fonctionne, nous aurons d'abord besoin de IoU .

3.4.5.1 Perte IoU

IoU a été introduit comme une solution à la fonction de perte traditionnelle telle que l'erreur Quadratique Moyenne MSE (MSE : Mean Square Error) qui n'était pas adaptée à la détection d'objets. IoU prend en considération la zone de la boîte englobante prévue (BBox) et la vérité terrain (ground truth) de la boîte englobante, et comme nous le voyons à partir de la formule (3.4) ci-dessus, IoU est simplement la zone d'intersection divisée par la zone d'Union. Par conséquent, il est suggéré d'adopter la fonction de perte \mathcal{L}_{IoU} pour la métrique IoU

$$\mathcal{L}_{IoU} = 1 - \frac{|B \cap B^{gt}|}{|B \cup B^{gt}|}. \quad (3.5)$$

3.4.5.2 Perte CIOU

La perte CIOU [35] introduit deux nouveaux concepts par rapport à la perte IoU. Le premier est le concept de distance du point central, qui est la distance entre le point central réel de la boîte englobante et le point central prévu de la boîte englobante. Deuxièmement, le rapport d'aspect. Nous comparons le rapport d'aspect de la vraie boîte englobante et le rapport d'aspect de la boîte englobante prédictive. Avec ces 3 mesures, nous pouvons mesurer la qualité de la boîte englobante prédictive, donnée par

$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{p^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2} + \alpha v, \quad (3.6)$$

où :

- \mathbf{b} et \mathbf{b}^{gt} désignent respectivement les points centraux de la boîte prédictive et de la boîte de vérité terrain.
- $p()$ est la distance euclidienne.
- c représente la longueur diagonale de la plus petite boîte englobante couvrant les deux boîtes.
- α est un paramètre de compromis positif.
- v mesure la cohérence du rapport d'aspect, c'est-à-dire,

$$v = \frac{4}{\pi^2} (\arctan \frac{w^{gt}}{h^{gt}} - \arctan \frac{w}{h})^2, \quad (3.7)$$

où w et h sont respectivement la largeur et la hauteur de la boîte englobante.

3.4.6 Activation Mish

Yolov4 utilise la fonction Mish [36] comme fonction d'activation au lieu d'utiliser ReLu ou Relu fuyant [20] dans le réseau backbone. Cette fonction d'activation (voir figure 3.24) montre des résultats très prometteurs. Par exemple, l'exécution de tests sur l'ensemble de données CIFAR-100 a respectivement augmenté la précision de 0,494 et 1,671 par rapport au même réseau avec Swish et ReLU. La fonction Mish est définie par

$$f(x) = x \tanh(\ln(1 + e^x)). \quad (3.8)$$

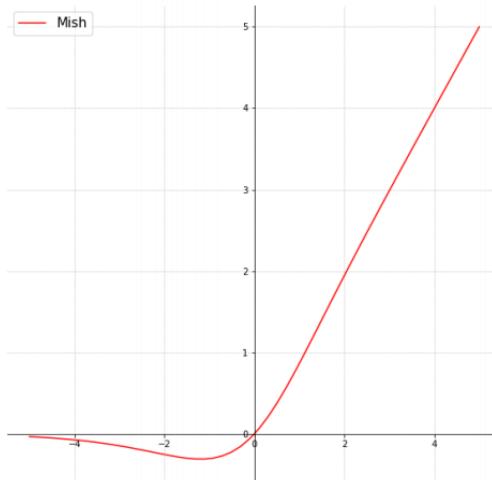


FIGURE 3.24 – Fonction d'activation Mish [13].

3.4.7 Conclusion

Comme nous pouvons le voir, il existe de nombreuses classes pour construire un modèle de détection d'objets. Pour Yolov4, les chercheurs ont décidé de faire le meilleur compromis entre le *mAP* et la vitesse d'apprentissage et d'inférence du modèle permettant son utilisation dans des dispositifs embarqués. Nous avons pu voir que ses performances ne lui viennent pas d'une architecture complètement nouvelle, mais d'une combinaison efficace de nombreuses optimisations pour la plupart déjà présentes dans d'autres modèles de la littérature scientifique, et inclue à l'architecture éprouvée de la famille Yolo.

3.5 Générateur

3.5.1 Introduction

Dans notre approche, le générateur représente la vérité terrain, car il simule l'impression d'un objet à partir de son fichier GCode, ce qui implique qu'il restituera des images

du modèle 3D à la fin de chaque couche (vérité terrain), pour chaque vue de caméra. Cependant, afin d'avoir un générateur précis, nous aurons besoin de la position des caméras par rapport à l'imprimante 3D.

3.5.2 Étalonnage

Le calibrage de la caméra vise à déterminer les paramètres géométriques de l'image, ces paramètres font référence à la fois à l'intrinsèque et à l'extrinsèque. Intrinsèques sont les paramètres internes de la caméra elle-même, y compris la distance focale, le point principal et l'inclinaison de l'axe. Tandis que les extrinsèques expriment la position de la caméra dans le monde (translation et rotation).

Selon le modèle de caméra Pinhole [37], la relation entre les coordonnées spatiales d'un point (X, Y, Z) dans l'espace avec le point associé dans l'image prise par la caméra, est définie par la formule (3.9) ci-dessous :

$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}, \quad (3.9)$$

où :

- p signifie le "picture plane".
- w signifie le "world space".
- La matrice $K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$ contient les paramètres intrinsèques de la caméra avec f_x, f_y et c_x, c_y respectivement les distances focales et le centre optique dans les coordonnées de l'image.
- $R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$ est la matrice de rotation et $T = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$ est le vecteur de translation.

3.5.3 Simulateur GCode

Les résultats sont générés à partir du programme de simulation. Le simulateur, compte tenu des paramètres d'angle de caméra en entrée avec un fichier GCode, affiche un ensemble de données de cinq images pour chaque couche d'impression, chacune avec un angle de vue correspondant aux images réelles de la caméra.

3.6 Conclusion

Les approches que nous avons décrites ont été analysées et considérées comme réalisables pour notre projet. Parmi eux, le comparateur utilise le réseaux neuronal *siamois* pour créer une matrice de comparaison entre l'image réelle et l'image contenant l'angle

de vue correspondant dans le fichier GCode. Nous utilisons ensuite cette matrice de comparaison comme entrée du réseau Yolov4 pour détecter le type d'erreur de l'image réelle.

Nous avons testé les approches ci-dessus sur des données d'exemples et obtenu des résultats positifs. De plus, leur efficacité a également été démontrée dans des articles scientifiques récents. Cependant, à l'avenir, pour être plus sûrs, nous utiliserons sur notre jeu de données réel (comprenant : 5 images de 5 angles de caméra et 5 images de GCode correspondant à 5 angles de caméra) pour obtenir les résultats les plus précis.

4

Base de données

Sommaire

4.1	Introduction	35
4.2	Distribution et étiquetage des classes	37
4.2.1	Classes	37
4.2.2	Étiquetage	38
4.2.3	Distribution	39

4.1 Introduction

Le gros problème avec la détection des anomalies est le manque de données. l'ensemble de données de défaut d'une imprimante 3D plug-and-play était introuvable sur Internet. Du coup, notre équipe n'a eu d'autre choix que de visiter les imprimantes 3D situées à l'université, et d'essayer de créer ces anomalies manuellement. Au final, nous avons réussi à produire certaines anomalies qui, en manipulant les paramètres de l'imprimante 3D, ne pourraient autrement pas en provoquer d'autres.

Les anomalies que nous avons pu générer étaient suffisantes pour nos besoins, puisque selon le superviseur des imprimantes 3D, ce sont celles qui surviennent le plus fréquemment (voir figure 4.1). Dans la section suivante, nous passerons en revue chaque défaut ainsi que la façon dont nous l'avons créé.

— Warping :

Le warping se produit en raison du rétrécissement de la matière lors de l'impression 3D, ce qui entraîne le soulèvement et le détachement des coins de l'impression de la plaque de construction (lit). Nous générions ce défaut en utilisant l'**ABS** comme filament d'impression en raison de sa propriété de retrait et en refroidissant le lit. Cela provoque la contraction de l'objet et sa courbure de l'objet 3D à partir de la plaque de construction.

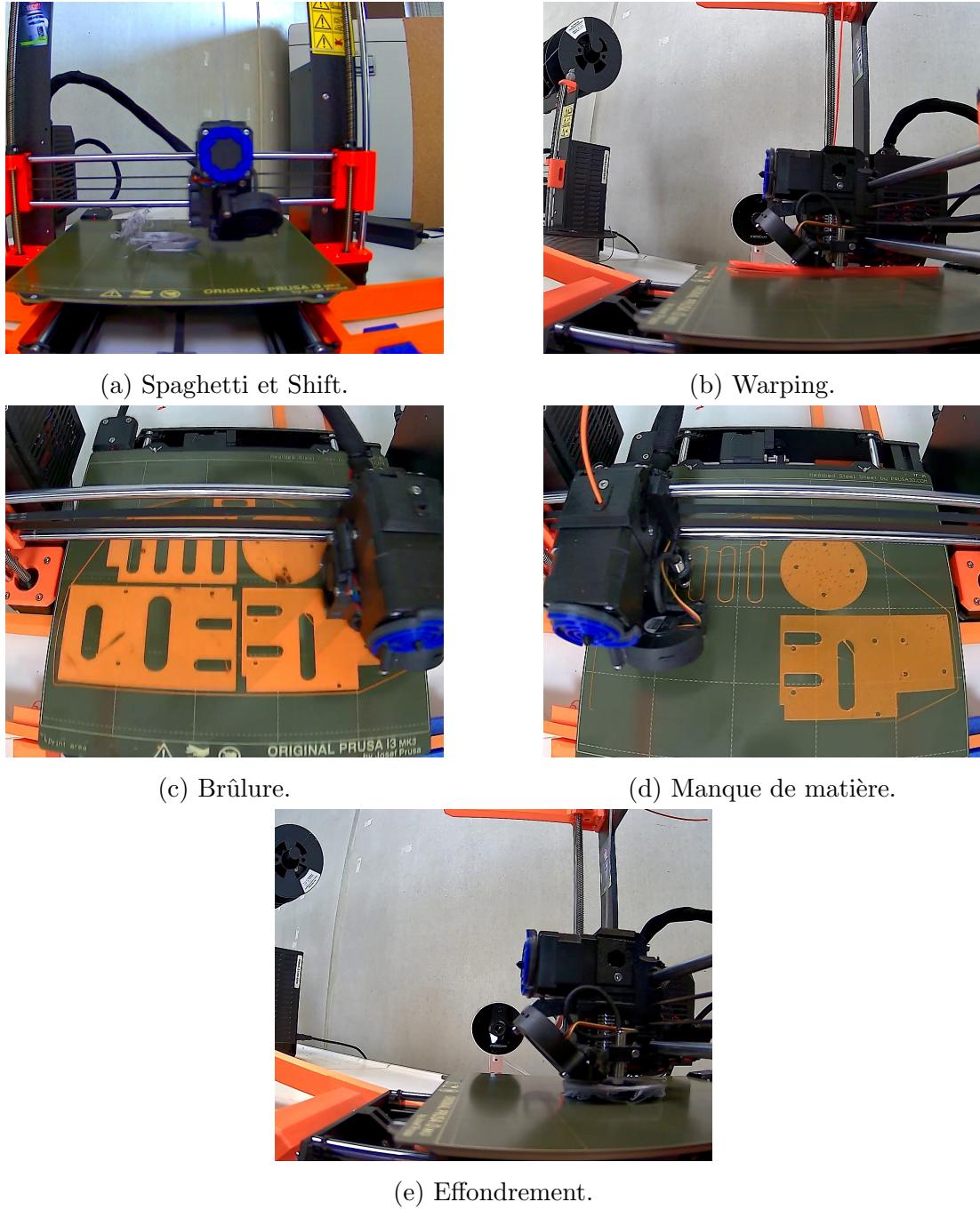


FIGURE 4.1 – Défauts générés.

- **Spaghetti :**

Le spaghetti est l'un des défauts les plus fréquents. Il se produit lorsque le filament extrudé est mal placé par l'extrudeuse (tête d'impression) car, à un certain moment de l'impression, l'objet a bougé ou s'est effondré. Nous avons causé ce problème en modifiant le fichier GCode, soit en supprimant une section du GCode, soit en modifiant la valeur Z .

- **Shift :**

Le shift ou décalage d'un calque le long des axes X ou Y . Cela se produit lorsqu'il y a un mauvais placement de l'extrudeuse ou du lit. Nous avons causé ce problème en mettant l'imprimante en pause et en déplaçant manuellement le lit ou l'extrudeuse par petite marge, puis en reprenant l'impression.

- **Brûlure :**

Le brûlure est la présence de plastique brûlé, engendrant une tache sombre sur l'objet (noire dans la plupart des cas). Nous l'avons générée en démontant la buse (la partie de la tête d'impression qui extrude le filament).

- **Effondrement :**

L'effondrement ou la mauvaise qualité de la surface saillante est également un problème très fréquent. Une cause possible est que la solidification de la résine déposée à la périphérie de la saillie n'a pas été assez rapide et que les fibres déposées ont migré avant de se solidifier. Nous formons ce défaut en réduisant la chaleur et en considérant un objet 3D avec une face saillante.

- **Manque de matière :**

C'est également un défaut courant et il existe de nombreuses raisons pour les trous et manques de matière dans une impression, mais elles sont principalement liées à l'imprimante ou au lit d'impression. Nous avons généré cela, mais en modifiant la distance entre la buse et le lit.

4.2 Distribution et étiquetage des classes

4.2.1 Classes

En plus des cinq défauts que nous avons couverts ci-dessus, nous incluons trois autres classes. Nous avons pensé qu'il serait essentiel que notre modèle apprenne à détecter et à identifier l'extrudeuse et l'objet 3D, car il sera plus facile d'ignorer l'extrudeuse et de se concentrer sur la zone de l'objet. La troisième classe que nous avons ajoutée est le fil, lors de l'impression et surtout pour les objets qui demandent beaucoup de temps, il arrive

que le filament soit complètement utilisé mais que l'impression continue sans extruder de filament, ce qui provoque un objet incomplet. Pour cette raison, le modèle apprend à détecter la classe de fil et à suspendre l'impression dès qu'aucun fil n'est détecté. La figure 4.2 illustre l'étiquetage des classes de fils, d'extrudeuses et d'objets.

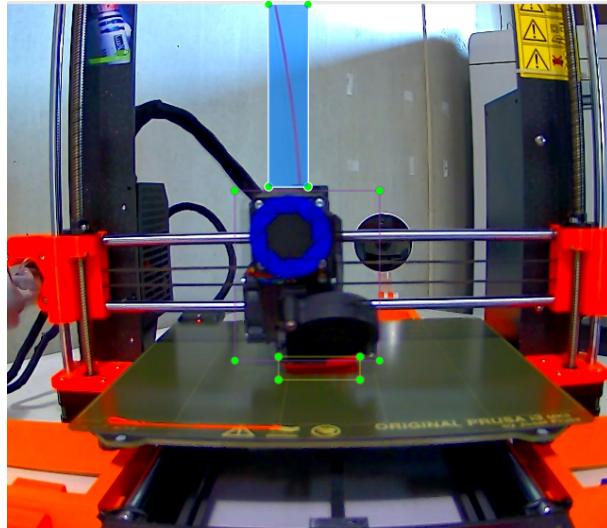


FIGURE 4.2 – Classes Fil, Extrudeuse et Objet.

Par nature, certains défauts se produisent dans certaines zones, ce qui implique que certains seront plus susceptibles d'être détectés sous un certain angle de vue. Pour résumer, le tableau 4.1 montre chaque défaut ainsi que les angles de vision des caméras qui peuvent le détecter (objet et extrudeuse sont exclus).

Classes	Angle de caméra
Warping	Avant, Arrière, Droite, Gauche
Spaghetti	Haut, Avant, Arrière, Droite, Gauche
Shift	Avant, Arrière, Droite, Gauche
Brûlure	Haut
Fil	Avant, Arrière
Manque de matière	Haut

TABLE 4.1 – Anomalies par angle de caméra.

4.2.2 Étiquetage

Pour l'étiquetage, nous n'avions pas d'autre choix que d'étiqueter manuellement toutes les images. Mais avant cela, nous avons utilisé les informations du fichier GCode pour créer un algorithme d'estimation du temps par couche. Nous avons cadré les vidéos en

choisissant uniquement les cadres où le calque vient d'être imprimé. En ce qui concerne l'outil d'étiquetage, nous avons utilisé LabelImg [14], un cadre d'étiquetage open source gratuit et facile à utiliser. Pour étiqueter un objet dans une image, nous faisons simplement glisser et déposer une boîte englobante sur l'objet, puis une fenêtre contextuelle contenant une liste des classes que nous avons précédemment définies. En sélectionnant la classe et en sauvegardant un fichier .txt est créé avec le même nom que l'image et dans le même répertoire.

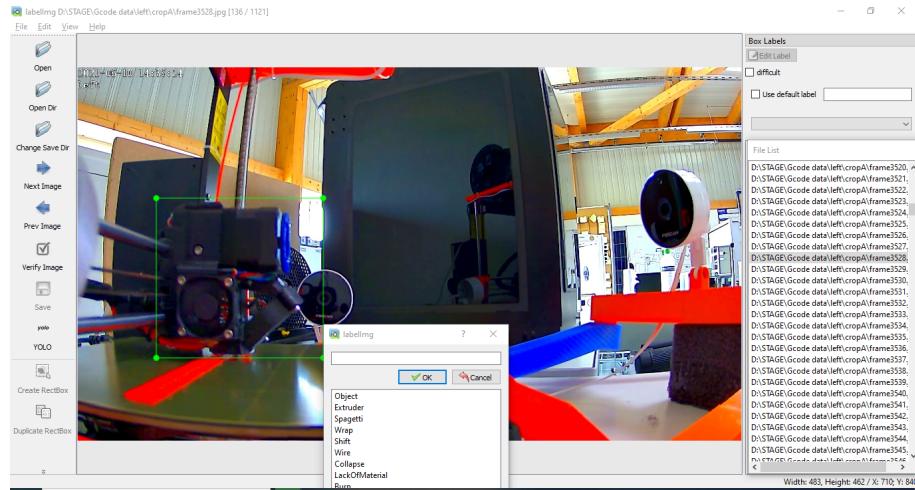


FIGURE 4.3 – Programme LabelImg [14].

Chaque fichier .txt contient une liste de classes et une boîte englobante de son image correspondante, chaque ligne ayant le format d'annotation yolo, défini comme suit :

$$< \text{object_class } x_{\text{Centre}} \text{ } y_{\text{Centre}} \text{ largeur hauteur } >, \quad (4.1)$$

où :

- *object_class* : nombre entier faisant référence à une classe unique.
- *xCentre* et *yCentre* : représentent les coordonnées du point central de la boîte de liaison correspondante.
- *largeur* et *hauteur* : valeurs flottantes relatives à la largeur et la hauteur de l'image, leurs valeurs varient de 0.0 à 1.0.

4.2.3 Distribution

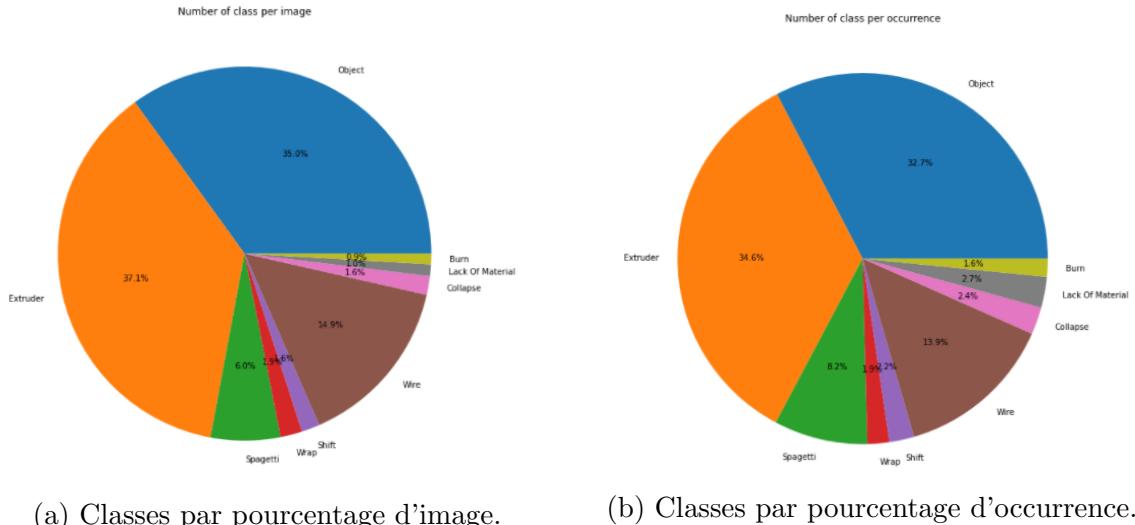
Chaque image peut contenir plusieurs classes, il a donc fallu un temps considérable (plus de 6 semaines) pour concevoir notre jeu de données : cadrage vidéo, étiquetage, recadrage d'image, imbrication avec comparaison de matrices, etc. Au final, nous avons reçu 5155 images taguées, issues de 12 impressions 3D différentes, chacune avec une forme et une couleur, et des anomalies survenues lors du processus d'impression. Le tableau 4.2 indique le nombre de fois que nous pouvons provoquer une erreur dans l'objet imprimé.

Classes	Nombre d'objets
Warping	2
Spaghetti	5
Shift	4
Brûlure	2
Fil	12
Manque de matière	4

TABLE 4.2 – Nombre d'anomalies par Objet.

Il est important de mentionner que lorsque certains défauts tels que le *manque de matière* ou la *brûlure* se produisent, ils peuvent ne rester visibles que pendant une courte période de temps, après quoi ils sont cachés par la couche suivante.

Dans les figures 4.4, nous voyons le pourcentage de chaque classe par rapport au nombre d'occurrences (voir figure 4.4a) et par rapport au nombre d'images (voir figure 4.4b).



(a) Classes par pourcentage d'image.

(b) Classes par pourcentage d'occurrence.

FIGURE 4.4 – Pourcentage de chaque classe.

Nous excluons les classes Object et Extruder dans la figure 4.5.

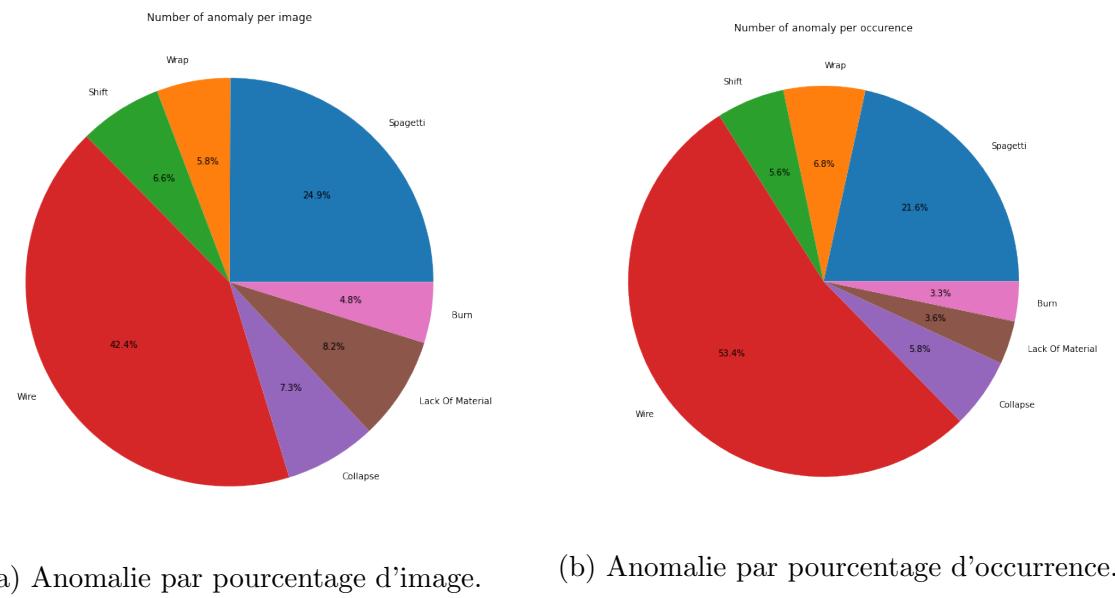


FIGURE 4.5 – Pourcentage de chaque anomalie

5

Expérimentations et résultats

Sommaire

5.1	Introduction	42
5.2	Outils utilisés	43
5.3	Générateur	44
5.3.1	Étalonnage	44
5.3.2	Simulation	44
5.4	Comparateur	45
5.4.1	Introduction	45
5.4.2	Résultat	47
5.4.3	Méthode alternative.	48
5.5	Détecteur	49
5.5.1	Data Augmentation	49
5.5.2	Méthode de régularisation	50
5.5.3	Entraînement	51
5.5.4	Expériences	52
5.6	Problèmes rencontrés	63
5.7	Conclusion	64

5.1 Introduction

Dans ce chapitre nous allons décrire les techniques de mise en œuvre, d'entraînement, des difficultés rencontrées et enfin comparer et discuter des différents résultats des expérimentations menées pour notre étude sur la base d'architecture hybride (la figure 2.1) présentée dans le chapitre 2.

5.2 Outils utilisés

— Python 3

Python est un langage de programmation populaire, open source, dynamique et de haut niveau qui est largement utilisé pour produire des réseaux d'apprentissage en profondeur. La force de Python par rapport aux autres outils de programmation réside dans ses vastes bibliothèques de support. De plus, cet outil de programmation permet à ses utilisateurs d'effectuer une variété de tâches en seulement quelques lignes de code, contrairement à d'autres langages de programmation, tels que Java, C++, etc.

— Darknet

Darknet [9] est un framework open source hautes performances pour la mise en œuvre de réseaux de neurones. Écrit en C et CUDA, il peut être intégré aux CPU et aux GPU. Des implémentations avancées de réseaux de neurones profonds peuvent être effectuées à l'aide de Darknet. Ces implémentations incluent la classification ImageNet [22], les réseaux de neurones récurrents (RNN) [38] et bien d'autres.

— Matplotlib

Matplotlib est une bibliothèque graphique populaire pour le langage de programmation Python. C'est l'outil de visualisation de données incontournable de tout chercheur. Elle est largement utilisée pour créer des graphiques de haute qualité.

— Moviepy

MoviePy est un module Python pour le montage vidéo, qui peut être utilisé pour des opérations de base telles que les coupes, les concaténations, la composition vidéo, le traitement vidéo.

— Mayavi

Mayavi [39] est un framework Python qui permet une visualisation simple et interactive des données 3D. **Matplotlib** peut également faire des tracés 3D simples, mais nous choisissons Mayavi car il s'appuie sur un moteur puissant, *VTK* [39], ce qui le rend plus adapté à l'affichage de données volumineuses ou complexes.

— OpenCV

OpenCV (Open Source Computer Vision) [40] est une bibliothèque de fonctions de programmation principalement destinées à la vision par ordinateur en temps réel. En langage simple, c'est la bibliothèque utilisée pour le traitement d'image. Il est principalement utilisé pour effectuer toutes les opérations liées à l'image.

5.3 Générateur

5.3.1 Étalonnage

Vision par ordinateur Matlab. Pour les paramètres de la caméra, nous avons utilisé Matlab, il fournit de nombreuses tâches de vision par ordinateur, dans lesquelles une application de calibrage [41] avec une interface utilisateur facile à utiliser. D'abord, nous plaçons une image de motifs en damier (voir figure 5.1), devant la caméra, puis nous tournons l'image de 5 degrés pour collecter dans plusieurs directions, afin d'assembler environ 10 à 20 images distinctes. Après cela, nous fournissons la taille du damier de notre image.

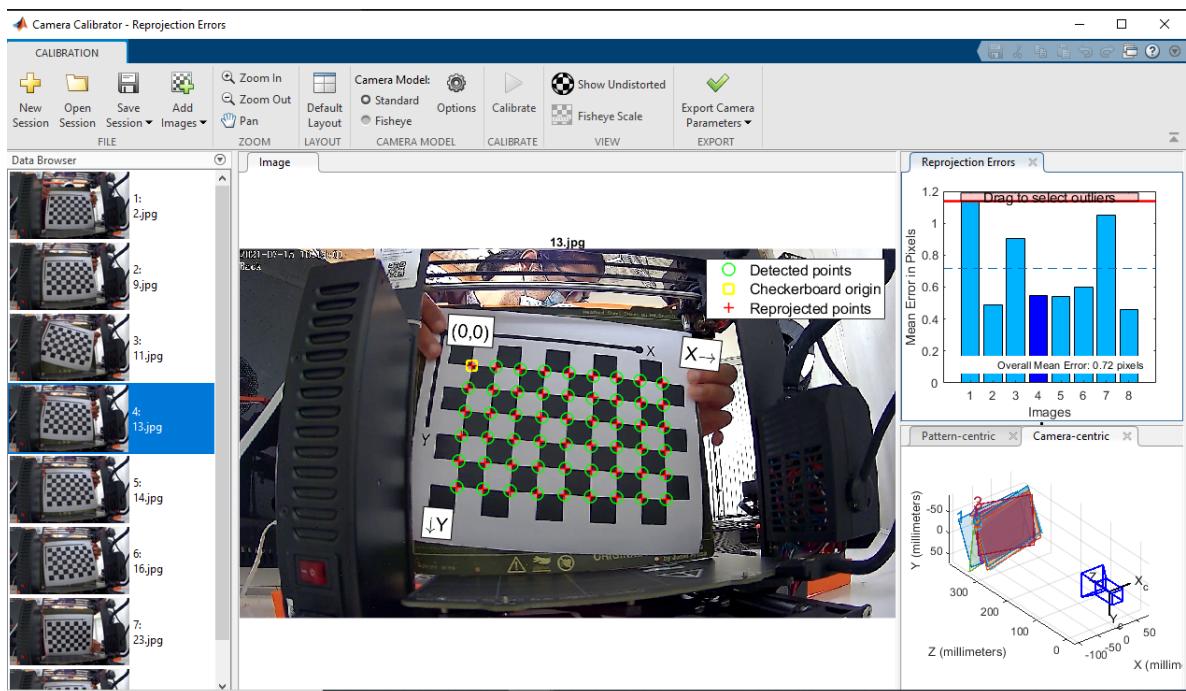


FIGURE 5.1 – Calibrage de la caméra de Matlab.

5.3.2 Simulation

Comme nous avons rassemblé toutes les informations requises pour le générateur, des paramètres de la caméra au fichier GCode. Nous avons utilisé avec succès ces informations pour simuler le processus d'impression d'un modèle 3D. Là où nous avons d'abord pris une image et mesuré la taille du lit de l'imprimante, nous avons ensuite créé un plan maillé avec la taille du lit et l'image comme texture.

En plus de cela, nous avons créé un fichier GCode de l'extrudeur pour simuler son mouvement lors de l'impression selon le plan (O, X, Z). Nous avons également configuré le

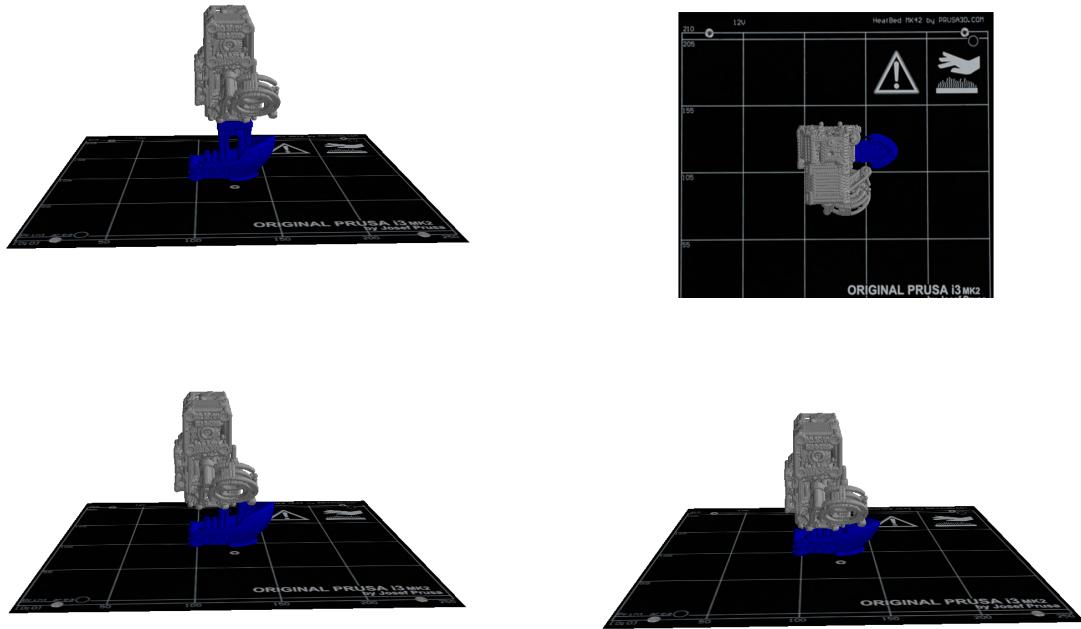


FIGURE 5.2 – Simulation d'impression d'un modèle 3D à partir de deux perspectives (avant, dessus).

générateur pour déplacer le lit avec l'objet 3D lors de l'impression le long des axes (O, Y).

Nous avons créé le programme [A.1](#) qui capture les faces des objets d'un fichier GCode à intervalles réguliers du début à la fin de l'impression et l'enregistre sous forme de fichiers images.

Dans la figure 5.2, nous voyons une simulation d'impression d'un modèle de bateau 3D à partir de deux perspectives (avant, dessus), à différents pas de temps.

Grâce au programme [A.1](#), nous pensons que le modèle **Traducteur** (que nous décrivons dans la section [2.2.2](#)), permettra de cartographier plus facilement les correspondances entre l'espace réel des mots et l'espace simulé. Nous illustrons cela dans la figure 5.3.

5.4 Comparateur

5.4.1 Introduction

Notre objectif est d'entraîner un modèle de réseau siamois capable de détecter des anomalies dans le processus d'impression grâce à la saisie d'images virtuelles (extraites de fichiers GCode) et d'images réelles correspondantes (extraites de 5 caméras) à des temps fixés.

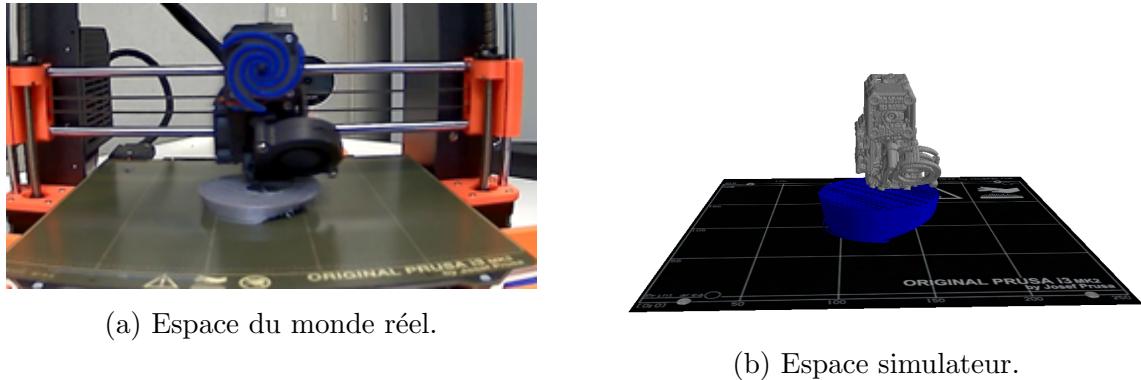


FIGURE 5.3 – L'espace simulateur correspond à l'espace réel.

Nous divisons l'image virtuelle (taille 1184×864 pixels) et l'image réelle (1184×864 pixels) en petits patchs de 32×32 pixels et les sauvegardons en tant que fichier image comme le montre la figure 5.4. Ensuite, nous prenons chaque paire de patchs d'image

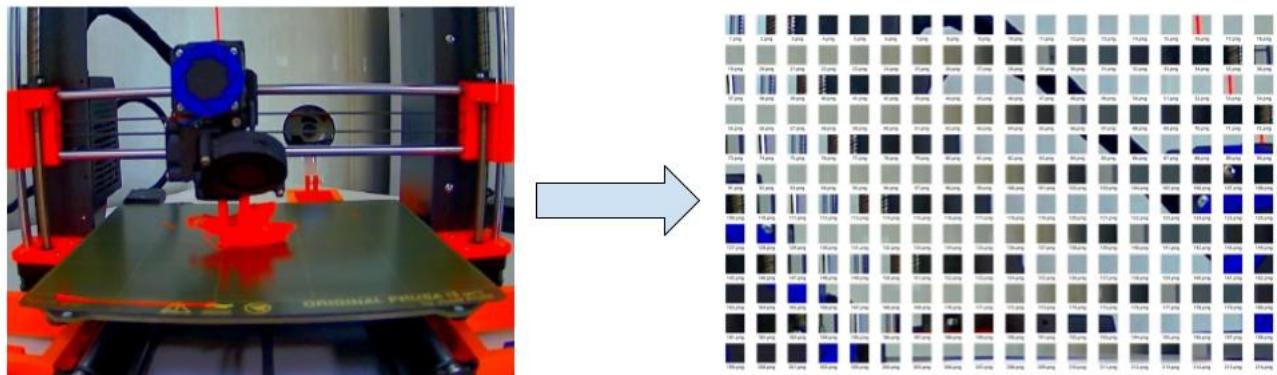


FIGURE 5.4 – Image est subdivisée en plusieurs patchs de 32×32 pixels.

correspondants et les comparons les uns aux autres via le modèle de réseau siamois que nous avons entraîné. Si les paires de patchs sont identiques, le résultat est 0 et s'ils sont différents, le résultat est strictement supérieur à 0 (plus ils sont différents, plus la valeur est élevée). Ces résultats sont créés ensemble dans une matrice de comparaison (27×32 éléments) avec des éléments représentant différentes valeurs des paires d'images de patch. La figure 5.5 décrit mieux le processus d'utilisation d'un réseau siamois pour comparer des paires d'images et créer une matrice.

Pour découper une image en patchs, nous utilisons les logiciels PhotoFlare [42] et PhotoFiltre [43] pour découper les parties redondantes de l'image et assembler l'image afin que la position de l'objet soit dans leurs coordonnées respectivement afin que les deux photos aient la même taille et la même coordonnées de l'objet imprimé en 3D entre l'image

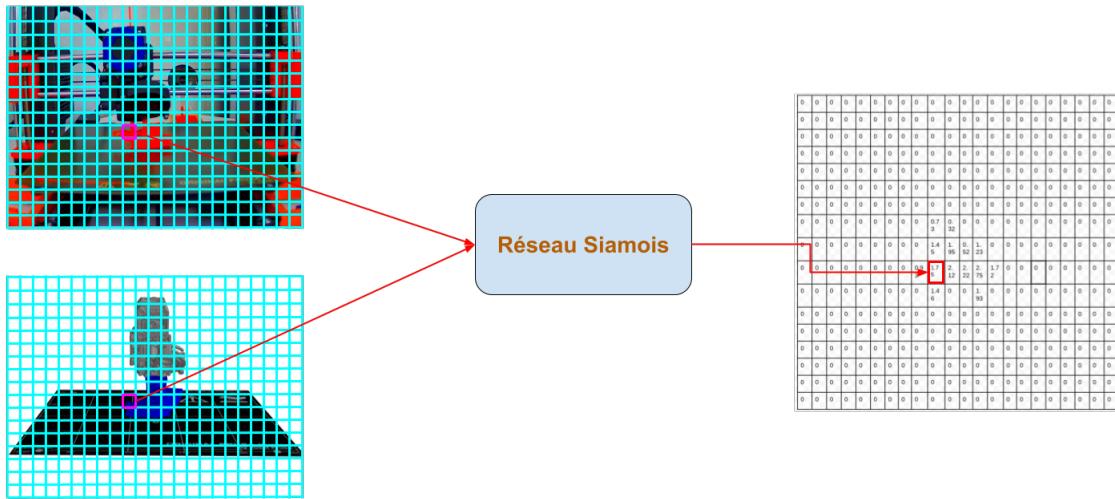


FIGURE 5.5 – Calibrage de la caméra de Matlab.

réelle et l'image virtuelle (voir annexe B.2 pour une capture écran lors de utilisation de PhotoFiltre).

Après, nous avons créé le programme A.2 pour découper les images en petits patchs correspondants et les enregistrer dans 2 fichiers de dossiers séparés (dossier image réelle et dossier image virtuelle). Une fois que suffisamment de données ont été collectées, il est nécessaire d'utiliser le modèle siamois pour comparer chaque paire de patchs entre eux et la sortie sera une matrice de comparaison composée des résultats de comparaison de chaque paire de patchs.

5.4.2 Résultat

Nous avons utilisé la base de données comprenant des images virtuelles d'objets et des images réelles contenant des objets anormaux sur 2 modèles siamois en utilisant des fonctions de *perte contrastive* et de *perte de triplet* (voir sous-section 3.3.6). Cependant, en raison du manque de données d'entrée (des images où l'objet n'avait pas d'anomalies) à entraîner, nous n'avons pas pu terminer cette partie. Pour l'entraînement, nous avons besoin d'une base de données composée d'images virtuelles et d'images réelles correspondantes à des images virtuelles. Au lieu de cela, nous utilisons des images réelles qui contiennent des anomalies. Cela rend la partie la plus importante de la matrice de comparaison, qui contient l'objet imprimé en 3D, imprécise. Comme le temps était limité et qu'il n'était pas possible d'obtenir une vidéo de l'objet imprimé en 3D complet capturé par la caméra (aucune anomalie), nous avons créé un autre programme pour comparer ces patchs d'image entre eux dont nous parlerons dans la sous-section suivante.

5.4.3 Méthode alternative.

OpenCV (Open Source Computer Vision) [40] est une bibliothèque de fonctions de programmation principalement destinées à la vision par ordinateur en temps réel. En langage simple, c'est une bibliothèque utilisée pour le traitement d'image. Elle est principalement utilisée pour effectuer toutes les opérations liées à l'image. Ici, nous utilisons cette bibliothèque pour programmer un programme pour remplacer le réseau *siamois* [24].

Nous avons utilisé *openCV* pour traiter les images et encadré les parties comparées en utilisant les coordonnées fournies par le fichier yolo texte dont nous disposions. Nous avons créé le programme A.2 pour prendre en entrée les images réelles et virtuelles traitées respectivement et le fichier yolo texte que nous avons créé manuellement dans la sous-section 4.2.2 pour cadrer les zones d'anormaux de l'image réelle (voir figure 5.6). À partir de là, nous comparons ces parties avec les parties correspondantes du fichier GCode pour trouver les différents résultats des correctifs et les enregistrer dans la matrice de comparaison. Le reste en dehors du carré nous laissons les valeurs par défaut à 0.

La matrice de comparaison résultante (voir figure B.1) a le résultat souhaité. Cependant,

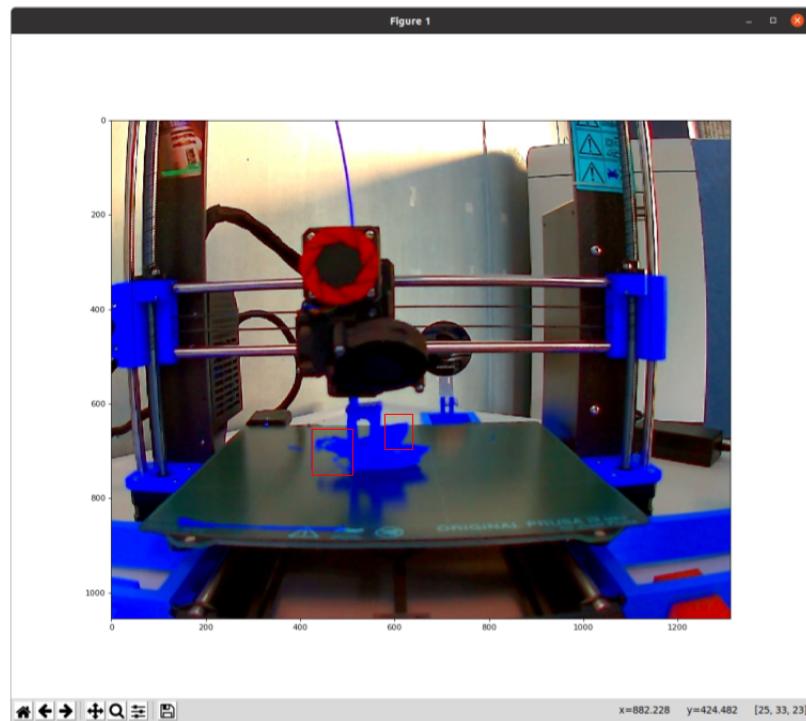


FIGURE 5.6 – Encadrer les parties anomalies du fichier image réel.

nous devons le faire manuellement pour encadrer les objets anomalies dans le fichier image réelle. Cela affectera le processus d'automatisation lorsqu'il sera utilisé pour le programme de détection d'anomalies dans le processus d'impression 3D. Entraîner un modèle de réseau *siamois* pour générer automatiquement des matrices de comparaison reste la solution la

plus appropriée. Cependant, dans le cadre expérimental et pas encore nécessaire à la mise en place d'un programme de comparaison automatique, le programme A.2 a parfaitement répondu à nos besoins.

5.5 Détecteur

5.5.1 Data Augmentation

L'augmentation des données est une méthode qui améliore la précision du modèle sans le modifier, c'est-à-dire en augmentant la quantité de données. Les données sont augmentées en manipulant l'ensemble de données d'origine en données synthétiques légèrement modifiées ou nouvellement créées. Cela augmente la généralisabilité du modèle et aide à réduire l'overfitting.

Dans Yolov4 [7], les auteurs utilisent une série de méthodes d'augmentation de données, dans lesquelles en changeant simplement la stratégie d'entraînement, la précision du modèle est garantie d'augmenter.

5.5.1.1 CutMix

CutMix [15] appartient à la famille des stratégies d'abandon régionales, où des patchs d'images sont supprimés et remplacés par des boîtes noires, ainsi que leurs vérités terrain respectives (*CutOut* [44] par exemple). *CutMix* [15] d'autre part remplace les patchs supprimés par une partie d'une autre image (voir figure 5.7). Cela signifie augmenter de manière significative la robustesse du modèle contre les corruptions d'entrée, car cela oblige le modèle à avoir une vue large de ce dont un objet peut être inclus.

	ResNet-50	Mixup [48]	Cutout [3]	CutMix
Image				
Label	Dog 1.0	Dog 0.5 Cat 0.5	Dog 1.0	Dog 0.6 Cat 0.4
ImageNet Cls (%)	76.3 (+0.0)	77.4 (+1.1)	77.1 (+0.8)	78.6 (+2.3)
ImageNet Loc (%)	46.3 (+0.0)	45.8 (-0.5)	46.7 (+0.4)	47.3 (+1.0)
Pascal VOC Det (mAP)	75.6 (+0.0)	73.9 (-1.7)	75.1 (-0.5)	76.7 (+1.1)

FIGURE 5.7 – Aperçu des résultats de Mixup, Cutout et CutMix [15].

5.5.1.2 Augmentation des données en Mosaic

Basé sur *CutMix*, les auteurs de Yolov4 [7] proposent une nouvelle méthode d'augmentation de données appelée *Mosaic*, où ils fusionnent quatre images d'entraînement, au lieu de deux comme dans CutMix [15] (voir figure 5.8). Ainsi, 4 contextes différents sont mélangés, ce qui amène le modèle à apprendre des objets en dehors de leur contexte normal.



FIGURE 5.8 – Mosaic représente une nouvelle méthode d'augmentation des données [7].

5.5.2 Méthode de régularisation

5.5.2.1 Introduction

Comme l'augmentation des données, la régularisation est un ensemble de techniques qui vise à éviter de suradapter (overfitting) le modèle de réseau de neurones. Une technique de régularisation très courante est le dropout [45] qui consiste essentiellement à désactiver des neurones arbitraires pendant l'entraînement.

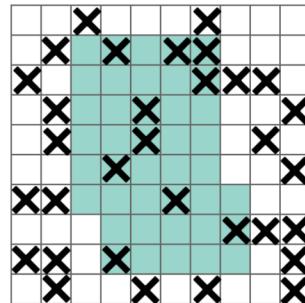
5.5.2.2 DropBlock

Dans CNN, appliquer dropout aux couches convolutionnelles ne sera pas suffisant. Parce que, comme nous le voyons dans la figure 5.9b, l'abandon des activations au hasard n'est pas efficace pour supprimer les informations sémantiques car les activations à proximité contiennent des informations étroitement liées. Au lieu de cela, la technique DropBlock[45] se concentre sur la suppression de régions continues (dans la figure 5.9c) afin de supprimer certaines informations sémantiques et ainsi forcer les unités restantes à apprendre des fonctionnalités qui aident le modèle à détecter des objets.

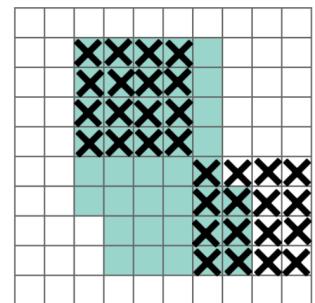
Selon les auteurs, DropBlock montre en fait de très bons résultats dans les réseaux CNN et est utilisé dans le backbone YOLOv4 [7] [45].



(a) Image d'entrée dans une couche de Conv.



(b) Traditional DropOut.



(c) DropBlock

FIGURE 5.9 – Les régions vertes dans (b) et (c) incluent les unités d'activation qui contiennent des informations sémantiques dans l'image d'entrée.

5.5.2.3 Lissage des étiquettes de classe

Le lissage d'étiquettes [46] a l'idée qu'au lieu de forcer le réseau à s'approcher de l'étiquette cible, le lissage d'étiquettes propose de convertir une étiquette dure en étiquette souple pour l'entraînement, par exemple, ajuste la limite supérieure cible de la prédiction, à une valeur inférieure à 0.9. Le réseau utilisera alors cette valeur au lieu de 1 pour calculer la perte. Ce qui peut faire en sorte que le modèle apprenne les données au lieu de les mémoriser, ce qui peut le rendre plus robuste.

5.5.3 Entraînement

5.5.3.1 Évaluation

Dans les modèles de détection d'objets, une seule image peut avoir de nombreux objets multiples, chaque objet appartenant à une classe différente, ce qui signifie que les techniques d'évaluation traditionnelles ne suffiront pas.

Par conséquent, l'évaluation des modèles de détection d'objets n'est pas simple comme c'est le cas dans les problèmes de régression ou de classification. Les modèles de détection d'objets utilisent ce qu'on appelle mAP (mean Average Precision) comme métrique pour mesurer les performances.

Définissons maintenant quelques notions importantes :

- **Vrais/faux positifs**

Les vrais positifs correspondent au nombre de fois où le modèle détecte correctement des objets, tandis que les faux positifs correspondent à la détection erronée d'objets.

- **Vrais/Faux négatifs**

Les vrais négatifs représentent le nombre d'objets correctement non détectés, tandis que les faux négatifs représentent le nombre d'objets faussement non détectés.

— Précision

La précision [47] est une métrique qui détermine l'exactitude des prédictions du modèle. La précision répond à la question suivante : de toutes les prédictions positives, combien d'entre elles étaient réellement positives ? Il est défini dans l'équation (5.1) comme suit :

$$\text{Précision} = \frac{\text{True_Positives}}{\text{True_Positives} + \text{False_Positives}} \quad (5.1)$$

— Rappel

Le rappel [47] est la fraction d'instances positives correctement prédites sur le nombre total d'instances positives. Il calcule le nombre de positifs réels que le modèle capture en l'étiquetant comme positif. Il est défini dans l'équation (5.2) comme suit :

$$\text{Rappel} = \frac{\text{True_Positives}}{\text{True_Positives} + \text{False_Negatives}} \quad (5.2)$$

— Précision moyenne (AP)

Comme nous l'avons vu dans la section 3.4.4, les modèles de détection d'objets génèrent le score de probabilité pour les classes prédites ainsi que leurs boîtes englobantes respectives, ce qui est un problème car nous ne pouvons pas utiliser les métriques traditionnelles. La solution consistait à utiliser *IoU* [35] Intersection Over Union (CIoU [35] dans Yolov4). *IoU* [35] trie les scores de probabilité de la sortie, puis considère une boîte englobante comme vrai positif, si elle est supérieure au seuil d'*IoU* (normalement 0,5), et comme faux positif dans le cas contraire.

— Mean Average Precision (mAP)

Nous avons vu ce qu'est AP (Average Precision) [48] et comment il est affecté par le seuil IoU [35]. Le score mAP (mean Average Precision) [48] est la moyenne de tous les AP qui satisfont à un seuil IoU [35] prédéfini.

5.5.4 Expériences

5.5.4.1 Introduction

Cette sous-section 5.5.4 vérifie l'exactitude et l'efficacité du modèle Yolov4 [7] à travers des expériences. Nous décrivons d'abord le processus d'entraînement de Yolov4 pour

apprendre à détecter les anomalies. Ensuite, nous décrivons en détail les résultats expérimentaux ainsi que la précision de la détection en considérant les vues à cinq angles du sujet.

5.5.4.2 Réglages et Hyperparamètres

Comme mentionné en sous-section 4.2.3, notre jeu de données se compose de 5155 images étiquetées. Nous créons un programme A.3 qui divise les données en parties : training, test, validation. Nous avons sélectionné au hasard 4755 pour les entraînements et 250 pour les tests. Nous avons également sélectionné 150 images pour validation où chacune des cinq images présente une scène sous les cinq angles (haut, gauche, droite, haut, arrière), ce qui signifie que les 150 images constituent 30 scènes. De plus les images ont été recadrées de [1920, 1080] à [1312, 1056], afin de se débarrasser des données inutiles (fond) et de réduire le temps de calcul.

En ce qui concerne les paramètres de configuration, nous avons mené trois expériences, toutes les trois ont été implémentées sur le framework darknet [9]. Le premier utilisait le GPU *Geforce RTX 2080*, les autres utilisaient le GPU *Tesla P100-PCIE-16GB* sur *Google Colab Pro* [49]. La raison de l'utilisation de *Goole Colab* est que *Geforce RTX 2080* a rencontré des problèmes de système et de mémoire lors de l'exécution de la deuxième expérience.

Les hyperparamètres des deux expériences se résument comme suit :

- **Taille de la résolution d'entrée :** détermine le nombre de pixels qui seront transmis au modèle pour apprendre et prédire. Une grande résolution de pixels améliore la précision, mais se compromet avec un entraînement et un temps d'inférence plus élevés. La résolution en pixels du modèle Yolov4 doit être un multiple de 32. Pour la première expérience, nous avons choisi la taille standard de 416×416 pixels, pour la deuxième et la troisième, nous avons choisi une taille de 608×608 pixels.
- **Batch size (taille de l'échantillon) :** c'est le nombre d'images choisies dans chaque batch pour réduire la perte. Pour les trois expériences, il était de 64.
- **Max batches (Max échantillons) :** le nombre d'itérations (batches) pour lesquelles l'entraînement sera réalisé. Il est calculé par la formule (5.3) suivant :

$$\text{max_batches} = (\text{nombre_classes}) \times 2000. \quad (5.3)$$

Dans notre ensemble de données, nous avons 9 classes donc $\text{max_batches} = 9 \times 2000 = 18000$.

- **Subdivisions :** nombre de mini échantillons (mini batches) dans un batch, ($\text{mini_batch} = \text{échantillon}/\text{subdivisions}$). Ainsi, le GPU traite les échantillons de mini batches à la

fois, et les poids seront mis à jour pour les échantillons de batches (1 itération traite les images par batches). Pour les expériences, la première est subdivisions = 16, la seconde est subdivisions = 32 et la 3ème est subdivisions = 64.

- **Optimiseur** : darknet [29] utilise *SGD* (Stochastic gradient descent) comme optimiseur.
- **Taux d'apprentissage (learning Rate)** : *learning rate* initial pour l'entraînement était de 0,001 pour les trois expériences.
- **Momentum** : accumulation de mouvement, combien l'histoire affecte le changement ultérieur de poids (pour le *SGD*), Momentum = 0.949 pour les trois expériences.
- **Decay** : une mise à jour plus faible des pondérations pour les caractéristiques typiques, elle aide à éliminer le déséquilibre dans l'ensemble de données. Decay = 0.0005 pour les trois expériences.
- **Steps** : steps = (80% de max_batches), (90% de max_batches) signifiant qu'à ces nombres d'itérations, le taux d'apprentissage sera multiplié par le facteur scales (scales = 0.01).
- **Classes** : tous les paramètres de classes doivent être retournés exactement le même nombre de classes des objets que nous entraînons. Ici le nombre de classes = 9.
- **Filters** : nous changeons [*filters* = 255] en filters = (classes + 5) × 3 dans les 3 [*convolutional*] avant chaque couche [*yolo*]. Donc dans notre cas, nombre de classes = 9, puis nombre de filters = 42.

5.5.4.3 Résultats

A. Première expérience

Dans la première expérience, il a fallu 38 heures pour terminer l'entraînement du modèle, la figure 5.10 montre la courbe de *mAP* [48] (mean average precision) et la fonction de perte. La courbe bleue est la perte d'apprentissage ou l'erreur sur le jeu de données d'apprentissage, elle est calculée à l'aide de *CIoU*[35] (Complete Intersection Over Union). Tandis que la ligne rouge représente la courbe de mean average precision à 50% (*mAP@0.5*) pendant l'entraînement sur le jeu de données de test. Nous voyons que la courbe bleue descend en douceur, ce qui signifie que le modèle continue de s'améliorer tant qu'il essaie toujours d'atteindre la perte la plus faible possible.

À partir de la courbe *mAP*, nous pouvons voir qu'après chaque 1000 itérations, la

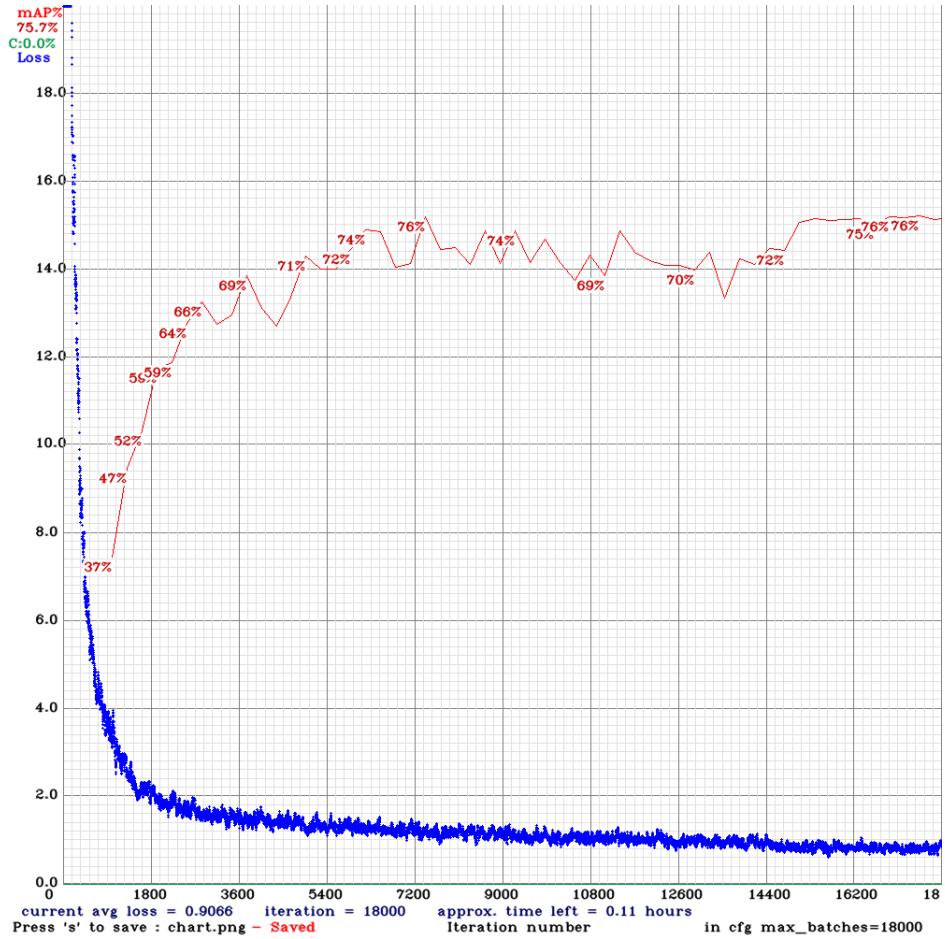


FIGURE 5.10 – mAP 0.5 et fonction de perte de la première expérience.

$mAP@0.5$ de l’ensemble de test a été calculée et dessinée. La valeur de mAP augmentait doucement au début, puis elle continuait d’osciller entre une précision de 70% à 80%. Nous pensons que le modèle peut encore s’améliorer si nous continuons à l’entraîner (plus grande valeur maximale des échantillons), car la perte continue de diminuer, mais craignant de tomber dans un surajustement, nous avons d’abord appris à tester et évaluer le modèle avec les hyperparamètres standards. Le tableau 5.1 montre le pourcentage de précision moyenne (AP) pour chaque classe, ainsi que la précision, le rappel et la mAP pour un seuil de 50%. Clairement, le modèle donne un AP exceptionnel supérieur à 90%, pour les classes *Objet*, *Extrudeuse*, *Fil* et *Brûlure*. En ce qui concerne *Spaghetti*, *Warping*, *Shift* et *Effondrement*, le modèle donne de bons résultats, un AP supérieur à la moyenne. La classe *Manque de matière*, en revanche, affiche un AP inférieur à la moyenne, ce qui est compréhensible car difficile à détecter (petits vides et remplissages).

Les mesures de *précision* et de *rappel* sont supérieures à 90%, cela indique une bonne courbe Précision × Rappel, ce qui signifie que même si nous faisons va-

Classes	Vrais positifs	Faux positifs	AP
Objet	229	8	98.74%
Extrudeuse	240	3	98.75%
Spaghetti	53	26	71.39%
Warping	10	3	77.05%
Shift	13	10	58.77%
Fil	110	2	98.87%
Effondrement	9	11	51.71%
Manque de matière	9	12	44.99%
Brûlure	3	0	100%
	Précision	Rappel	mAP
	90%	94%	76.05%

TABLE 5.1 – Première expérience : Résultats Yolov4 sur l’ensemble de test avec un seuil de 0.5.

riier le *seuil* (threshold), la *précision* et le *rappel* seront toujours élevés. Quant au mAP = 76.05% (bonne valeur), il stipule un modèle stable et cohérent.

Dans notre contexte, il est crucial pour nous de détecter les anomalies plutôt que de détecter les fausses anomalies, en d’autres termes, les *faux négatifs* (FN) ne sont pas acceptables pour nous, mais les *faux positifs* (FP) ne nous dérangent pas. Pour cette raison, nous avons choisi un seuil *mAP@0.25*, afin que notre modèle ait un *rappel* plus élevé. Le tableau 5.2 montre les résultats de la même expérience mais avec un seuil de 25%. Il est clair que le modèle s’est amélioré, *mAP* de 76,05% à 84,75% d’exactitude, de précision et de rappel reste le même, comme pour *AP*, une amélioration explicite a été observée pour chaque classe. Il est clair que le modèle a amélioré *mAP* de 76.05% à 84.75% d’exactitude, de *précision* et de *rappel* reste le même, comme pour *AP*, une amélioration explicite a été observée pour chaque classe.

B. Deuxième expérience

Dans la deuxième expérience, nous avons changé la taille de la résolution de l’image d’entrée du modèle, de 416 à 608, dans l’espérance d’améliorer la précision spécialement pour la classe Manque de matière. En raison de l’utilisation de Google Colab Pro, les performances d’utilisation du GPU sont bien inférieures à celles de la première expérience, ce qui rend le temps d’entraînement beaucoup plus élevé, en l’occurrence, environ 70 heures. Cependant, Google Colab a une évaluation meilleure et plus précise pendant le test *mAP*. D’après la figure 5.11, nous voyons le même résultat que la première expérience, la courbe de perte converge en douceur, tandis que la courbe *mAP* zigzague entre 70% et 80%.

Classes	Vrais positifs	Faux positifs	AP
Objet	215	23	99.35%
Extrudeuse	241	1	99.18%
Spaghetti	59	14	88.23%
Warping	15	0	100%
Shift	13	8	66.04%
Fil	110	1	98.82%
Effondrement	11	10	66.11%
Manque de matière	9	12	44.99%
Brûlure	3	0	100%
	Précision	Rappel	mAP
	90%	94%	84.75%

TABLE 5.2 – Première expérience : Résultats Yolov4 sur l’ensemble de test avec un seuil de 0.25.

Les tableaux 5.3, 5.4 montrent les mesures de précision dans la deuxième expérience, pour un seuil de 50% et 25% respectivement. Nous avons remarqué une grande amélioration de la précision de mAP de 76.05% à 94.65% pour $mAP@0.5$ et de 84.75% à 98.99% pour $mAP@0.25$. Nous avons utilisé le fichier de modèle contenant les pondérations `yolov4-custom_final.weights` au lieu de `yolov4-custom_last.weights` afin que le résultat mAP du tableau 5.3 soit supérieur au résultat de la figure 5.11. Les changements que nous avons apportés dans la deuxième expérience ont amélioré la précision de mAP de la classe *Manque de matière*, 70.45% pour $mAP@0.5$ et 97.28% pour $mAP@0.25$. De plus, les AP de chaque classe sont également grandement améliorés.

C. Troisième expérience

Dans la troisième expérience, nous avons conservé les hyperparamètres de la 2ème expérience et seulement modifié la valeur des subdivisions à 16 avec une taille d’image 608 pour l’entraînement pour voir si nous pouvions obtenir de meilleurs résultats sans surcharge de mémoire. Dans cette troisième expérience, il nous a fallu environ 56 heures pour terminer l’entraînement. D’après la figure 5.12 on voit que les résultats sont légèrement meilleurs que la deuxième expérience, la courbe mAP est aussi plus stable entre 73% et 80%, la courbe de perte converge en douceur.

Les tableaux 5.5 et 5.6 montrent les mesures de précision dans la troisième expérience, pour un seuil de 50% et 25% respectivement. Nous avons constaté que la précision de mAP est diminuée de 94.65% à 86,74% pour $mAP@0.5$, de 98.99% à 95.58% pour $mAP@0.25$. De plus, les changements que nous avons apportés dans

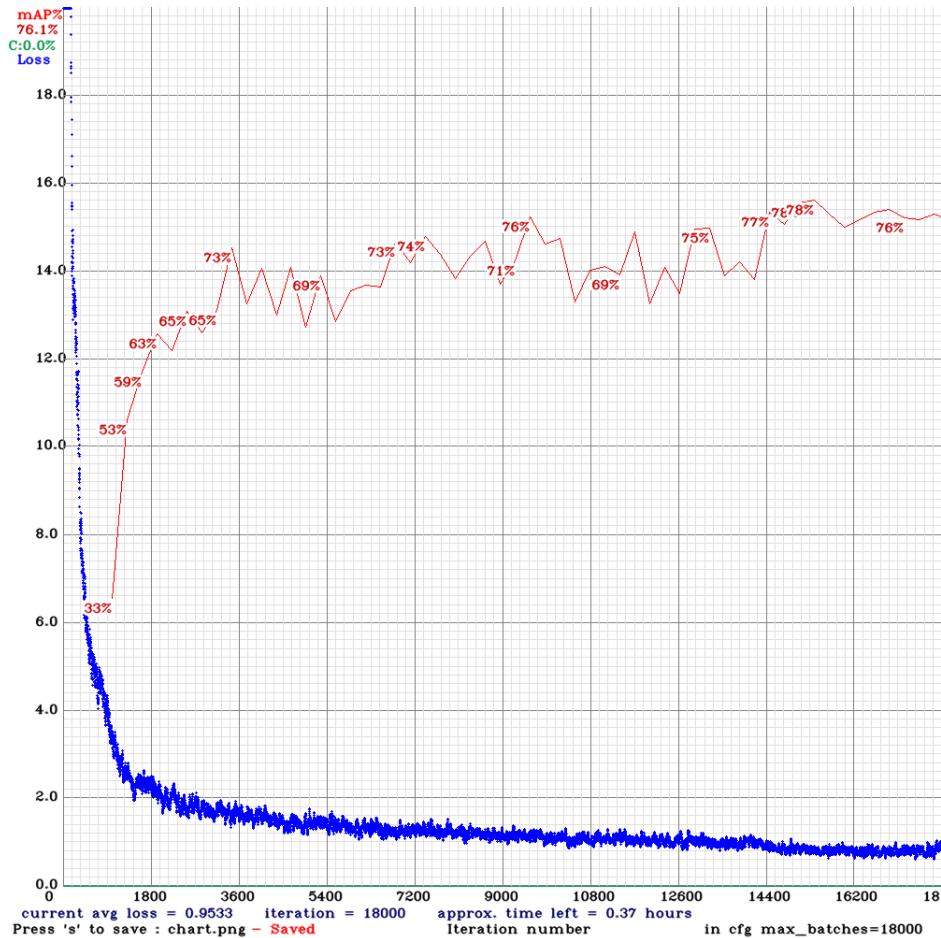


FIGURE 5.11 – mAP 0.5 et fonction de perte de la Deuxième expérience.

cette troisième expérience ont été de diminuer la précision AP de la classe *Manque de matière*, 63.65% pour $mAP@0.5$ et 74.16% pour $mAP@0.25$. Les AP des autres classes sont également légèrement réduits.

Discussions

À partir des expériences ci-dessus, nous avons obtenu le modèle Yolov4 dans la deuxième expérience avec les meilleures performances. Il atteint une mAP de 94.65%, avec une précision de 97% et un rappel de 99% à une IoU de 0.5, une mAP de 98.99%, avec une précision de 97% et un rappel de 98% à une IoU de 0.25. En augmentant le seuil IoU (0.75 par exemple), les prédictions sont obligées d'être plus strictes, ce qui entraîne les résultats des métriques seront réduites. Par conséquent, nous n'avons sélectionné que des seuils IoU de 0.5 et 0,25 dans les entraînements. Avec cela, l'utilisation d'un seuil IoU bas peut conduire à plus de *Faux Positifs* (précision inférieure), car le seuil d'acceptation du classificateur devient plus élevé. Les objets imprimés en 3D peuvent présenter de nombreuses

Classes	Vrais positifs	Faux positifs	AP
Objet	441	6	99.77%
Extrudeuse	467	1	100%
Spaghetti	103	8	97.01%
Warping	21	2	94.59%
Shift	30	3	100%
Fil	189	0	100%
Effondrement	26	0	95.71%
Manque de matière	27	17	70.45%
Brûlure	11	2	96.95%
	Précision	Rappel	mAP
	97%	99%	94.65%

TABLE 5.3 – Deuxième expérience : Résultats Yolov4 sur l’ensemble de test avec un seuil de 0.5.

Classes	Vrais positifs	Faux positifs	AP
Objet	429	18	100%
Extrudeuse	467	1	100%
Spaghetti	105	6	98.61%
Warping	21	2	98.94%
Shift	30	3	100%
Fil	189	0	100%
Effondrement	26	0	99.14%
Manque de matière	32	12	97.28%
Brûlure	11	2	96.95%
	Précision	Rappel	mAP
	97%	98%	98.99%

TABLE 5.4 – Deuxième expérience : Résultats Yolov4 sur l’ensemble de test avec un seuil de 0.25.

anomalies, et ces anomalies sont proches les unes des autres (ou se chevauchent). Par conséquent, la position de la zone de délimitation de prédiction doit coïncider avec la zone « ground truth »(vérité terrain) aussi haut que possible pour éviter le chevauchement d’autres objets. Cela signifie que plus IoU est élevé, mieux c’est. Ainsi, le seuil IoU de 0.5 est un seuil acceptable dans notre cas. Nous avons également constaté que des seuils d’ IoU entre 0.4 et 0.6 sont souvent choisis pour détecter des objets personnalisés similaires, par exemple dans [12] [50]. Un objet avec un taux de prédiction correct supérieur à 50% sera considéré comme un vrai

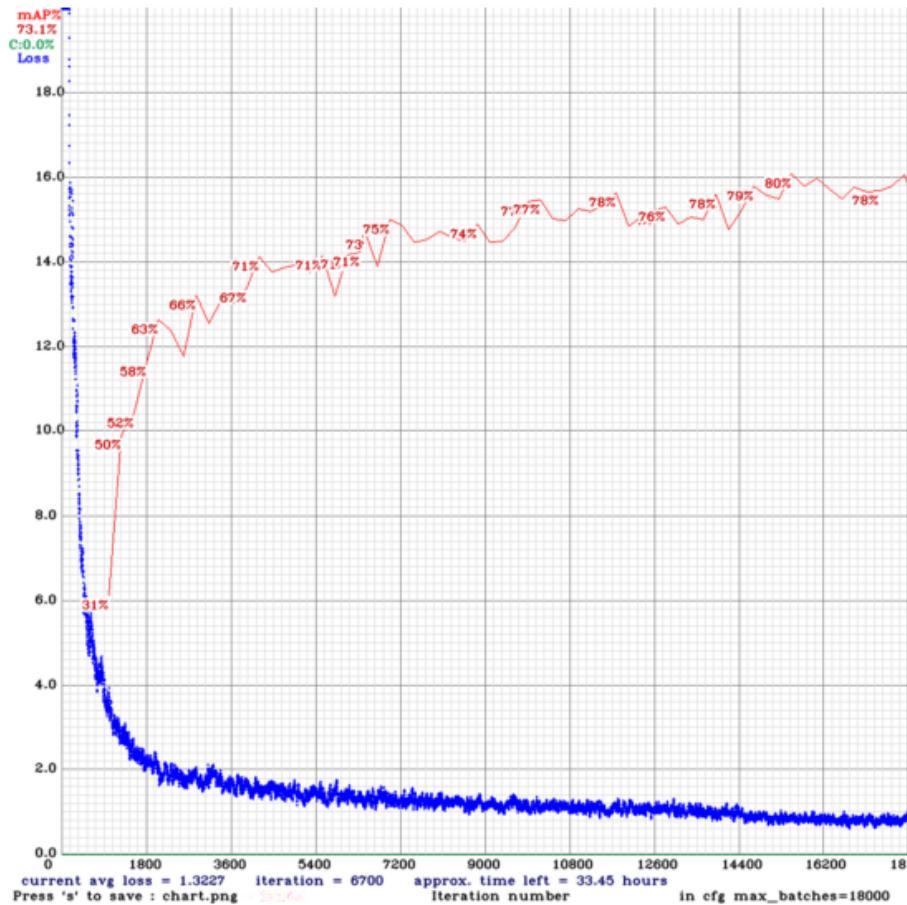


FIGURE 5.12 – mAP 0.5 et fonction de perte de la Troisième expérience.

positif et considéré comme une prédiction réussie. La figure 5.13 illustre l'utilisation du modèle yolov4 de la deuxième expérience pour créer une boîte englobante de prédiction pour les classes d'entités.

Avec des données d'entraînement pas beaucoup et pas assez riches (certains types d'anomalies sont en petit nombre) comme pour le moment, si on veut augmenter le nombre d'erreurs réellement détectées (True Positive), il faut garder le seuil IoU à ce minimum (0.5). Le choix d'une valeur pour le seuil IoU est une question d'équilibre entre *Vrais Positifs* et *Faux Négatifs*. Dans notre cas, nous pouvons même accepter de fausses alarmes (détection d'erreurs) plutôt que de ne pas être alarmés du tout. La précision du modèle peut être améliorée si nous pouvons équilibrer notre ensemble de données. La solution peut être de joindre une matrice de comparaison à l'image réelle pour marquer le modèle yolov4 où les anomalies sont souvent difficiles à détecter (anomalies que Yolov4 peut difficilement détecter si il ne connaît pas le modèle 3D à l'avance, par exemple Manque de matière et Shift).

Classes	Vrais positifs	Faux positifs	AP
Objet	429	25	98.62%
Extrudeuse	465	3	99.70%
Spaghetti	101	25	91.30%
Warping	21	4	87.26%
Shift	20	8	71.86%
Fil	188	1	99.17%
Effondrement	23	10	76.73%
Manque de matière	12	5	63.65%
Brûlure	13	5	92.35%
	Précision	Rappel	mAP
	94%	96%	86.74%

TABLE 5.5 – Troisième expérience : Résultats Yolov4 sur l’ensemble de test avec un seuil de 0.5.

Classes	Vrais positifs	Faux positifs	AP
Objet	415	39	99.87%
Extrudeuse	465	3	99.93%
Spaghetti	106	20	97.14%
Warping	23	2	98.30%
Shift	24	4	93.40%
Fil	189	0	100%
Effondrement	29	4	97.39%
Manque de matière	13	4	74.16%
Brûlure	14	4	100%
	Précision	Rappel	mAP
	94%	96%	95.58%

TABLE 5.6 – Troisième expérience : Résultats Yolov4 sur l’ensemble de test avec un seuil de 0.25.

5.5.4.4 Précision du modèle sous Cinq Angles

Après avoir trouvé un modèle Yolov4 approprié, nous voulions savoir à quel point le modèle serait précis si nous considérions les cinq angles de l’objet ensemble. Cette question vient naturellement, car les résultats que nous avons vus précédemment traitent chaque image indépendamment, ce qui n’est pas le cas dans notre contexte. Par conséquent, comme nous l’avons mentionné précédemment, nous avons sélectionné 30 scènes, chacune contenant cinq images, ce qui fait 150 images au total.

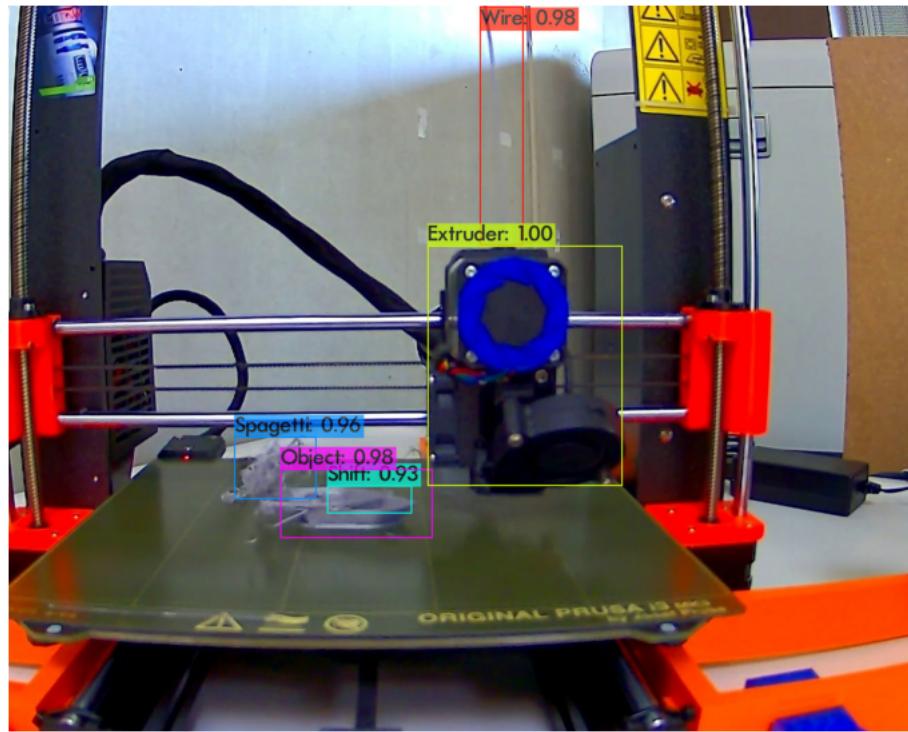


FIGURE 5.13 – Application du modèle Yolov4 à la prédiction des classes d’objets anormaux.

Maintenant, la question se pose : comment évaluerions-nous cette expérience ? Disons que dans une scène, l’objet contient des anomalies de Brûlure, à deux endroits différents, où l’angle supérieur les a détectés tous les deux, tandis que les angles arrière, droit et gauche n’étaient pas capables (les anomalies étaient invisibles), d’autre part, le l’angle avant n’a détecté qu’un seul anomalie Brûlure, car c’était le seul visible. Dans ce cas faut-il considérer les deux défauts ou un seul comme vérité terrain.

En fait, pour notre cas, nous ne nous soucions pas vraiment du nombre d’occurrences d’une anomalie dans une scène, notre objectif principal est de détecter les anomalies lorsqu’elles surviennent, pour arrêter l’impression. Pour cette raison, nous avons généré une nouvelle vérité terrain par scène, chacune ne contenant que les classes d’anomalies qui y sont présentes (pas de boîtes englobantes). En d’autres termes, nous évaluerons cette expérience comme s’il s’agissait d’un problème de classification. Là où nous avons choisi le modèle de la deuxième expérience avec un *IoU* de 0.5, il sortira des prédictions des 30 scènes, les classes de ces prédictions sont ajoutées à un ensemble. À la fin, chaque scène aura un ensemble contenant toutes les classes qui ont été prédites à partir des cinq images.

La tableau 5.7 présente les résultats de notre expérience. Les métriques de précision et de rappel sont supérieures à 90% pour presque toutes les classes (à l’exception des classes *Collapse* et *Manque de matière*). Ici, les classes *Shift*, *Effondrement* et *Manque de matière*

Classes	Vrais positifs	Faux positifs	Précision	Rappel
Objet	144	0	100%	100%
Extrudeuse	147	1	99.32%	100%
Spaghetti	21	0	100%	100%
Warping	14	0	100%	93.34%
Shift	3	0	100%	100%
Fil	64	0	100%	100%
Effondrement	5	2	71.42%	100%
Manque de matière	6	0	100%	85.71%
Brûlure	8	1	88.88%	100%
Moyenne			95.51%	97.67%

TABLE 5.7 – Métrique du modèle compte tenu des cinq angles.

n'ont pas suffisamment de données pour effectuer l'expérience, de sorte que leurs résultats ne sont pas complètement précis. Il semble que les cinq angles capturent la même quantité d'informations qu'un angle. La précision et le rappel moyens étaient de 95.51% et de 97.67% respectivement, indiquant la durabilité et la stabilité du modèle.

5.6 Problèmes rencontrés

L'un des problèmes rencontrés est l'environnement de configuration du ordinateur pour utiliser le GPU alloué à utiliser pour l'entraînement les réseaux neuronals. Après l'installation, j'ai rencontré des erreurs de mémoire et de système qui ont pris du temps à résoudre ces problèmes. Ces problèmes ont été résolus, mais ils ont réapparus lors des expériences sur Yolov4. Cela m'a pris un certain temps à résoudre et j'ai décidé d'utiliser le GPU sur Google Colab Pro pour entraîner le modèle.

De nombreux autres problèmes ont continué à survenir sur Google Colab. La durée d'exécution maximale de Google Colab par fois est d'environ 15 heures. Google Colab après 15 heures sera arrêté et déconnecté. J'ai trouvé une solution qui enregistrera le fichier de poids de Yolov4 toutes les 1000 itérations et continuera à l'entraîner à un autre moment. Cela signifie que le temps d'entraînement réel de chaque modèle Yolov4 est plus long que ce qui a été rapporté.

De plus, après avoir entraîné les premières expériences, je n'ai pas enregistré le graphique de perte et mAP toutes les 1000 itérations. Cela m'a pris plus de temps pour m'entraîner à partir du début.

5.7 Conclusion

Le travail et les expériences que nous avons effectués sur ce projet ont donné des résultats souhaitables : nous avons créé un programme qui peut traiter automatiquement des images réelles et virtuelles, nous avons également entraîné un modèle Yolov4 qui peut reconnaître avec précision la plupart des anomalies survenant lors de l'impression 3D, la précision du détecteur allant jusqu'à 95%.

Cependant, certaines anomalies sont plus difficiles à détecter car le modèle Yolov4 ne les reconnaît pas comme des anomalies ou comme des objets à imprimer. Pour ce faire, nous avons besoin d'un modèle de réseau de neurones *siamois* pour comparer automatiquement l'image de la caméra de l'objet en cours d'impression avec l'objet de l'image virtuelle dans le fichier GCode. À partir de là, le réseau *siamois* nous dira exactement où se produit la différence grâce à la matrice de comparaison. À partir de cette matrice de comparaison combinée à l'image en entrée du modèle, Yolov4 lui dira exactement où dans l'image il y a un problème et yolov4 n'a besoin que de s'y fier pour prédire l'anomalie. Nous avons fait beaucoup de recherches et testé de nombreux exemples différents à travers le réseau *siamois* avant de l'utiliser pour notre base de données, y compris les techniques d'apprentissage métrique à distance : *entropie croisée binaire*, la *perte de triplet* et la *perte contrastive*. Mais en raison du manque de données d'entrée, nous avons été interrompus dans l'utilisation du réseau *siamois* pour créer le modèle de comparaison. Nous avons ensuite utilisé le programme A.2 pour générer manuellement la matrice de comparaison (voir figure B.1), puis utilisé le programme A.4 pour étendre la taille et concaténer la matrice de comparaison avec l'image d'entrée (la matrice de comparaison sert de quatrième dimension de l'image). Étant donné que la création et l'exécution manuelles de l'imbrication nécessitent beaucoup de temps et que de nombreux problèmes surviennent lors de l'exécution des expériences, nous n'avons pas créé suffisamment de bases de données pour effectuer cette de l'entraînement sur le réseau Yolov4.

À l'avenir, nous pensons que si nous pouvons obtenir suffisamment de ressources pour entraîner le modèle de réseau *siamois* et le combiner avec le réseau Yolov4 que nous avons réalisé dans ce projet, nous pouvons créer un programme intelligent qui détecte automatiquement les anomalies dans le processus d'impression 3D complet sans aucune opération manuelle.

Conclusion

Actuellement, la demande d'impression 3D augmente, offrant des avantages dans la création de modèles aux structures complexes. Cependant, l'un des inconvénients qui surviennent dans le processus d'impression est la formation d'anomalies et de défauts. Cela consomme un temps et des ressources considérables. Ce projet de recherche met en évidence le potentiel et l'efficacité de l'utilisation de l'apprentissage en profondeur pour détecter les défauts imprimés en 3D en temps réel.

Plus précisément, nous avons pris en compte la synthèse des connaissances, en appliquant les concepts de modélisation des réseaux de neurones Siamose et Yolov4. Le modèle de réseau Siamose est utilisé pour comparer la similarité de deux images afin de générer une matrice de comparaison composée de patchs de cette image, tandis que le réseau Yolov4 est utilisé pour détecter les anomalies. Les deux types de modèles de réseaux neuronaux ont leurs propres caractéristiques et produisent des modèles appropriés qui ont été entraînés pour obtenir des résultats précis d'un ensemble de données d'entrée.

Ce rapport décrit ce que mon responsable de stage Cédric BOBENRIETH et moi avons envisagé ainsi que le travail que nous avons planifié et mis en œuvre (le diagramme de Gantt [B.3](#) joint en annexe B décrit l'avancement des tâches du projet que nous avons réalisées). Il y a des tâches où nous avons réussi et atteint notre objectif, à savoir : le prétraitement des données, les expériences Yolov4. Mais souvent, nous rencontrons des obstacles inattendus qui bouleversent tous les plans. Nous avons rencontré des problèmes tels que : le manque de données d'image réelles d'objets imprimés en 3D pour l'entraînement sur le modèle de réseau siamois, les lacunes dans l'étiquetage des objets dans les images, la combinaison de la matrice de comparaison avec les données d'image en entrée du réseau Yolov4, etc. Certains problèmes ont été résolus, d'autres non.

Malgré quelques difficultés, faire ce stage a été une expérience très enrichissante qui m'a permis de mettre en pratique mes connaissances des concepts d'apprentissage profond. Faire des expériences me permet aussi d'acquérir certaines connaissances et pratiques. De plus, c'est aussi un tremplin solide pour m'aider à acquérir de l'expérience dans le domaine de l'intelligence artificielle en général ainsi que de la vision par ordinateur en particulier.

Pour les travaux futurs, la prochaine étape pour nous est de terminer notre pipeline. Pour la partie "comparateur", nous allons collecter suffisamment de données pour s'entraîner

sur le modèle de réseau siamois et générer une matrice de comparaison, puis utiliser cette matrice comme entrée du réseau yolov4 pour la partie "détecteur". Le résultat final sera une application unique capable de capturer les images de cinq caméras, qui détectera alors automatiquement les anomalies de l'imprimante 3D en temps réel.

Bibliographie

- [1] Noël-Arnaud Maguis. *The Companion Rédigez des documents de qualité avec L^AT_EX - L'outil des professionnels pour publier mémoires, thèses rapports, articles scientifiques*. Addison-Wesley, Reading, Massachusetts, 1993.
- [2] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the Institute of Radio Engineers*, 86(11) :2278–2323, 1998.
- [3] Sukanya Dodke³ Akshata pingale¹, Ashwini Kanskar². Road feature extraction from high resolution satellite images using neural network. *International Research Journal of Engineering and Technology (IRJET)*, 2019.
- [4] Sounak Dey, Anjan Dutta, Juan Ignacio Toledo, Suman K. Ghosh, Josep Lladós, and Umapada Pal. Signet : Convolutional siamese network for writer independent offline signature verification. *CoRR*, abs/1707.02131, 2017.
- [5] Renu Khandelwal. One-shot learning with siamese network, 2021.
- [6] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet : A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015.
- [7] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4 : Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020.
- [8] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.
- [9] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. CspNet : A new backbone that can enhance learning capability of CNN. *CoRR*, abs/1911.11929, 2019.
- [10] Christine Dewi, Rung-Ching Chen, and Hui Yu. Weight analysis for various prohibitory sign detection and recognition using deep learning. *Multimedia Tools and Applications*, 79, 11 2020.
- [11] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. CBAM : convolutional block attention module. *CoRR*, abs/1807.06521, 2018.
- [12] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once : Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.
- [13] Dimitris Ziouzios, Nikolaos Baras, Vasileios Balafas, Minas Dasygenis, and Adam Stimoniaris. Intelligent and real-time detection and classification algorithm for recycled materials using convolutional neural networks. *Recycling*, 7 :9, 02 2022.

- [14] Tzutalin. Labelimg. Free Software : MIT License, 2015.
- [15] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix : Regularization strategy to train strong classifiers with localizable features. *CoRR*, abs/1905.04899, 2019.
- [16] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.
- [17] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014. cite arxiv :1406.2661.
- [18] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *CoRR*, abs/2003.05991, 2020.
- [19] Claude Touzet. *Les Reseaux de Neurones Artificiels, Introduction Au Connexionnisme*. EC2, 1992.
- [20] XingLong Liang and Jun Xu. Biased relu neural networks. *Neurocomputing*, 423 :71–79, 2021.
- [21] V.P. Plagianakos and M.N. Vrahatis. Training neural networks with threshold activation functions and constrained integer weights. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing : New Challenges and Perspectives for the New Millennium*, 2000.
- [22] Eric Nowak and Frédéric Jurie. Learning Visual Similarity Measures for Comparing Never Seen Objects. In *CPVR 2007 - IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, Minneapolis, United States, June 2007. IEEE Computer society.
- [23] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2) :91–110, November 2004.
- [24] Davide Chicco. *Siamese Neural Networks : An Overview*, pages 73–94. Springer US, New York, NY, 2021.
- [25] Wanxin Cui, Wei Zhan, Jingjing Yu, Chenfan Sun, and Yangyang Zhang. Face recognition via convolutional neural networks and siamese neural networks. In *2019 International Conference on Intelligent Computing, Automation and Systems (ICICAS)*, pages 746–750, 2019.
- [26] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet : A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.
- [27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
- [28] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

- [29] Joseph Redmon. Darknet : open source neural networks in c. 2013–2016. URL <http://pj-reddie.com/darknet>, 2016.
- [30] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4) :640–651, 2017.
- [31] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. *CoRR*, abs/1803.01534, 2018.
- [32] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN : towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.
- [33] Joseph Redmon and Ali Farhadi. Yolov3 : An incremental improvement. *CoRR*, abs/1804.02767, 2018.
- [34] Wendong Gai, Yakun Liu, Jing Zhang, and Gang Jing. An improved tiny yolov3 for real-time object detection. *Systems Science & Control Engineering*, 9 :314–321, 01 2021.
- [35] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distanceiou loss : Faster and better learning for bounding box regression. *CoRR*, abs/1911.08287, 2019.
- [36] Diganta Misra. Mish : A self regularized non-monotonic neural activation function. *CoRR*, abs/1908.08681, 2019.
- [37] Peter Sturm. Pinhole camera model. In Katsushi Ikeuchi, editor, *Computer Vision : A Reference Guide*, pages 610–613. Springer, 2014.
- [38] Alex Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *CoRR*, abs/1808.03314, 2018.
- [39] Mayavi : 3d scientific data visualization and plotting in python. <https://mayavi.readthedocs.io/en/latest/>. Accessed : 2022-08-03.
- [40] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [41] Camera Calibration calibrate single or stereo cameras and estimate camera intrinsics, extrinsics, and distortion parameters using pinhole and fisheye camera models. <https://www.mathworks.com/help/vision/camera-calibration.html>. Accessed : 2022-08-03.
- [42] Photoflare is inspired by the image editor currently only available on microsoft windows. <https://photoflare.io/>. Accessed : 2022-08-10.
- [43] Photofiltre est un logiciel de retouche d'images tres complet. <http://photofiltre.free.fr/frames.html>. Accessed : 2022-08-03.
- [44] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017.
- [45] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout : A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15 :1929–1958, 06 2014.
- [46] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

-
- [47] M. Hossin and M.N Sulaiman. A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process (IJDKP)*, 5(2) :1–11, November 2019.
 - [48] Adam Van Etten. Satellite imagery multiscale rapid detection with windowed networks. *CoRR*, abs/1809.09978, 2018.
 - [49] Colaboratory : colab notebooks allow you to combine executable code and rich text in a single document, along with images, html, latex and more. https://colab.research.google.com/?utm_source=scs-index. Accessed : 2022-08-03.
 - [50] Konstantinos Paraskevoudis, Panagiotis Karayannis, and Elias P. Koumoulos. Real-time 3d printing remote defect detection (stringing) with computer vision and artificial intelligence. *Processes*, 2020.

A

Programmes Python

Programme A.1 – Programme pour convertir un fichier GCode 3D en 5 côtés (gauche, droite, avant, arrière, haut) en temps réel.

```
import io
import os
import string
import math
import sys
from mayavi import mlab
from tvtk.api import tvtk
import numpy as np

file_g = 'bateaux'

document_Str = [ 'Front' , 'Back' , 'Top' , 'Right' , 'Left' ]

def create_dir( dir ):
    if not os.path.exists( dir ):
        os.makedirs( dir )
    print( "Created Directory : " , dir )
else :
    print( "Directory already existed : " , dir )
return dir

def make_document( document_Str ):
    create_dir( file_g )
    os.chdir( file_g )
    # Print the current working directory
    print( "Current working directory: {0}" .format( os.getcwd() ) )
    for i in document_Str:
        create_dir( i )
```

```

    retval = os.getcwd()
    print ("Current working directory %s" % retval)
    # parent directory
    parent = os.path.dirname(retval)
    print("Parent directory", parent)
    os.chdir(parent)
    print("Current working directory: {0}".format(os.getcwd()))

def Gcode2Image(filePath):
    e1 = []
    e2 = []
    e3 = []
    CurrentZValue = 0
    k = 0
    e = open('extruder.gcode', 'r')
    while True:
        flin = e.readline()
        if flin == '':
            break
        elif flin[0:2] == 'G1':
            array1 = flin.split()
            if 'Z' in flin[2:]:
                CurrentZValue = float(array1[1][1:])
            elif 'X' in flin[2:] and 'Y' in flin[2:] and 'E' in flin[2:]:
                e1.append(float(array1[1][1:])+30)
                e2.append(float(array1[2][1:])+15)
                e3.append(CurrentZValue)
        i = 0
        x = []
        y = []
        z = []
        CurrentZValue = 0
        f = open(filePath+'.gcode', 'r')
        x1, y1, z1 = (0, 230, -1.5) # | => pt1
        x2, y2, z2 = (250, 230, -1.5) # | => pt2
        x3, y3, z3 = (0, -20, -1.5) # | => pt3
        x4, y4, z4 = (250, -20, -1.5) # | => pt4

        mlab.figure(figure=1, bgcolor=(1, 1, 1),
                    fgcolor=None, engine=None, size=(656, 528))
        bed = mlab.mesh([[x1, x2], [x3, x4]], [[y1, y2], [y3, y4]], [
            [z1, z2], [z3, z4]], color=(0.7529, 0.7529, 0.7529))

```

```

img = tvtk.JPEGReader(file_name="bed_texture.jpg")
texture = tvtk.Texture(
    input_connection=img.output_port, interpolate=1, repeat=0)
bed.actor.actor.texture = texture
bed.actor.tcoord_generator_mode = 'plane'

while True:

    flin = f.readline()
    if flin == '':
        break
    elif flin[0:2] == 'G1':
        array1 = flin.split()
        if 'Z' in flin[2:]:
            CurrentZValue = float(array1[1][1:])

        elif 'X' in flin[2:] and 'Y' in flin[2:] and 'E' in flin[2:]:
            x.append(float(array1[1][1:]))
            y.append(float(array1[2][1:]))
            z.append(CurrentZValue)
        elif ';'LAYER_CHANGE' in flin and x:
            mlab.clf()
            mlab.figure(figure=1, bgcolor=(1, 1, 1),
                        fgcolor=None, engine=None, size=(656, 528))
            x1, y1, z1 = (0, 230+(105-y[-1]), -1.5) # | => pt1
            x2, y2, z2 = (250, 230+(105-y[-1]), -1.5) # | => pt2
            x3, y3, z3 = (0, -20+(105-y[-1]), -1.5) # | => pt3
            x4, y4, z4 = (250, -20+(105-y[-1]), -1.5) # | => pt4
            bed = mlab.mesh([[x1, x2], [x3, x4]], [[y1, y2], [y3, y4]],
                           [[z1, z2], [z3, z4]]], color=(0.7529, 0.7529,
                           0.7529))
            img = tvtk.JPEGReader(file_name="bed_texture.jpg")
            texture = tvtk.Texture(input_connection=img.output_port,
                                   interpolate=1, repeat=0)
            bed.actor.actor.texture = texture
            bed.actor.tcoord_generator_mode = 'plane'
            Te1 = [t+(125-x[-1]) for t in e1]
            Ty = [t+(105-y[-1]) for t in y]
            Te3 = [t+z[-1] for t in e3]

```

```

    mlab.plot3d(x, Ty, z, color=(0, 0, 0.8), line_width=2.0,
                 representation='wireframe')
    mlab.points3d(Te1, e2, Te3, color=(0.5, 0.5, 0.5))

    mlab.view(270.5, 75, 550, [105, 105, 0])
    print(i)
    mlab.savefig(filename=file_g+'Front/frame'+
    + str(i)+'.png', magnification=2)

    mlab.view(82, 85, 550, [105, 105, 0])
    mlab.savefig(filename=file_g+'Back/frame'+
    + str(i)+'.png', magnification=2)

    mlab.view(270, 12, 500, [105, 105, 0])
    mlab.savefig(filename=file_g+'Top/frame'+
    + str(i)+'.png', magnification=1)

    mlab.view(355, 82, 380, [105, 105, 0])
    mlab.savefig(filename=file_g+'Right/frame'+
    + str(i)+'.png', magnification=2)

    #mlab.view(195, 82, 372, [105, 105, 0])
    #mlab.savefig(filename=file_g+'Left/frame'+
    + str(i)+'.png', magnification=2)

    i += 1

# Camera angles
# Front mlab.view(268, 78, 543)
# Back mlab.view(80, 85, 200)
# Right mlab.view(350, 82, 380) mlab.yaw(10) mlab.pitch(5)
# top mlab.view(270, 20, 500)mlab.pitch(-10)
# left mlab.view(190, 85, 390) mlab.yaw(-8) mlab.pitch(9)

make_document(document_Str)
Gcode2Image(file_g)

```

Programme A.2 – Programme pour convertir un fichier GCode 3D en 5 côtés (gauche, droite, avant, arrière, haut) en temps réel et comparer des paires d’images pour créer une matrice de comparaison.

```

import glob
import os

```

```

import numpy as np
import pandas as pd
import cv2
import matplotlib.pyplot as plt
from PIL import Image

#NOTE: this script is to be called on images and label already
cropped via the "dataset_diagnostic" code

#Les parametres que l'on peut changer
#The image dimension must match the cropped dimensions
length_image = 1184
height_image = 864
size_bloc_pixel = 32 #la taille que chaque case de la matrice
correspond
yoloReel = "frame3755.txt"
imageTest="frame3755.jpg"
yoloVirtuel="frame230.txt"
imageVirtuel= "frame228.png"

class Bbox:
    def __init__(self, x1, y1, x2, y2):
        self.x1 = max(x1, x2)
        self.x2 = min(x1, x2)
        self.y1 = max(y1, y2)
        self.y2 = min(y1, y2)
        self.box = [self.x1, self.y1, self.x2, self.y2]
        self.width = abs(self.x1 - self.x2)
        self.height = abs(self.y1 - self.y2)

    @property
    def area(self):
        """
        Calculates the surface area. useful for IOU!
        """
        return (self.x2 - self.x1 + 1) * (self.y2 - self.y1 + 1)

    def intersect(self, bbox):
        #dx = min(axmax, bxmax) - max(axmin, bxmin)
        #dy = min(aymax, bymax) - max(aymin, bymin)
        #if (dx>=0) and (dy>=0):
        #    return dx*dy
        x1 = min(self.x1, bbox.x1)

```

```

#print("x1:", x1)
y1 = min( self.y1, bbox.y1)
#print("y1:", y1)
x2 = max( self.x2, bbox.x2)
#print("x2:", x2)
y2 = max( self.y2, bbox.y2)
#print("y2:", y2)
intersection = max(0, x1 - x2) * max(0, y1 - y2)
return intersection

def intersect_ratio(self, bbox):
    """
    Return the ratio of self that intersect bbox. I.E : if 'self' is inside bbox, should return 1
    """
    areaI = self.intersect(bbox)
    return areaI / self.area

def iou(self, bbox):
    intersection = self.intersection(bbox)
    iou = intersection / float(self.area + bbox.area -
                                intersection)
    # return the intersection over union value
    return iou

#Take as input :
#yoloPathVirtuel = path to .txt file containing the yolov4 annotation
#imReelPath = path to the reel image corresponding
#imVirtuelPath = path to the virtual image (from GCODE) corresponding
#to the reel image
#name = name of the folder in which save everything
#NOTE : ONE FOLDER BY IMAGE, DONT REUSE AN EXISTING FOLDER OR YOU'LL
#        OVERRIDE THE CONTENT
def createMatrix(yoloPathReel, imReelPath, imVirtuelPath, name):
    #fl = open(yoloPathVirtuel, 'r')
    #DataVirtuel = fl.readlines()
    #fl.close()
    fl = open(yoloPathReel, 'r')
    DataReel = fl.readlines()
    fl.close()
    dw= lenght_image
    dh = height_image
    matrice = np.zeros((int(height_image/size_bloc_pixel), int(

```

```

length_image / size_bloc_pixel)))
im=Image . open ( imReelPath )
imV=Image . open ( imVirtuelPath )
if not os . path . exists ( name ) :
    os . makedirs ( name )
    os . makedirs ( name + "/ reel / similar " )
    os . makedirs ( name + "/ reel / diff " )
    os . makedirs ( name + "/ virtuel / diff " )
    os . makedirs ( name + "/ virtuel / similar " )
pathToSaveReel=name + "/ reel "
pathToSaveVirtuel=name + "/ virtuel "
for dt in DataVirtuel :
    # Split string to float
    cl , x , y , w , h = map ( float , dt . split ( ' ' ) )
    l = int (( x - w / 2) * dw )
    r = int (( x + w / 2) * dw )
    t = int (( y - h / 2) * dh )
    b = int (( y + h / 2) * dh )
    if cl != 0 and cl != 1 and cl != 5 : #on ne note pas les objets ,
        extrudeur et wire
        if l < 0:
            l = 0
        if r > dw - 1:
            r = dw - 1
        if t < 0:
            t = 0
        if b > dh - 1:
            b = dh - 1

    # l , t = left top point coord ( x , y )
    # r , b = right bottom point ( x , y )
    print (" Classe :" + str ( cl ) + " de " + str ( l ) + "," + str ( t ) + " to "
           "+str ( r ) + "," + str ( b ) )

rectangleDetection = Bbox ( l , t , r , b )
for i in range ( 0 , height_image , size_bloc_pixel ) :
    for j in range ( 0 , length_image , size_bloc_pixel ) :
        b = Bbox ( j , i , j + size_bloc_pixel , i +
                   size_bloc_pixel )
        intersectionValue = b . intersect_ratio (
            rectangleDetection )
        matrice [ int ( i / size_bloc_pixel ) ] [ int ( j /
            size_bloc_pixel ) ] += intersectionValue

```

```

imCrop = imV.crop((j, i, j+size_bloc_pixel, i+
size_bloc_pixel))
if intersectionValue >=0.35 :
    imCrop.save(os.path.join(pathToSaveVirtuel ,"
diff",str(i)+"_" +str(j)+"_" +str(
intersectionValue)+".png"))
else :
    imCrop.save(os.path.join(pathToSaveVirtuel ,"
similar",str(i)+"_" +str(j)+"_" +str(
intersectionValue)+".png"))

for dt in DataReel:
    # Split string to float
    cl, x, y, w, h = map(float, dt.split(' '))
    l = int((x - w / 2) * dw)
    r = int((x + w / 2) * dw)
    t = int((y - h / 2) * dh)
    b = int((y + h / 2) * dh)
    if cl!=0 and cl!=1 and cl!=5 : #on ne note pas les objets ,
        extrudeur et wire
        if l < 0:
            l = 0
        if r > dw - 1:
            r = dw - 1
        if t < 0:
            t = 0
        if b > dh - 1:
            b = dh - 1
        print("Classe :" +str(cl)+" de " + str(l)+"," +str(t)+" to "
"+str(r)+"," +str(b))
        rectangleDetection = Bbox(l, t, r, b)
        for i in range(0,height_image,size_bloc_pixel):
            for j in range(0,length_image,size_bloc_pixel):
                b = Bbox(j, i, j+size_bloc_pixel, i+
size_bloc_pixel)
                intersectionValue= b.intersect_ratio(
                    rectangleDetection)
                matrice[int(i/size_bloc_pixel)][int(j/
size_bloc_pixel)]+=intersectionValue
                imCrop = im.crop((j, i, j+size_bloc_pixel, i+
size_bloc_pixel))
                if intersectionValue >=0.35 :
                    imCrop.save(os.path.join(pathToSaveReel ,"
diff",str(i)+"_" +str(j)+"_" +str(

```

```

                intersectionValue)+".png"))
        else :
            imCrop.save(os.path.join(pathToSaveReel ,
            similar",str(i)+"_"+str(j)+"_"+str(
            intersectionValue)+".png"))

df = pd.DataFrame(data=matrice)
df.to_csv(name+outfile.csv', sep=' ', header=False ,
float_format='%.2f', index=False)
print("Similarity Done\n On decoupe les images now \n")

for i in range(0,height_image ,size_bloc_pixel):
    for j in range(0,lenght_image ,size_bloc_pixel):
        b = Bbox(j , i , j+size_bloc_pixel , i+size_bloc_pixel)
        intersectionValue= matrice[int(i/size_bloc_pixel)][int(j/
        size_bloc_pixel)]
        imCrop = im.crop((j , i , j+size_bloc_pixel , i+
        size_bloc_pixel))
        imCrop2 = imV.crop((j , i , j+size_bloc_pixel , i+
        size_bloc_pixel))
        if intersectionValue >=0.35 :
            imCrop.save(os.path.join(pathToSaveReel , "diff",str(i)
            +"_"+str(j)+"_"+str(intersectionValue)+".png"))
            imCrop2.save(os.path.join(pathToSaveVirtuel , "diff",
            str(i)+"_"+str(j)+"_"+str(intersectionValue)+".png"))
        else :
            imCrop.save(os.path.join(pathToSaveReel , "similar",str(
            (i)+"_"+str(j)+"_"+str(intersectionValue)+".png")))
            imCrop2.save(os.path.join(pathToSaveVirtuel , "similar",
            str(i)+"_"+str(j)+"_"+str(intersectionValue)+".png"))

#check if a point is inside a Rectangle
def isInside(pointX,pointY,topLeftRectangleX,topLeftRectangleY ,
bottomRightRectangleX,bottomRightRectangleY):
    if pointX>=topLeftRectangleX and pointX<=bottomRightRectangleX
    and pointY >= topLeftRectangleY and pointY<=
    bottomRightRectangleY :
        return True;
    else :
        return False;

```

```

def visualization(imgPath ,yoloPathVirtuel):
    plt . figure( figsize=(14, 12) , dpi=80)
    img = cv2.imread(imgPath)
    dh, dw, _ = img . shape
    f1 = open(yoloPathVirtuel , 'r')
    data = f1 . readlines()
    f1 . close()

    for dt in data:
        # Split string to float
        c1 , x, y, w, h = map( float , dt . split( ' '))
        l = int((x - w / 2) * dw)
        r = int((x + w / 2) * dw)
        t = int((y - h / 2) * dh)
        b = int((y + h / 2) * dh)
        if c1!=0 and c1!=1 and c1!=5 :
            print(dt)
            if l < 0:
                l = 0
            if r > dw - 1:
                r = dw - 1
            if t < 0:
                t = 0
            if b > dh - 1:
                b = dh - 1
            cv2 . rectangle(img , (l , t) , (r , b) , (0 , 0 , 255) , 1)

        plt . imshow(img)
        plt . show()

visualization(imageTest ,yoloReel)
createMatrix(yoloReel ,imageTest ,imageVirtuel , "test2")
b = Bbox(151,832,351,869)
a = Bbox(128,832,160,864)
print(str(a . intersect(b)))
print(str(a . area))
print(str(a . intersect_ratio(b)))

```

Programme A.3 – Programme aide à diviser l’ensemble de données en dossiers d’entraînement, de test et de validation en tant qu’entrée pour le processus d’entraînement du réseau CNN.

```
import glob , os
```

```

# Current directory
current_dir = os.path.dirname(os.path.abspath(__file__))

print(current_dir)
current_dir = 'data/obj'

# Percentage of images to be used for the test set
percentage_test = 10;
# Create and/or truncate train.txt and test.txt
file_train = open('data/train.txt', 'w')
file_test = open('data/test.txt', 'w')

# Populate train.txt and test.txt
counter = 1
index_test = round(100 / percentage_test)
for pathAndFilename in glob.iglob(os.path.join(current_dir, "*.jpg")):
    :
        title, ext = os.path.splitext(os.path.basename(pathAndFilename))
        if counter == index_test:
            counter = 1
            file_test.write("data/obj/" + title + '.jpg' + '\n')
        else:
            file_train.write("data/obj/" + title + '.jpg' + '\n')
            counter = counter + 1

image_files = []
os.chdir(os.path.join("data", "valid"))
for filename in os.listdir(os.getcwd()):
    if filename.endswith(".jpg"):
        image_files.append("data/valid/" + filename)
os.chdir("../")
with open("valid.txt", "w") as outfile:
    for image in image_files:
        outfile.write(image)
        outfile.write("\n")
    outfile.close()
os.chdir("../")

```

Programme A.4 – programme pour concaténer une matrice de comparaison (un fichier csv) avec une image.

```

import glob
import os
import numpy as np

```

```

import pandas as pd
import cv2
import math

import matplotlib.pyplot as plt
from PIL import Image
from numpy import genfromtxt

image = "frame1453.jpg"
matrix= "outfile.csv"

width = 608
height = 608

def getConcatenation(image , matrix , width=608, height=608):
    df = genfromtxt(matrix , delimiter=' ')
    height_csv = len(df)
    width_csv = len(df[0])
    coeffHeight = height/height_csv
    coeffWidth = width/width_csv
    a = np.eye(height , width)
    for i in range(0,height) :
        for j in range(0,width) :
            line = math.trunc(i/coeffHeight)
            column = math.trunc(j/coeffWidth)
            a[i][j] = round(df[line][column],2)

#LOAD IMAGE
im=Image.open(image)
#RESIZE IMAGE
im=im.resize((width , height))
#CONVERT TO NUMPY ARRAY
pix = np.array(im)
#print(pix.shape)
#print(a.shape)
#CONCATENATE THE IMAGE WITH THE ARRAY
a = np.reshape(a,(width ,height ,1))
#print(a.shape)
pix = np.dstack((pix , a))
#print(pix.shape)
return pix

#EXAMPLE

```

```
monImageConcatene = getConcatenation(image ,matrix ,608 ,608)
```

B

Capture d'écran

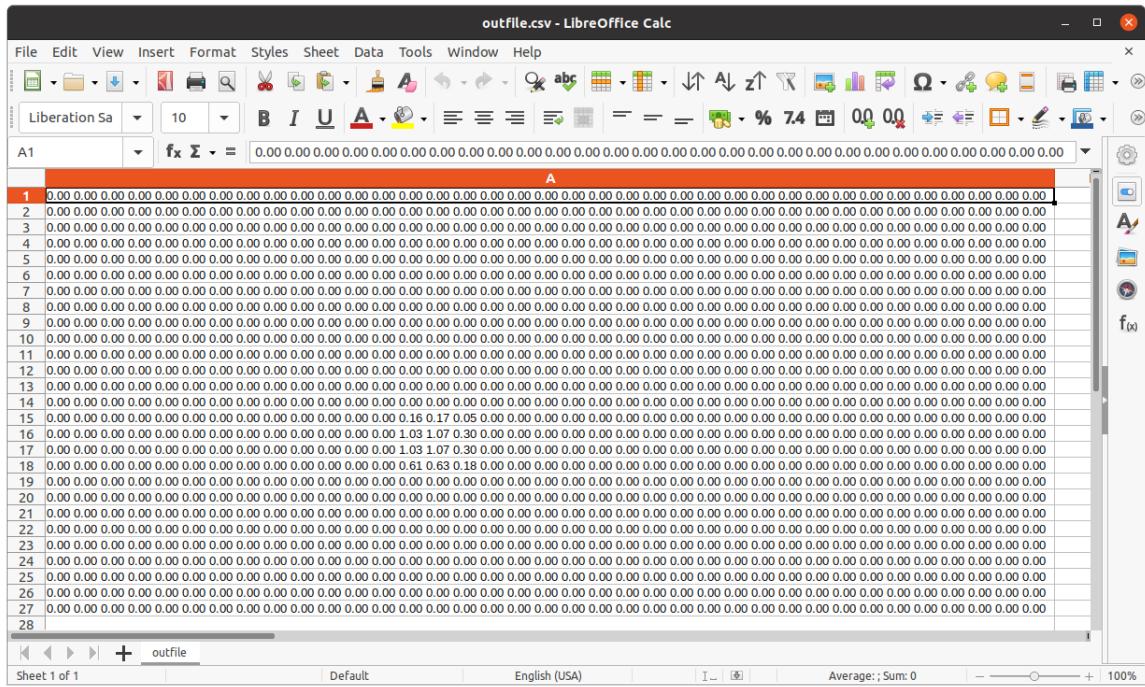


FIGURE B.1 – La matrice de comparaison est générée après l'utilisation du programme A.2.

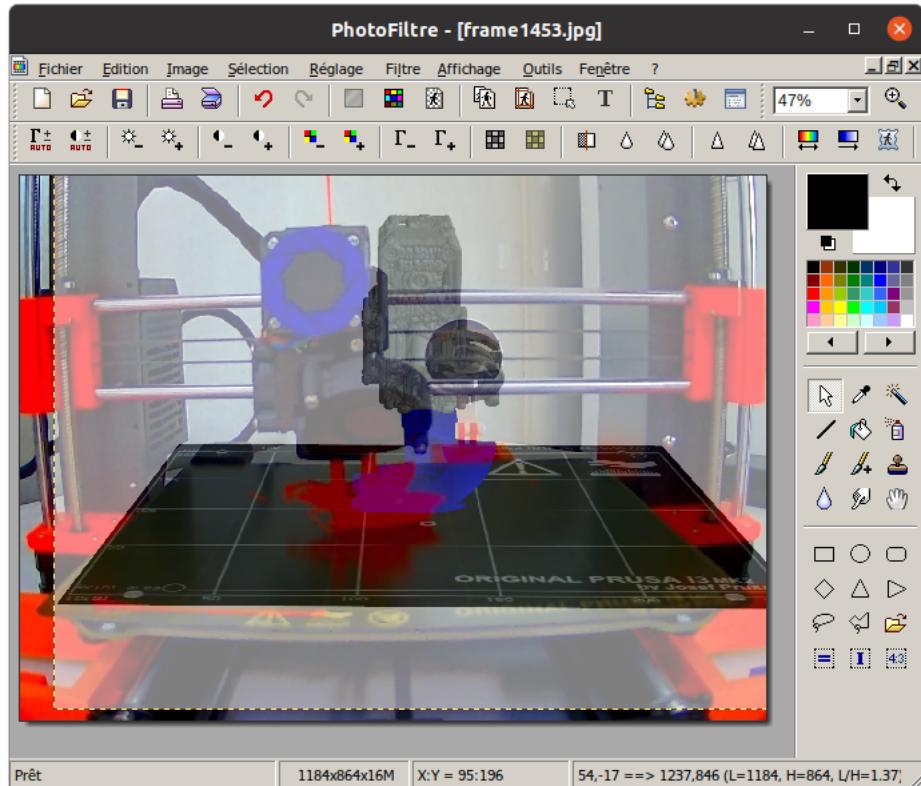


FIGURE B.2 – PhotoFiltre permet de recadrer et de comparer la position d'objet de 2 images.

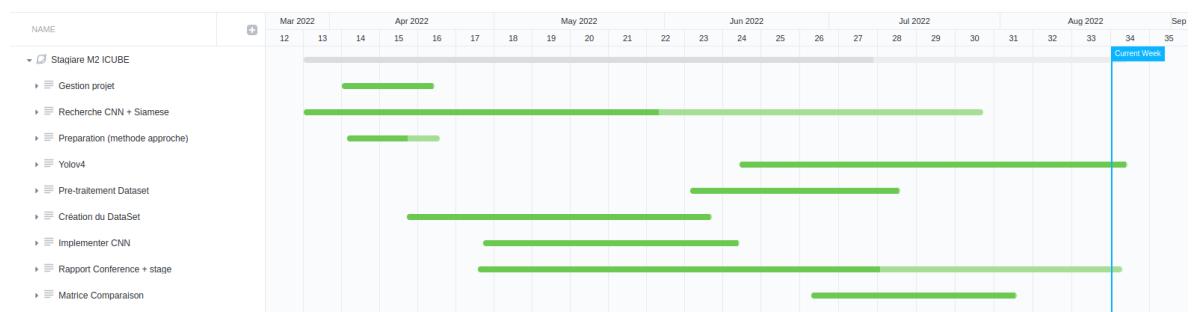


FIGURE B.3 – Le diagramme de Gantt représente l'avancement des tâches du projet que nous avons effectuées.