UNIVERSITÉ DU QUÉBEC À MONTRÉAL

RÉSEAU NEURONAL CONVOLUTIF AVEC DONNÉES D'APPRENTISSAGE AUGMENTÉES POUR DÉTECTER LES OBSTACLES DANS LES SYSTÈMES DE PILOTAGE DE TRAINS AUTONOMES

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

HOCINE KADDOUR DRIZI

SEPTEMBRE 2022

REMERCIEMENTS

En commençant par remercier tout d'abord Monsieur Mounir Boukadoum , directeur de recherche de ce mémoire, pour le soutien, la confiance et la disponibilité dont il a toujours fait preuve à mon égard.

Je remercie la direction de l'Université du Québec à Montréal et mes enseignants au cours de mes études de maîtrise en informatique pour la richesse et la qualité de leurs enseignements, ainsi que les efforts déployés pour me permettre de finaliser mon cursus malgré la crise difficile due à la pandémie de COVID-19.

Finalement, je tiens à remercier ma famille pour son soutien : ma mère, ma conjointe, ma petite sœur et tous mes proches et amis, qui m'ont accompagné, aidé, soutenu et encouragé tout au long de la réalisation de ce mémoire.

CONTENTS

LIST OF TABLES

LISTE DES FIGURES

ABSTRACT

Ce travail décrit un système d'apprentissage automatique pour la détection d'obstacles dans les systèmes de pilotage de trains, autonomes et autres. Nous présentons une architecture de réseau de neurones à convolution (CNN) qui s'appuie sur des modèles pré-entraînés et étudions plusieurs modèles afin de définir le plus apte à accomplir la tâche désirée, notamment en relation avec la taille relativement faible des ensembles d'entraînement disponibles pour le CNN.

La base d'images ferroviaires RailSem19 sert de référence initiale pour l'entraînement. Elle contient 8500 images de taille 1024 x1024 prises dans un environnement ferroviaire et un fichier JSON décrit chaque image en identifiant les objets qu'elle contient et leurs dimensions. Pour pallier la taille relativement modeste de RailSem19, des techniques d'augmentation de données sont utilisées pour créer de nouvelles images et obtenir un ensemble d'images de taille suffisante pour l'entraînement efficace du CNN développé. Plusieurs techniques sont utilisées à cette fin dont l'inversion de sens des image, le déplacement d'obstacles, l'ajout de bruit et les modifications climatiques et de luminosité.

Les expériences de validation avec plusieurs combinaisons de l'ensemble d'images initial et des images modifiées pour former des ensembles d'entraînement augmentés montrent que notre approche atteint un taux moyen de bonnes détections de 98.68%, avec une précision et un rappel excellents quand au biais et à la variance des détections.

Mots-clés : détection d'anomalies, CNN, Augmentation des données, détection automatique d'obstacles, systèmes de pilotage de trains.

INTRODUCTION

Mise en contexte :

La détection d'anomalies réfère au processus de trouver les valeurs aberrantes d'un ensemble de données. Depuis l'avènement de l'apprentissage automatique, la détection d'anomalies est devenue un sujet de recherche important dans plusieurs domaines comme la détection d'obstacles dans les systèmes de conduite de véhicules autonomes, la détection d'intrusions dans les réseaux de communications, le diagnostic médical, la détection de fraudes, la détection des défauts de fabrication et d'autres encore. Les domaines peuvent être classés selon le type des données nécessaires pour entraîner le modèle. Dans la plupart des cas, l'ensemble des données anormales représente un très petit pourcentage des données disponibles.

PROBLÉMATIQUE :

En raison de de la diversité et de la rareté des donnés traitées, Plusieurs défis sont à relever dans le domaine de la détection d'anomalies, dont l'ignorance (la ou les anomalies sont indéterminées jusqu'à ce qu'elles se produisent), l'hétérogénéité (l'existence de plusieurs anomalies) et la rareté des donnés (Pang *et al.*, 2020). On note les problèmes suivants en particulier :

- La difficulté d'obtenir un taux de rappel élevé pour la détection d'anomalies à cause de leurs rareté et hétérogénéité, de la difficulté à les identifier et la probabilité de confondre des instances normales et anormales.

- Le défi de détecter les anomalies dans des données de grande dimension et non

indépendantes, car elles sont déterminées dans des espaces de faible dimension.

- La difficulté d'un apprentissage efficace sur des données normales et anormales avec une petite quantité de données anormales, d'où la nécessité de pouvoir apprendre les représentations du normal et de l'anormal, et apprendre les modèles de détection pour détecter de nouvelles anomalies.

- Savoir apprendre des modèles de détection d'anomalies tolérants au bruit.

- Intégrer les concepts d'anomalies conditionnelles et les grouper dans les métriques du modèle de détection pour détecter les anomalies complexes.

Ce mémoire décrit une approche de solution basée sur une architecture de réseau de neurones artificiels avec données d'apprentissage augmentées.

CHAPTER I

CONTEXTE

1.1 introduction

La détection d'obstacle fait partie des problèmes de détection d'anomalies. Selon le type de données disponibles, plusieurs méthodes existent, chacune avec sa définition de comportement normal et comment l'utiliser pour la détection d'anomalies. Ce chapitre présente plusieurs méthodes avant de présenter des arguments en faveur de celle basées sur l'apprentissage en profondeur (Forward, 2020).

1.2 Approches de détection des anomalies

Les approches existantes peuvent être classées selon le type d'apprentissage utilisé et sa mise en oeuvre. On peut noter ce qui suit :

1.2.1 Méthodes à apprentissage supervisé

Dans l'apprentissage supervisé pour la détection d'anomalies, une machine apprend une fonction qui associe une entrée formée de caractéristiques à une sortie donnée sur la base d'exemples de pair d'entrées et de sorties. L'algorithme utilisé s'appuie donc sur une base de savoir prédéfinie pour son apprentissage (paires entrée-sortie). Par la suite, la machine ainsi entraînée peut prédire si des données

d'entrée non étiquetées sont anormales.

1.2.2 Méthodes à apprentissage non supervisé

Dans l'apprentissage non supervisé pour la détection d'anomalies, l'algorithme d'apprentissage n'a pas d'exemples d'entrées étiquetées pour les guider. Ils doit donc apprendre seul à étiqueter les entités d'entrée en instances normales et anormales, malgré la rareté relative des secondes. Cependant, Peut être trouvé par des méthodes d'apprentissage non supervisées de bruit considéré comme une anomalie.(Forward, 2020)

1.2.3 apprentissage semi-supervisé :

L'apprentissage semi-supervisé tente de trouver un compromis entre les apprentissages supervisé et non supervisé. Cette approche hybride utilise un grand ensemble de données non étiquetées et un petit ensemble de données étiquetées. En général, le grand ensemble correspond à des données normales et le petit à des données anormales. le modèle de détection est entraîné avec les données normales non étiquetées et évalué avec les données anormales étiquetées (Ruff *et al.*, 2019).

1.3 Métriques d'évaluation de modèles :

Comme déjà mentionné, la distribution des classes dans les applications de détection d'anomalies n'est pas uniforme. Pour de tels problèmes déséquilibrés, le modèle peut être exact pour classer les instances de la classe normale, et l'être moins ou pas du tout pour classer celles des données anormales. Par exemple, un ensemble d'entraînement pour la détection d'anomalies de 1000 images dont 950 images normales et 50 images anormales peut mener un classifieur à classer

toutes les images comme normales, ce qui donne un taux de bonnes classifications de 95 % en moyenne alors qu'aucune classe anormale ne l'a été. Ceci montre que l'utilisation du taux de bonnes classification (accuracy en anglais) peut être insuffisantes en soi pour évaluer les modèles de détection d'anomalies, et il faut aussi tenir compte du bais et de la variance du classifier. Partant du taux de classification des exemples anormaux comme normaux (faux positif ou FP) et celui de la classification des exemples normaux comme anormaux (faux négatifs ou FN), on peut s'appuyer sur les métriques de précision et de rappel, définies par :

$$precision = \frac{TP}{TP + FP} \tag{1.1}$$

$$rappel = \frac{TP}{TP + FN} \tag{1.2}$$

Dans ce cas, le taux de bonnes classifications renseignera sur la performance du classifieur toutes classes confondues, la précision renseignera sur sa capacité à ne pas se tromper, et le rappel renseignera sur sa capacité à identifier chaque classe correctement. La précision et le rappel sont souvent remplacés par leur moyenne géométrique, appelée score F, qui donne un meilleur aperçu de la performance du classifiuer pour les classes déséquilibrées :

$$scoreF = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{1.3}$$

1.4 L'apprentissage profond pour la détection d'anomalies

Les méthodes d'apprentissage en profondeur pour la détection d'anomalies reposent sur des réseaux de neurones artificiels et offrent des avantages dont la capacité innée de gérer des données de grande dimension et d'intégrer différents types de données sans avoir à modéliser les anomalies pour chaque variable.

Ces méthodes peuvent exploiter plusieurs variables et donner un bon résultat une fois spécifiés leur hyper-paramètres (nombre de couches, nombre d'unités par couche, etc.) et un apprentissage adéquat.

1.5 Difficultés relatives à la détection d'anomalies

Plusieurs aspects sont à considérer pour la construction d'un modèle de détection efficace. On peut citer :

1.5.1 Exemples contaminés :

Un petit sous-ensemble de données jugées normales est inclus dans l'ensemble de la classe normale alors qu'il appartient vraiment à la classe anormale, et la même chose peut être trouvée pour un ensemble de la classe anormale. De ce fait, l'ensemble d'apprentissage se trouve contaminé.

1.5.2 Supervision humaine

L'un des principaux problèmes des méthodes non supervisées et semi-supervisées est qu'elles génèrent un grand nombre de faux positifs et des coûts de main-d'œuvre associés aux révisions manuelles (Forward, 2020). Un objectif important des systèmes de détection d'anomalies est d'intégrer les résultats de l'examen humain (sous forme d'étiquettes) pour améliorer la qualité du modèle.

1.5.3 Définition des anomalies

Dans la plupart des applications de détection d'anomalies, les frontières entre le comportement normal et le comportement anormal sont pas définies avec précision et sont changées constamment.

1.5.4 Seuil de détection des anomalies

Généralement, l'entraînement d'un modèle de détection d'anomalies se termine avec des scores d'anomalie pour tout l'ensemble de données. Le groupe normal devrait obtenir un score faible et le groupe anormal un score élevé. Pour cela, il est nécessaire d'établir un seuil de détection pour la distinction entre les deux états. Ensuite, les données dont le score est supérieur (ou inférieur) au seuil seront considérées comme anormales.

Le seuil peut être optimisé pour tenir en compte des coûts des faux positifs et des faux positifs, Ou bien celui de l'évaluation du modèle sur des jeux de données anormaux.

1.5.5 Interprétabilité des méthodes

Les méthodes d'apprentissage en profondeur présentes pour la détection d'anomalies fonctionnent en boîte noire, et leurs résultats ne peuvent être fiables sans savoir comment ils ont été obtenus. Cela est particulièrement vrai pour les applications dans des domaines sensibles comme la médecine et la finance. Ce problème peut être géré de plusieurs façons, notamment en utilisant des modèles de substitution qui font l'approximation des systèmes concernés tout en étant explicables ou interprétables. Parmi eux, il y a LIME, SHAP, Eli5...

CHAPTER II

L'APPRENTISSAGE EN PROFONDEUR POUR LA DÉTECTION D'ANOMALIES

2.1 Introduction

Les réseaux de neurones profonds sont utilisés dans la plupart des applications de détection d'anomalies récentes, car ils n'ont généralement pas besoin de savoir les caractéristiques des données traitées. Ces méthodes d'apprentissage en profondeur comprennent les réseaux de neurones à convolutions (CNN), les auto-encodeurs et les réseaux génératifs antagonistes.

2.2 Réseau neuronal à convolutions

Les réseaux de neurones à convolutions (CNN) sont incontournables dans quasiment tout ce qui touche au traitement d'images; que ce soit pour la la classification, la détection ou la reconnaissance d'objets, les CNN ont prouvé leur efficacité. Leur popularité est due à leurs performances sur des données contenant des motifs invariants comme les images, pour lesquelles ils se sont avérés plus efficaces que d'autres algorithmes.

Le RNA traditionnels de type perceptron multi-couches sont gourmands en calculs. Par exemple, pour une image en couleur de taille (64x64), une taille relativement

petite pour une photo, que nous représentons numériquement et passons à l'entrée à un réseau entièrement connecté. Nous obtenons des calculs sur des tensors de l'ordre de:

$$(64^2 \times 3) \times (64^2 \times 3) \approx 15 \times 10^7$$

Malgré le progrès technologique et de la puissance de calcul des GPU modernes, cette dimension reste difficile à traiter. Les réseaux à convolutions apportent une solution à ce problème grâce à une méthode de partage de poids (« weight Sharing »). Plus précisément, les régions ou entités qui partagent les mêmes informations de domaine d'application sont détectées par le même sous-ensemble de paramètres. Pour ce faire, un ou plusieurs filtres balayent toute l'image, et les régions qui produisent le même résultat conduisent à ce que les motifs associés à l'apprentissage de ces régions sont détectés par les mêmes paramètres.

Pour appliquer avec succès ce principe aux images, le réseau doit détecter les structures ou caractéristiques similaires dans l'entrée. Le masque de convolution permet d'extraire les traits communs recherchés (« feature extraction »).

2.2.1 Convolution

La fonction de convolution peut se résumer en deux points. Le premier est la détection de motifs (patterns) invariants dans les images. En d'autres termes, afin de partager équitablement les poids entre des caractéristiques similaires, la convolution doit pouvoir détecter les similitudes de formes indépendamment de leur emplacement ou de leur variation.

La deuxième caractéristique de la convolution est l'interaction locale (« sparse interaction »). Le principe réside dans la capacité du réseau à détecter des objets

dans des régions locales indépendamment de la globalité de l'image. En d'autres termes, chaque valeur de sortie d'une couche dépend d'un petit nombre d'entrées, plutôt que de tout l'ensemble. En bref, la convolution tente de capturer l'essentiel de chaque entrée en regroupant les parties possibles séparément de l'image entière.

2.2.2 Cartes de traits

Dans les réseaux de neurones convolutifs, les convolutions sont appliquées dans le CNN via des filtres (matrices de poids de taille variable) pour créer des cartes de traits pouvant être utilisées ultérieurement. Les unités de couches cachées sont divisées en "cartes de traits", ces unités recherchent les mêmes caractéristiques et partagent des matrices de poids correpondant à chacune.

Les unités cachées d'une carte de traits sont uniques car elles sont connectées à différentes unités dans les couches inférieures. Par conséquent, pour la première couche cachée, les unités de la carte de traits seront connectées à différentes régions de l'image d'entrée.

En résumé, les couches cachées sont divisées en cartes de traits, où chaque unité recherche les mêmes caractéristiques mais à des emplacements différentes de l'image d'entrée.

Un exemple intuitif pour représenter ce mode de fonctionnement consiste à imaginer quatre cartes de traits, chacune avec un trait différent : verticale (V), horizontale (H), bissectrice gauche (G) et une autre inclinée à droite (D). On applique ensuite ces quatre cartes à une image représentant une étoile, si on détecte en les appliquant à l'image la présence ou l'absence d'un motif de recherche, dans notre cas la correspondance entre les traits trouvés et les traits de l'objet, on obtient la Valeur 1. En supposant une étoile se compose d'au moins 3 caractéristiques, alors nous obtenons une classification en étoile si :

$$H + V + D + G \geq 3$$

2.2.3 Création des cartes de traits:

La différence fondamentale entre les couches entièrement connectées et convolutives est que les couches entièrement connectées apprennent des modèles globaux dans leur espace d'entrée global, tandis que les couches convolutives apprennent des modèles locaux dans de petites fenêtres à deux dimensions.

On peut dire que le but principal d'une couche convolutive est de détecter des caractéristiques ou des éléments visuels dans une image, tels que des bords, des lignes, des couleurs, etc. En revanche, dans un réseau neuronal entièrement connecté, il doit réapprendre le motif s'il apparaît à un nouvel emplacement dans l'image.

Les convolutions sont appliquées dans le CNN via des filtres pour créer des cartes de traits qui peuvent être utilisées ultérieurement. Un filtre est représenté par une matrice de poids de taille variable qui est utilisée pour effectuer une opération matricielle (convolution) avec la matrice d'entrée pour remplir la case souhaitée.

Comme on peut le voir à la figure 1, la première case de la cartes de traits Z(1,1) est obtenue en multipliant la première case des entrées X(1,1) avec la première case du filtre K(1,1) et en les additionnant, nous déplaçons ensuite le filtre sur les points d'entrée à chaque fois pour obtenir notre carte. On généralise ce calcul avec la formule suivante :

$$Z_{i,j} = \sum_{m,n:1}^{f} X_{i+m-1,j+n-1} * K_{m,n}$$

$$\text{avec } X : n \times n \; et \; K : f \times f$$

En sachant que les indices commencent de la valeur 1 et que le filtre doit être

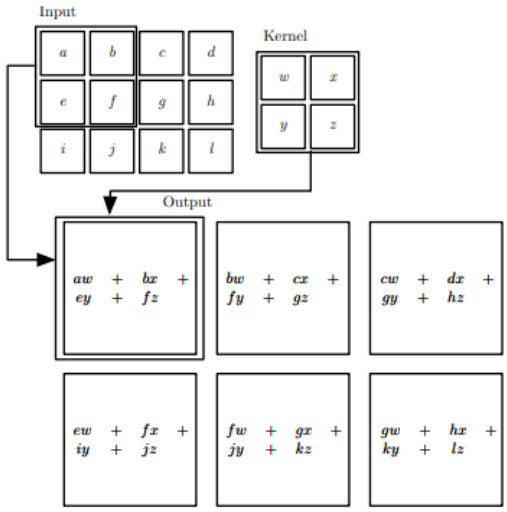

Figure 2.1: Exemple de convolution 2D (Boukaye Boubacar Traore, 2018)

remplacé pour que cette formule fonctionne. Une autre observation qu'on peut soulever est que les cartes de caractéristique sont d'une taille inférieure à celle de l'entrée originale. Cela est dû à l'application du filtre. Si on considère que l'entrée est de taille $(n \times n)$, le filtre est carré $(f \times f)$, la carte aura une taille.

$$(n - f + 1, n - f + 1).$$

À l'origine, les filtres étaient créés manuellement, mais il existe actuellement plusieurs types prédéfinis, nous pouvons donner quelques exemples :

- **Flou de Gauss (Gaussien Blur) :** Une convolution avec une distribution de poids spécifique faisant que plus le centre de l'image est proche, plus le poids est important, ce qui crée un effet de flou autour de l'image. Ces filtres jouent un rôle dans les méthodes traditionnelles de détection de formes. Dans l'apprentissage automatique, pour CNN, le filtre est appris automatiquement.

Figure 2.2: Convolution flou de Gauss (Sinha, 2022)

- **Détection de contours(Edge detection) :** Cette convolution est destinée à détecter les contours de l'image. un contour peut détecter par trois manière suivantes:

  – un changement brusque de l'intensité de l'image.

  – une différence sur la couleur de l'image.

  – les hautes fréquences du signal si on considère l'image comme étant un signal 2D.

  le Filtre de Sobel(Sobel Edge Operator) est populaire car elle est moins sensible au bruit que les autres convolutions de détection de contour

Figure 2.3: Détection de contours (Sinha, 2022)

Pour les filtres mobiles configurés en 2D, la convolution 2D est appelée de cette

manière. On peut aussi avoir une convolution 3D qui s'appliquera sur des entrées à 3 dimensions.

2.2.4 Champ de réception:

Lorsqu'il s'agit d'entrées de grande dimension comme des images, il n'est pas pratique de connecter des neurones à tous les neurones de la couche précédente. Au lieu de cela, nous connectons chaque neurone uniquement aux régions locales de la couche d'entrée. L'étendue spatiale de cette connectivité est un hyperparamètre appelé champ récepteur du neurone, qui est lié à la taille du filtre.

L'un des concepts les plus importants des réseaux de neurones convolutifs est le champ récepteur (Receptive field), qui permet de créer une sorte de traçabilité entre les cartes de traits. Si nous voulons connaître le point qui affecte l'activation d'un autre point, nous pouvons le faire à travers ce concept.

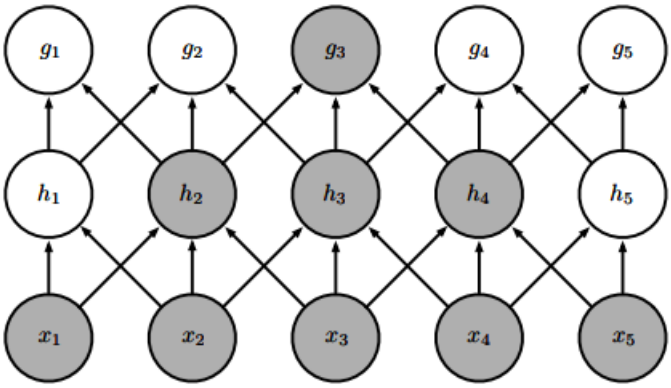

Figure 2.4: Exemple de champ de réception (Yu, 2019)

Une autre caractéristique du champ de réception est qu'il indique le chemin inverse du gradient.

Dans les réseaux profonds, des tâches telles que la classification ou la reconstruc-

tion sont appliquées à la sortie du dernier neurone. Par conséquent, il est important que ces neurones aient un grand champ récepteur (une grande zone d'entrée qui contribue à leur sortie) afin qu'ils obtiennent autant d'informations que possible de l'entrée.

2.2.5 Rembourrage (Padding)

Comme nous l'avons vu dans les sections précédentes, les cartes de traits sont un élément essentiel du bon fonctionnement d'un CNN. Cependant, elles présentent certains défauts qui limitent leurs résultats. En pratique, lors de la création d'une carte, certains points sont ignorés ou sous-représentés, ce qui peut fausser les résultats obtenus. Ces points sont généralement situés aux coins et aux bords de l'entrée. En raison de la manière dont les filtres sont appliqués, cela signifie que les champs de réception sont plus faibles et qu'ils sont moins impliqués par rapport au point central.

Pour résoudre ce problème, la méthode du padding est utilisée. Cette pratique comporte l'ajout de points ou de cases autour de l'entrée pour cibler des points auparavant sous-représentés, ce qui entraîne un engagement plus fréquent en augmentant le champ de réception (Figure 1.3).

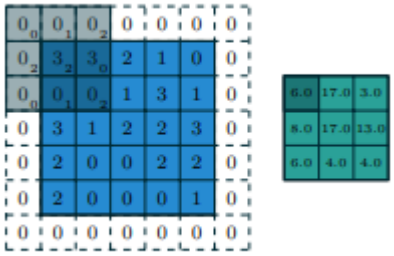

Figure 2.5: Exemple d'application des paddings (Dumoulin et Visin, 2016)

L'utilité des paddings n'est pas seulement d'augmenter le champ de réception, mais aussi de résoudre un autre problème qui peut survenir lors de l'application de la convolution, à savoir la réduction de la taille de la carte d'entités. En ajoutant des points supplémentaires, nous pouvons conserver la taille initiale et obtenir la carte dans les proportions souhaitées.

La formule ci-dessous nous donne la taille de la carte à générer (P est la taille du padding).

$$Z = (N + 2P - f + 1)$$

Une utilisation courante est de choisir la valeur de P de façon à conserver la même taille de l'entrée et ceci est expliqué par le calcul suivant :

$$N = N + 2P - f + 1$$
$$-2P = -f + 1$$
$$P = F + 1 \; P = \frac{f-1}{2}$$

Cette technique est appelée "Same Padding" ou "Half Padding" car la moitié du filtre est utilisée comme taille du padding ajouté.

2.2.6 Convolution par pas (Strided convolution)

Strided convolution est une autre technique utilisée pour affecter la taille obtenue par convolution. Lors de la création de la carte des traits, le filtre est déplacé pour laisser une différence d'un pas ($S = 1$) entre la position actuelle et la position précédente.

Si on augmente la taille du pas ou foulée (s), on peut modifier les dimensions obtenues par convolution.

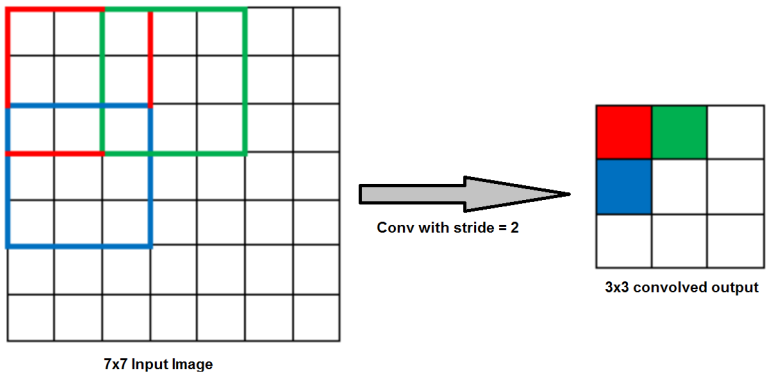

Figure 2.6: Application d'un pas Slide egale a 2 (Boukaye Boubacar Traore, 2018)

Pour calculer la taille d'une carte de traits (Z), sachant le pas (S) a été appliqué, on a le calcul suivant :

$$Z = (\lfloor \frac{N-F}{S} \rfloor + 1, \lfloor \frac{N-F}{S} \rfloor + 1)$$

En appliquant un filtre valide

On peut aussi généraliser le calcul en ajoutant le padding à l'équation :

$$(\frac{N+2P-F}{S} + 1, \frac{N+2P-F}{S} + 1)$$

$$\sum_{m,n}^{F} X \left\{ \begin{array}{l} S*(i-1)+1+m-1 \\ S*(j-1)+1+n-1 \end{array} \right\} * K_{m,n}$$

2.2.7 Agrégation (Pooling)

Le pooling peut être traduit en agrégation de points en vue de créer une représentation de l'ensemble. Il fonctionne comme une convolution en appliquant également un filtre d'une taille spécifique (FxF) et un pas de déplacement (S). Cependant, il

n'y a pas de paramètres ou de poids pour le pooling. En d'autres termes, le réseau n'a pas besoin d'apprendre de nouvelles variables pour appliquer le pooling.

Le pooling se présente sous de nombreuses formes et applications, notamment :

- **Maxpooling :** prend la valeur maximale de chaque région où le filtre est appliqué.
- **Average Pooling:** prend la moyenne des valeurs d'une région visitée par le filtre.
- **Norme L2 :** prend la norme L2 des valeurs (On peut aussi appliquer la norme L1).
- **Weighted average :** Les filtres utilisés pour le pooling sont paramétrés, c'est-à-dire qu'ils ont des poids. Nous Appliquons des opérations matricielles pour obtenir la moyenne générale.

Le Pooling est généralement utilisée pour réduire le volume spatial de l'image d'entrée après convolution. Le maxpooling est la variante la plus utilisée. Tel que décrit ci-dessus, le principe est d'appliquer un filtre qui prendre la valeur maximale des valeurs de données d'une région. Dans l'exemple ci-dessous, nous appliquons le maxpooling dans une seule tranche de profondeur avec un stride de 1x1. On peut observer une réduction de la taille de 5 x 5 pour l'entrée à 3 x 3 pour la sortie de la phase de pooling (Figure 2.7)

L'objectif principal du maxpooling est de rendre le réseau robuste, c'est-à-dire insensible aux perturbations que l'on peut trouver dans l'entrée comme les petites translations, car si nous avons le même patron dans 2 images, mais une petite différence de position, Après l'opération de pooling, les 2 patrons seront représentés par la même sortie. Le Maxpooling permet également de détecter

Figure 2.7: Exemple de maxpooling (Dumoulin et Visin, 2016)

plus de caractéristiques dans un voisinage donné, ce qui signifie qu'après le max-pooling, la même carte de traits peut être obtenue pour deux points dans des régions différentes avec de petites différences de position et de rotation, et faire une prédiction malgré les invariants.

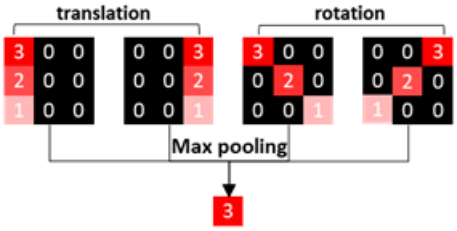

Figure 2.8: Exemple de l'effet du pooling aux invariants

2.2.8 Architecture et terminologie des CNN

Nous pouvons résumer la composition d'un réseau de neurones convolutif par un schéma simple (Figure 1.6). Le CNN consiste en une entrée qui est transmise à deux blocs constitués de convolution et de pooling, qui, après avoir appliqué les concepts vus dans les sections précédentes, convertissent le résultat en un vecteur compatible avec un réseau entièrement connecté. Ensuite, nous appliquons une fonction d'activation comme les fonctions sigmoïde ou softmax pour obtenir nos prédictions.

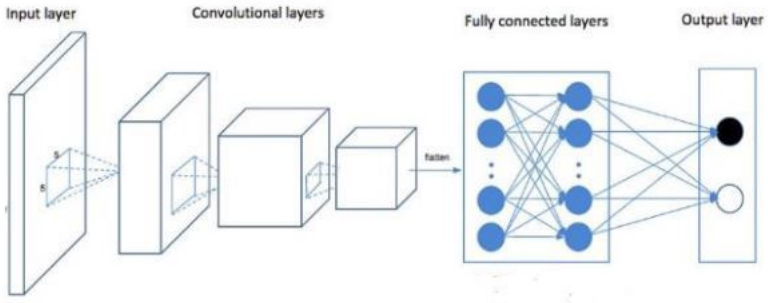

Figure 2.9: Architecture de la CNN(Mlyahilu *et al.*, 2019)

L'architecture des CNN est fortement inspirée du domaine neurocognitif et de la recherche de Fukushima.(Fukushima, 1988) sur le cerveau humain. Une convolution est une représentation de cellules simples et un pooling de cellules complexes. Les recherches nommées ont montré que cette hiérarchie existe dans le système visuel du chat. (Hubel et Wiesel, 1962)

Supposons que nous avons une entrée de 32 X 32 X 3 :

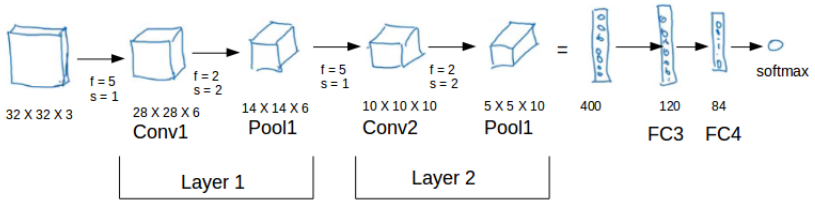

Figure 2.10: Exemple d'une architecture d'un CNN

Le traitement du CNN consiste en une séquences de couches de convolution et de pooling pour finir par des couches entièrement connectées avant un classificateur softmax pour classer l'entrée en différentes classes. traitement repose aussi sur des hyperparamètres importants tels que la taille, le déplacement du filtre et la méthode du pooling.

2.2.9 la Détection d'anomalies avec les CNN :

La détection d'anomalies à l'aide de CNN implique la formation d'un classificateur binaire composé de plusieurs couches convolutives pour apprendre les comportements normaux correspondant aux valeurs de la sortie 0 et apprendre les comportements anormaux correspondant aux valeurs de la sortie 1.

2.3 Autoencoders :

Les auto-encodeurs sont un type de réseau de neurones où l'entrée est la même que la sortie pour apprendre une représentation des données de faible dimension (compression de données), ils se composent de deux parties, la première partie est l'encodeur qui mappe l'entrée vers une faible dimension de représentation, et la deuxième partie est le décodeur, qui mapper la sortie de la partie codeur vers un résultat identique à l'entrée d'origine (l'entrée du codeur).

Pour construire un autoencodeur, on a besoins d'une méthode d'encodage, une méthode de décodage et fonction de perte pour calculer l'erreur de reconstruction qui est la différence (erreur quadratique moyenne) entre l'entrée d'origine et la sortie reconstruite produite par le décodeur.

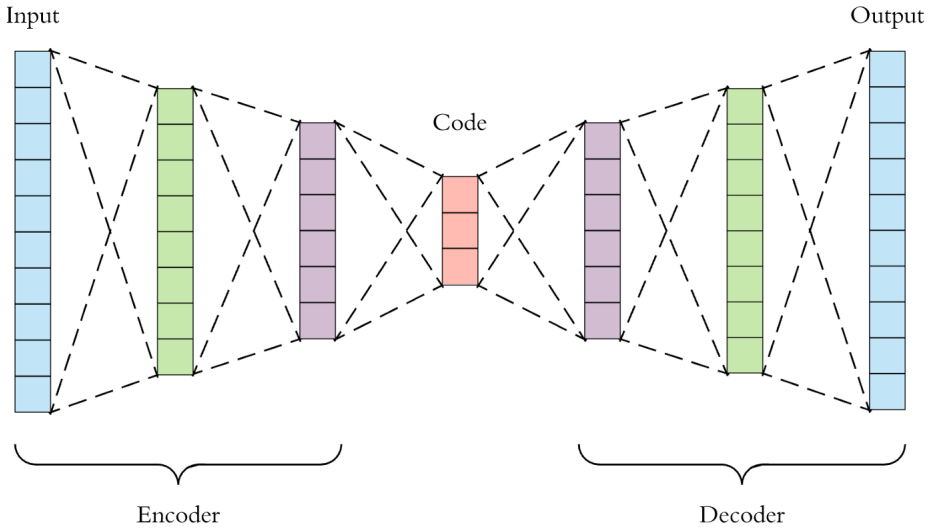

Figure 2.11: Architecture des Autoencodeurs (Dertat, 2017)

Les autoencodeurs sont souvent utilisés pour la réduction de dimensionnalité, l'extraction de caractéristiques non supervisées et la compression de données, la suppression du bruit d'image et la détection d'anomalies.

2.3.1 la Détection d'anomalies avec les auto-encodeurs

La détection d'anomalies avec des encodeurs automatiques implique la formation d'un modèle pour apprendre un comportement normal, puis la génération de scores d'anomalies pour celui-ci à chaque nouvel échantillon de données.

Les erreurs de reconstruction sont traitées comme des scores d'anomalies potentielles. Le seuil est ensuite calculé en sommant la moyenne et l'écart type des erreurs de reconstruction. Les erreurs de reconstruction au-dessus de ce seuil sont

considérées comme des anomalies (Forward, 2020).

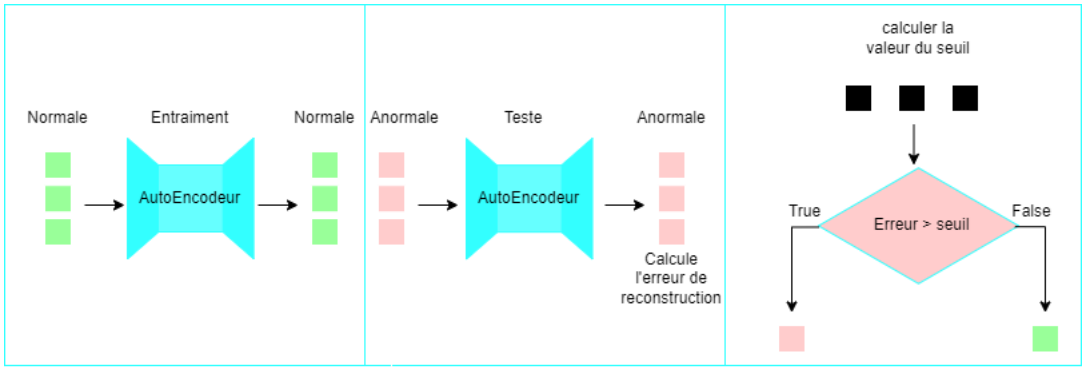

Figure 2.12: la Détection d'anomalies avec les Autoencodeurs

2.4 Les réseaux antagonistes génératifs :

Les réseaux antagonistes génératifs (GAN) (Goodfellow *et al.*, 2014) sont des architectures neuronales conçues pour apprendre un modèle génératif de la distribution des données d'entrée. Les GAN peuvent être utilisés dans différents domaines (image, texte, traitement du son, etc.) et comprenentd deux réseaux de neurones dont le premier, appelé le générateur, crée un échantillon de données, et le second, appelé le discriminateur, tente de détecter si cet échantillon est authentique ou la création de son générateur "adversaire". Les deux réseaux de neurones sont formés conjointement pour apprendre la distribution des données d'entrée. Lors des entraînements, les deux entités s'affrontent, ce qui leur permet d'améliorer leurs comportements respectifs.

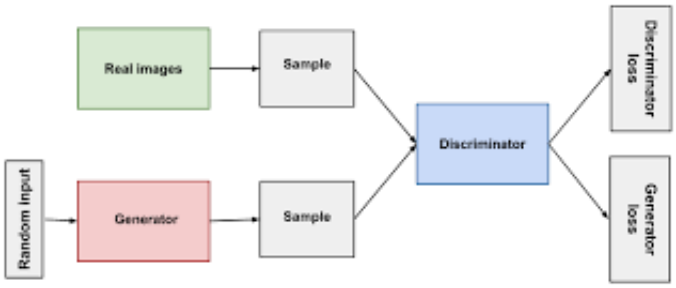

Figure 2.13: Architecture des GANs (Google Dev)

Le but du générateur est de générer des données qui ne peuvent pas être distinguées comme fausses par le discriminateur, et le but du discriminateur est d'identifier les faux. Ainsi, au fur et à mesure que le processus progresse, le générateur produit des données de meilleure qualité et le discriminateur détecte de mieux en mieux les contrefaçons, et cela jusqu'à ce qu'un équilibre soit établi tel qu'aucun du générateur ou du discriminateur ne peut faire mieux.

Les réseaux antagonistes génératifs apprennent à mapper des distributions latentes simples (choisies au hasard) à des distributions de données arbitrairement complexes afin que la meilleure puisse être trouvée pour représenter un ensemble d'échantillons fournis. Le calcul de cette dernière est intensif en calcul et ralentit le processus d'apprentissage.

Les Bigans peuvent résoudre ce problème, permettant un raisonnement contradictoire contrôlé en introduisant un réseau d'encodeurs qui apprendra le mappage inverse du générateur. C'est à dire. Il apprend à générer des données latentes à partir d'échantillons connus des donnés. Avec cette insertion, le discriminateur est également modifié. Il nécessite désormais une paire d'entrées composée d'une paire de données latentes (la première générée aléatoirement, la seconde générée par l'encodeur) et une seconde paire d'échantillons de données (le premier un échantillon vrai et le deuxième généré par le générateur).

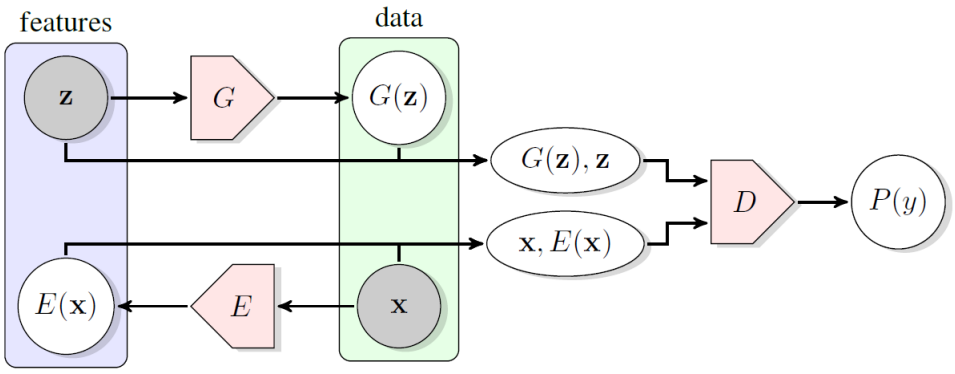

Figure 2.14: Architecture des BiGANs (Donahue *et al.*, 2016)

2.4.1 la Détection d'anomalies avec les BiGan :

La détection d'anomalies avec BiGan .(Akcay *et al.*, 2018) .(Mattia *et al.*, 2019) implique la formation d'un modèle pour apprendre un comportement normal et, au moment du test, de procéder comme suit pour générer des scores anormaux pour un échantillon donné .

Tout d'abord, nous obtenons la valeur latente des données X de l'encodeur et la transmettons au générateur pour obtenir les données X. Ensuite, nous pouvons calculer le score d'anomalie basé sur la perte de reconstruction (différence entre les données réelles et les données générées par le générateur) et la perte du discriminateur (la perte d'entropie croisée ou la différence de caractéristiques de la dernière couche dense du discriminateur) (Forward, 2020).

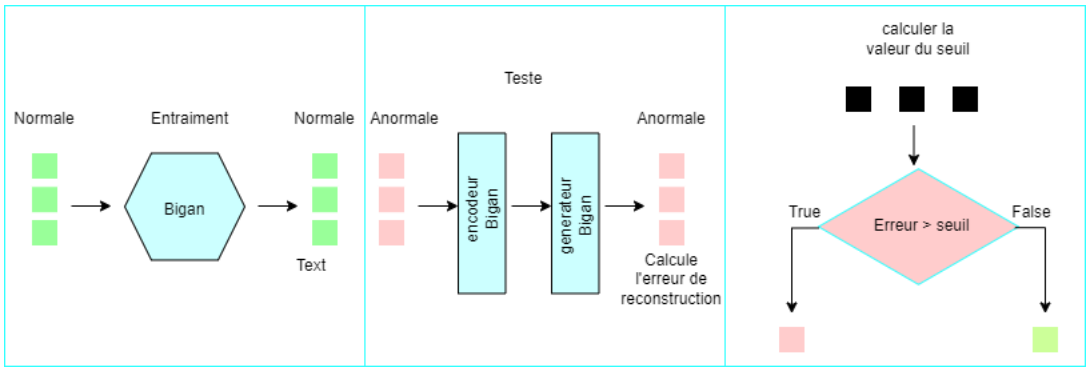

Figure 2.15: la Détection d'anomalies avec les BiGan (Donahue *et al.*, 2016)

2.5 conclusion

Dans ce chapitre, nous avons montré plusieurs méthodes d'apprentissage en profondeur pour la détection d'anomalies. Dans notre projet, nous utilisons une approche qui s'appuie sur les réseaux de neurones convolutifs (CNN) pour les raisons suivantes :

1. Il est relativement facile à mettre en oeuvre.
2. Il a déjà fait ses preuves en traitement d'image.
3. Il peut bénéficier du transfert de formation à partir d'un modèle pré-formé Inception-ResNet-v2, afin de pallier le peu de données d'entraînement.

Cependant, pour avoir un modèle de détection d'anomalies robuste, le réseau de neurones convolutif (CNN) requière une grande base de données d'images. Malheureusement, cette condition est difficile à satisfaire dans le domaine de la détection d'anomalies. Pour résoudre ce problème, nous introduisons au chapitre suivant des techniques pour augmenter les bibliothèques d'images afin d'augmenter la capacité des modèles de détection d'anomalies.

CHAPTER III

TECHNIQUES D'AUGMENTATION DE DONNÉES

3.1 Introduction

La plupart des Algorithmes d'apprentissage automatique d'aujourd'hui sont basés sur l'apprentissage en profondeur qui s'est révélé plus puissant que les méthodes traditionnelles, mais requière beaucoup de données d'apprentissage à cause du nombre de paramètres à régler. Cependant, pour les problèmes de détection d'anomalies, les données disponibles sont en petit nombre et avec des classes déséquilibrées en général. Pour résoudre ce problème, la création de données additionnelles à partir de celles disponibles est le meilleur moyen d'augmenter la taille de l'ensemble de données pour entraîner les modèles. Ce processus, appelé augmentation des données, est souvent essentiel pour entraîner les modèles de détection d'anomalies. La manière la plus simple de le mettre en oeuvre est par échantillonnage avec reprise des données existantes, créant ainsi des données redondantes. Cependant, le processus n'augmente pas la diversité des données et un risque de sur-apprentissage existe, faisant que le systèmes apprend à reconnaître les données de l'ensemble d'apprentissage sans pouvoir généraliser à des données non vues auparavant. De ce fait, d'autre techniques d'augmentation de données ont été proposées, notamment dans le cas des images.

Dans ce chapitre, nous présenterons certaines techniques d'augmentation de don-

nées couramment utilisées dans les modèles d'apprentissage en profondeur des images. On peut les diviser en deux catégories : Modification des images existantes et création d'images synthétiques. Dans le premier cas, on utilise surtout des transformations géométriques, de traits ou de texture, et dans le second, on génère des images virtuelles après avoir appris la distribution des images existante.

3.2 Modification des images existantes

3.2.1 Retournement

Il consiste à retourner l'image horizontalement en l'inversant par rapport à l'axe horizontal, ou la retourner verticalement en l'inversant par rapport à l'axe vertical. La figure 3.1 donne deux exemples de retournement d'image.

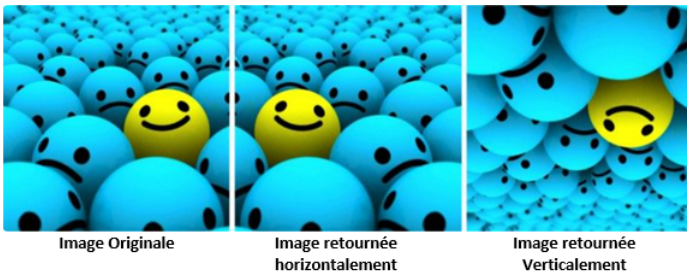

Figure 3.1: exemple de tourner une image (Gandhi, 2021)

3.2.2 Rotation :

Il s'agit de faire tourner l'image vers la droite ou vers la gauche d'un angle compris en 1 et 359°. Cette opération donne une image de dimension différent en général. La figure 3.2 donne deux exemples de rotation d'image.

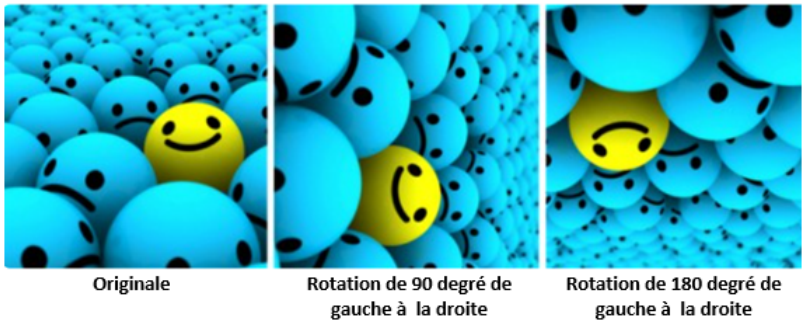

Figure 3.2: exemple de Rotation (Gandhi, 2021)

3.2.3 Recadrage

Il consiste à découper aléatoirement une partie de l'image et agrandir le résultat pour obtenir une image de mêmes dimensions. Cette opération est plus efficace avec les image de haute résolution afin d'éviter les effets de mosaïque dans l'image finale. La figure 3.3 donne deux exemples de recadrage d'image.

Figure 3.3: exemple de Recadrage (Gandhi, 2021)

3.2.4 Translation

Cela inclut le décalage de l'image à droite, à gauche, en haut ou en bas. Le but est de déplacer son contenant afin de créer une nouvelle perspective d'observation. La figure 3.4 donne deux exemples de translaltion d'image.

Figure 3.4: exemple sur la Traduction (Gandhi, 2021)

3.2.5 injection des bruit

L'injection de bruit consiste à ajouter aléatoirement aux pixels de l'image des valeurs aléatoires tirées généralement d'une distribution gaussienne. L'ajout de bruit aux images peut améliorer l'apprentissage. La figure 3.5 donne un exemple d'image bruitée.

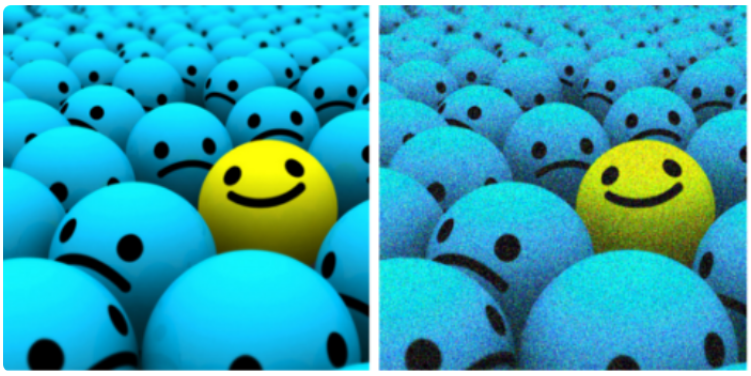

Figure 3.5: exemple d'injection des bruit (Gandhi, 2021)

3.2.6 Transformation de l'espace colorimétrique

La technique fournit plusieurs méthodes de transformation d'image. Les plus courants modifient la luminosité, le contraste ou l'inversion des couleurs d'une image. L'illustration ci-dessous donne un exemple.

Figure 3.6: exemple d'inversion de couleurs

3.2.7 Filtres de noyau

Il s'agit d'un moyen populaire en traitement d'images. Un filtre de noyau est une matrice qui rend une image floue ou plus nette. Pour la netteté de l'image, nous utilisons un filtre de bord et pour le flou de l'image, nous utilisons un filtre de flou gaussien. Les illustrations ci-dessous donnent des exemples.

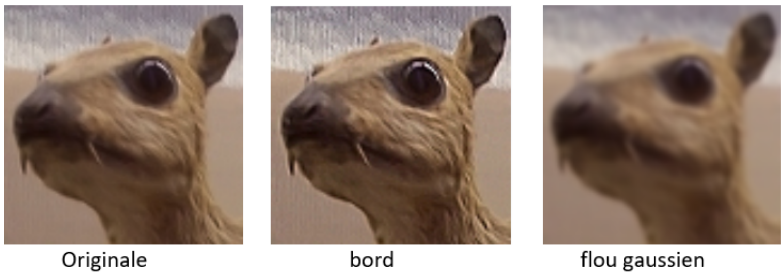

Figure 3.7: exemple sur les Filtres du noyau(wikipedia:Noyau (traitement d'image))

3.3 Augmentation par images de synthèse

Il s'agit de générer des images de synthèse après avoir appris la distribution sous-jacente de celles disponibles. Plusieurs approches existent, dont l'utilisation de modèles stochastiques comme les chaînes de Markov, et les réseaux de neurones artificiels. Parmi ces dernières, Celles à base de réseaux antagonistes génératifs

sont très populaires.

3.3.1 Augmentation des données basée sur le modèle GAN :

L'augmentation des données par les réseaux antagonistes génératifs (GAN) (Goodfellow *et al.*, 2014) ne nécessite pas de méthode d'augmentation prédéterminée, utilisant à la place un modèle neuronal qui apprend la distribution des données pour générer des exemples types. Noter cependant que les données ainsi générées n'apportent pas de nouvelle information à apprendre, puisqu'elles suivent la même distribution que les données d'entraînement.

3.3.2 Augmentation des données basée sur le CycleGAN :

CycleGAN (Zhu *et al.*, 2017) est une technique qui permet la traduction d'image à image. Contrairement au modèle GAN qui génère des images similaires aux données d'entraînement (images alignées), le modèle Cyclegan est entraîné de manière non supervisée avec des paires d'images non alignées. De ce fait, les images sources et les images cibles peuvent être très différentes.

Le modèle CycleGAN repose sur deux modèles GAN connectés tête-bêche. Il possède ainsi deux générateurs et deux discriminateurs, un générateur et un discriminateur pour chaque domaine de traduction. La figure 3.8 donne le diagramme d'un modèle CycleGAN.

Le modèle CycleGAN offre des résultats visuels impressionnants dans une vaste gamme de domaines d'application, y compris la détection d'anomalies en transformant des images normales en images anormales

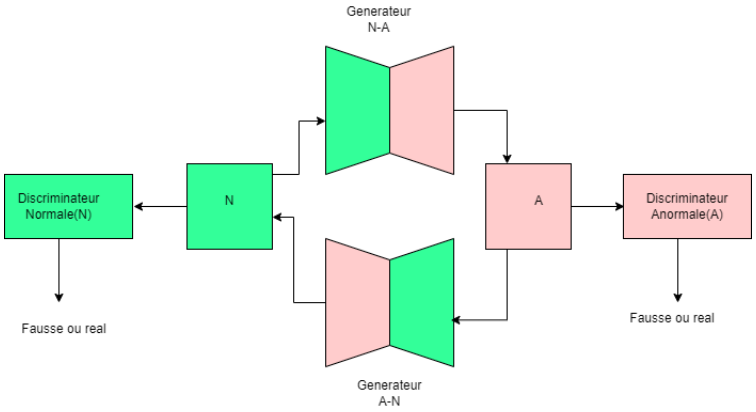

Figure 3.8: Architecture des CycleGANs

3.3.3 Augmentation des données basée sur l 'auto-encodeur variationnel :

L'auto-encodeur variationnel (VAE) (Kingma et Welling, 2013) hérite de l'architecture de l'auto-encodeur. Il se compose d'un encodeur et d'un décodeur et un espace latent, mais inclut une couche supplémentaire de moyenne et variance avant la couche d'espace latent pour forcer à regrouper chaque entrés à une distribution(groupe) et apprendre de reconstruire chaque entrée à partir de leur distribution et d'un espace latent.

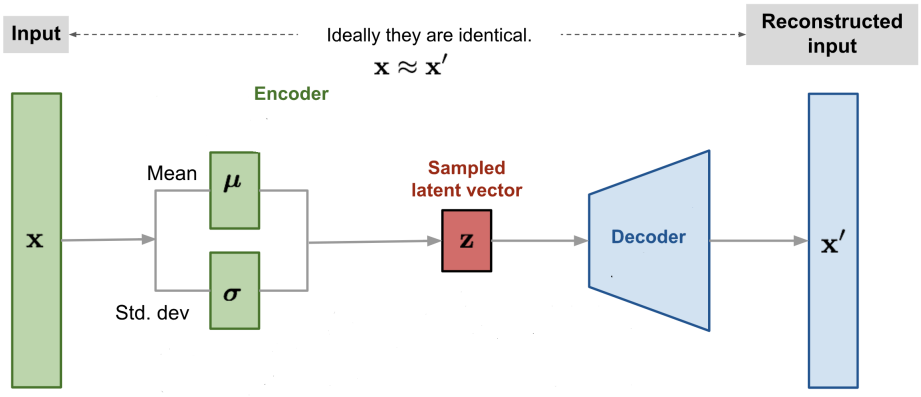

Figure 3.9: Architecture des auto-encodeur variationnel (wikipedia)

Après l'entraînement du modèle VAE, nous pouvons échantillonner un espace latent Z aléatoirement à partir d'une distribution normale(gaussienne) et l'envoyer à leur décodeur pour Générer de nouveaux données.

3.4 conclusion

Nous avons présenté plusieurs techniques d'augmentation de données (dans le cadre d'images) pour les modèles d'apprentissage en profondeur. Plusieurs de ces techniques sont utilisées dans ce travail pour augmenter notre base d'images et améliorer la performance de notre modèle comme décrit au chapitre suivant.

CHAPTER IV

MÉTHODOLOGIE ET EXPÉRIENCES

4.1 Introduction

Comme mentionné précédemment, ce travail vise à développer un modèle d'apprentissage en profondeur pour la détection automatique d'obstacles dans les systèmes de pilotage de trains. Le modèle choisi est un CNN et nous disposons d'un ensemble d'apprentissage constituée de 8500 images de base. À cause du le nombre important de paramètres à régler pour le CNN, cet ensemble s'est avéré de taille insuffisante pour un entraînement efficace du CNN et il a fallu l'augmenter.

Ce chapitre couvre l'architecture du CNN et sa configuration, l'augmentation des données d'apprentissage pour l'entraînement, et les expérience menées avec les résultats obtenus pour des images de test.

4.2 Réseau CNN

Il repose sur un modèle Inception-ResNet-v2 (Szegedy *et al.*, 2016) pré-entraîné. Il s'agit d'un réseau profond à 164 couches entraîné avec avec la base d'images ImageNets qui contient plus d'un million d'images pour classer les images en 1000 catégories d'objets (Deng *et al.*, 2009). La taille des images dans Imagenet est variable et une valeur commune de 299x299 est utilisée dans le modèle par

compression ou étirement. Nous partons donc d'un modèle qui a déjà appris à créer des représentation internes d'images à partir d'une grande bases d'images, et il s'agit de l'adapter spécifiquement à la reconnaissance d'obstacles dans les scènes ferroviaires.

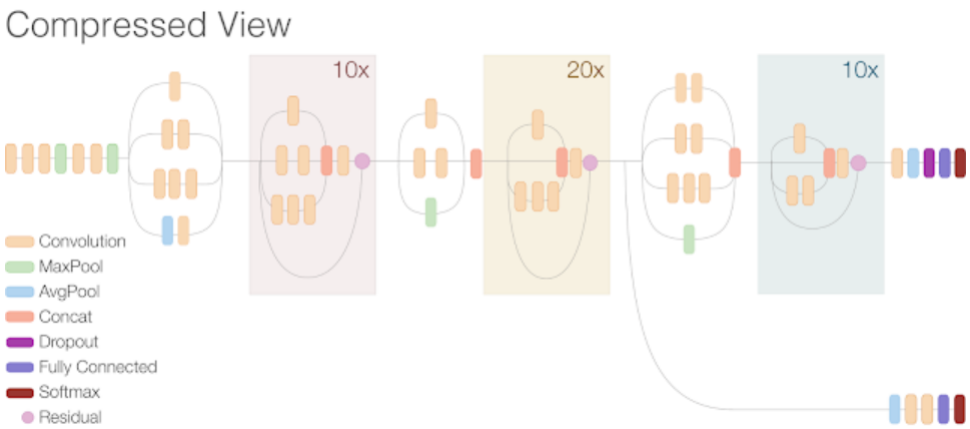

Figure 4.1: Schéma de principe de Inception-ResNet-v2(Google AI Blog)

4.2.1 Fonction de perte pour l'apprentissage

Nous utilisons la fonction de perte contrastive qui est populaire pour l'apprentissage binaire (Hadsell *et al.*, 2006). Il s'agit d'une fonction de perte métrique pour l'apprentissage des incorporations, généralement pour la validation des visages. Elle favorise les distances pour les paires d'entraînement positives et pénalise les distances pour les paires d'entraînement négatives. En d'autres termes, si les paires d'apprentissage positives sont détectées comme étant similaires et les paires d'apprentissage négatives sont détectées comme étant différentes, la perte est faible. La fonction est définie par :

$$LS(D, y) = (1 - y)\frac{1}{2}D^2 + y\frac{1}{2}\{max(0, (m - D))\}^2$$

où Y est notre étiquette binaire, D la distance euclidienne entre les images d'entrée

codées par neurones, et m un paramètre de marge positive généralement compris entre 0 et 1.

4.2.2 Fonction d'optimisation de l'apprentissage

L'optimiseur Ranger (Wright et Demeure, 2021) est utilisé. Il combine deux développements récents, RAdam et Lookahead, en un seul optimiseur d'apprentissage en profondeur pour obtenir le meilleur des deux mondes. Ainsi :

- **L'optimiseur RAdam :** «utilise un redresseur dynamique pour ajuster le momentum d'optimiseur Adam en fonction de la variance et fournit efficacement un échauffement automatisé, personnalisé et adapté à l'ensemble de données actuel pour assurer un bon début de formation.»(Wright et Demeure, 2021)
- **L'optimiseur LookAhead :**«réduit le besoin d'un réglage approfondi des hyperparamètres tout en réalisant "une convergence plus rapide entre différentes tâches d'apprentissage en profondeur avec une surcharge de calcul minimale.»(Wright et Demeure, 2021)

```
radam = tfa.optimizers.RectifiedAdam(lr=1e-5)
ranger = tfa.optimizers.Lookahead(radam, sync_period=6,slow_step_size=0.5)
```

Figure 4.2: l'optimiseur Ranger

- Le paramètre lamda de la fonction de perte contrastive et binaire pondérées a été fixé à une valeur de 0,3.

- Puisque nous avons utilisé le modèle pré-entraîné Inception-ResNet-v2 sur ImageNet, nous avons choisi une taille d'image de 299 x 299 pour le modèle.

- Pour obtenir une bonne précision du test, nous utilisons une méthode de régularisation :

- Weight decay (L2, L1)
- Dropout
- BatchNormalization

4.2.3 Régularisation des poids (weight decay) L1 et L2 :

Les régularisations de poids L1 et L2 sont les types de régularisation les plus courants. Celles-ci mettent à jour la fonction de coût générale en ajoutant un autre terme appelé terme de régularisation.

Fonction de coût = perte + terme de régularisation.

Du fait de l'ajout de ce terme de régularisation, les poids qui composent la matrice seront plus petits car la fonction de coût favorisera ces solutions. Espérons qu'en biaisant vers une certaine région de l'espace des solutions, le sur-apprentissage peut également être largement réduit.

Cependant, le terme de régularisation est différent en L1 et L2. Cette différence concerne la manière dont nous agrégeons les poids lorsque nous essayons de punir leur ensemble.

Dans L2 (également connu sous le nom de régression d'arête en statistique), la perte J est fonction d'un ensemble de paramètres du réseau, $\tilde{J}(\theta)$ devient :

$\tilde{J}(\theta) = J(\theta) + \frac{\lambda}{2m}\|\theta\|_2^2$

Ici, λ est le paramètre de régularisation. C'est un hyper-paramètre dont la valeur est optimisée pour de meilleurs résultats. La régularisation L2 est également con-

nue sous le nom de weight decay car elle force les poids à tendre vers 0 (sans jamais être à 0). λ est généralement déterminé par le jeu de validation. Généralement, la valeur de λ est comprise entre 10^{-2} et 10^{-6}.

Dans L1, également connu sous le nom de régression Lasso, nous avons :

$$\tilde{J}(\theta) = J(\theta) + \lambda \|\theta\|_1$$

Par conséquent, la valeur absolue du poids est pénalisée. Contrairement à L2, les poids peuvent ici être réduits à zéro. Par conséquent, L1 est très utile pour la compression du modèle. En fait, il convient de noter que la régression Lasso permet de simplifier, de l'interprétation du modèle et d'exclure certaines variables inutiles.

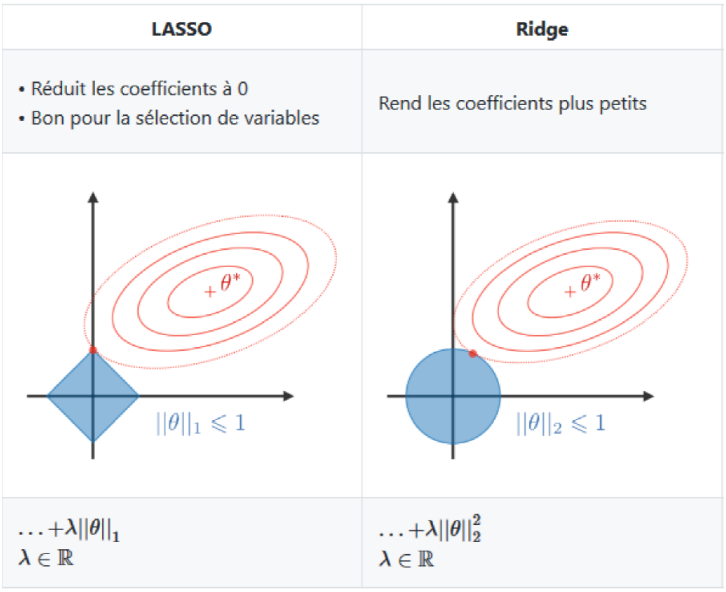

Figure 4.3: Illustration de la régularisation L1 (Lasso) et L2 (Ridge) (Amidi, 2019)

4.2.4 Désactivation

La désactivation (dropout) consiste à ignorer certaines unités (neurones) pendant la phase d'entraînement d'un ensemble de neurones sélectionnés au hasard. En effet, certaines unités et leurs connexions ne sont pas prises en compte lors du passage vers l'avant (forward) ou vers l'arrière (backward pass). Cependant, ces neurones ont été réactivés lors des phases de test et de prédiction. Par conséquent, la désactivation permet d'empêcher la co-adaptation sur les données d'entraînement, réduisant ainsi le risque de surentraînement.

Cette technique est largement utilisée pour améliorer les performances des réseaux de neurones dans divers domaines d'application. Cependant, l'un des inconvénients de sa désactivation est qu'elle augmente le temps d'entraînement (Srivastava *et al.*, 2014).

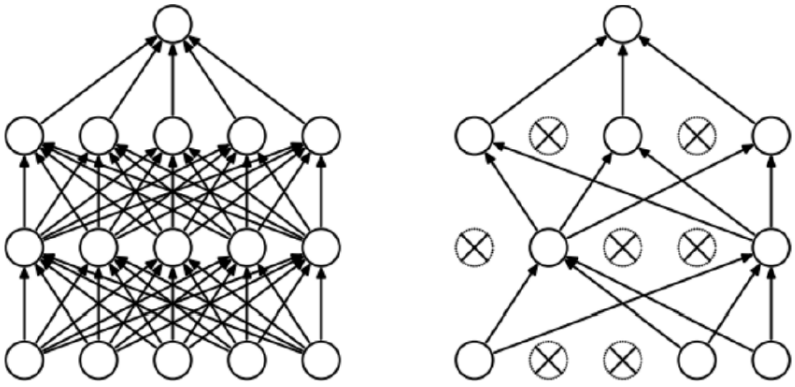

Figure 4.4: Illustration d'un réseau de neurone avec désactivation. (Srivastava *et al.*, 2014)

4.2.5 BatchNormalization

La normalisation des données peut être utile lorsque les composants des données proviennent d'échelles différentes. La normalisation vise à donner à tous les composants de l'entrée des caractéristiques similaires en se concentrant sur la moyenne et en la limitant à une plage de valeurs. Un des buts de la normalisation est d'avoir des valeurs du même ordre de grandeur pour accélérer la convergence. Une fois les données normalisées, ces dernières ont une forme de cercle, contrairement aux données non normalisées, qui ont tendance à avoir une forme d'ellipse. La normalisation de l'entrée présente également l'avantage de pouvoir être effectuée en prétraitement.

4.2.6 Résumé de l'architecture réseau

Notre modèle contient les couches suivantes :

1. Des couches de convolution du modèle préentraîné sur ImageNet le Inception-ResNet-v2 en excluant la couche entièrement connectée au sommet du réseau.
2. Une couche de neurone entièrement connectée de 512 neurones et une fonction d'activation «relu» avec un régularisateur qui applique les pénalités de régularisation L1 et L2.
3. Une couche de BatchNormalization.
4. Une couche de Dropout de 40% .
5. Une couche de neurone entièrement connectée de 256 neurones et une fonction d'activation «relu» avec un régularisateur qui applique les pénalités de régularisation L1 et L2..

6. Une couche de BatchNormalization.
7. Une couche de Dropout de 40% .
8. Une couche de neurone entièrement connectée de 128 neurones et une fonction d'activation «relu» avec un régularisateur qui applique les pénalités de régularisation L1 et L2.
9. Une couche de BatchNormalization.
10. Une couche de Dropout de 40% .
11. Une couche de neurone entièrement connectée de 64 neurones et une fonction d'activation «relu» avec un régularisateur qui applique les pénalités de régularisation L1 et L2.
12. Une couche de BatchNormalization.
13. Une couche de Dropout de 50% .
14. Une couche de neurone entièrement connectée de un neurone et une fonction d'activation «sigmoid».

Comme déjà mentionné, l'entrée du modèle est une image de taille 299x299 et ce dernier produit une sortie scalaire entre 0 et 1.

4.3 Données d'apprentissage

4.3.1 Collecte

Nous avons utilisé la base d'images RailSem19 (Zendel *et al.*, 2019) comme ensemble de base. Elle contient 8500 images uniques de taille 1024x1024 prises du point de vue de véhicules ferroviaires (trains et tramways) et de nombreuses

```
base_model= tf.keras.applications.InceptionResNetV2(input_tensor=input_tensor,include_top=False, weights='imagenet')
ase_model.trainable = True
x = base_model.output
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu', kernel_regularizer=keras.regularizers.l1_l2(l1=0.1, l2=0.01))(x)
x = BatchNormalization()(x)
x = Dropout(0.40)(x)
x = Dense(256, activation='relu', kernel_regularizer=keras.regularizers.l1_l2(l1=0.1, l2=0.01))(x)
x = BatchNormalization()(x)
x = Dropout(0.40)(x)
x = Dense(128, activation='relu', kernel_regularizer=keras.regularizers.l1_l2(l1=0.1, l2=0.01))(x)
x = BatchNormalization()(x)
x = Dropout(0.40)(x)
x = Dense(64, activation='relu', kernel_regularizer=keras.regularizers.l1_l2(l1=0.1, l2=0.01))(x)
x = BatchNormalization()(x)
x = Dropout(0.50)(x)
predictions = Dense(1,activation='sigmoid')(x)
# this is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)
```

Figure 4.5: implementation du Model

vues montrent des intersections entre véhicules routiers et ferroviaires (passages à niveau, tramways circulant dans les rues de la ville), rendant RailSem19 utilisable pour des applications ferroviaires et routières.

Figure 4.6: Exemples d'images dans Railsem19, accompagnées de figures d'annotations sémantiques

Chaque image est annotée dans un fichier JSON qui contient la liste d'objets dans image avec leurs dimensions. Les objets annotés sont :

- buffer-stop

- crossing
- guard-rail
- train-car
- platform
- rail
- switch-indicator
- switch-left
- switch-right
- switch-unknown
- switch-static
- track-sign-front
- track-signal-front
- track-signal-back
- person-group
- car
- fence
- person
- pole
- rail-occluder

- truck

Une large gamme d'annotations sémantiques est disponible pour ces images, grâce au code source fourni avec RailSem19. Ainsi, nous pouvons obtenir des annotations sémantiques à la fois basées sur la géométrie (polygones relatifs pour les chemins de fer, tous les chemins de fer sous forme de polylignes) et des cartes d'étiquettes denses avec de nombreuses étiquettes de route compatibles avec Cityscapes (voir figure 4.6 pour des exemples).

4.3.2 Préparation des classes

Nous divisons les images de Railsem19 en deux catégories :

classe 1 (anormale) : elle comprend les images contenant un ou plusieurs des objets :

- person-group
- car
- fence
- person
- rail-occluder
- truck

Classe 0 (normale) : elle comprend Railsem19 sans les images de classe 1.

L'analyse des fichiers JSON permet d'identifier 2153 images correspondant à la classe 1 et 6347 correspondant à la classe 0.

4.3.3 Augmentation des données

Nous avons utilisé plusieurs techniques d'augmentation de données en vue d'améliorer la capacité d'apprentissage du modèle. De ce fait, plusieurs ensembles d'entraînement on été créés et évalués. Leurs descriptions suivent :

Ensemble 1 : C'est la base dimages Railsem19. Comme déjà mentionné, elle contient 6347 images normales et 2153 images anormales.

Ensemble 2 : Il s'agit de l'ensemble Railsem19 équilibré en supprimant au hasard 4194 images normales. Il contient donc 2153 images normales et 2153 images anormales.

Ensemble 3 : Il s'agit de l'ensemble d'images obtenu en ajoutant du bruit aux images de l'ensemble 2. Les images ci-dessous fournissent un exemple.

Figure 4.7: exemple de l'ensemble 3

Ensemble 4 : Il s'agit d'un ensemble miroir de l'ensemble 2. Les images ci-dessous fournissent un exemple.

Figure 4.8: exemple de l'ensemble 4

Ensemble 5 : Il s'agit d'un ensemble d'images obtenues en inversant les couleurs d'obstacles pour les 2153 images anormales et en inversant les couleurs de l'image complète pour 2153 images normales sélectionnées au hasard parmi 6347 images normales originales. Les exemples ci-dessous illustrent la procédure.

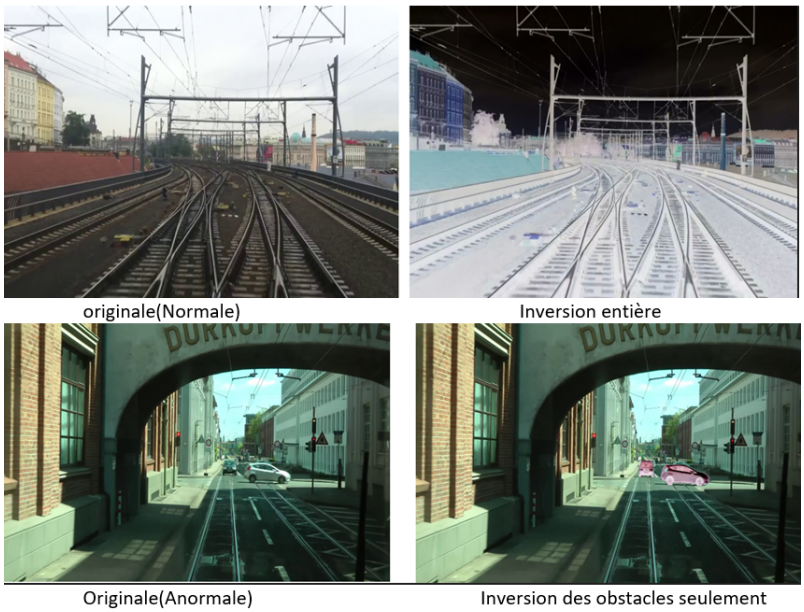

Figure 4.9: exemple de l'ensemble 5

Ensemble 6 : C'est l'ensemble d'images obtenu en ajoutant du brouillard aux images de l'ensemble 2 avec un modèle Foggy-CycleGAN (ghai szaher, 2020) . Les

exemples ci-dessous illustrent la procédure.

Figure 4.10: exemple de l'ensemble 6

Ensemble 7 : Cet ensemble crée 4191 nouvelles images anormales afin d'égaliser les tailles des deux classes de l'ensemble Railsem19 à 6347 images chacune. Les nouvelles images anormales comprennent :

- 2153 images obtenues en déplaçant vers la droite les obstacles des 2153 images anormales originales .

- 2041 images obtenues en déplaçant les obstacles des 2041 images anormales d'origine vers la gauche (les 2041 images sont sélectionnées aléatoirement parmi les

2153 images anormales d'origine). Les exemples ci-dessous illustrent la procédure.

Figure 4.11: exemple de l'ensemble 7 avec une deplacement

Nous avons aussi essayé d'augmenter l'ensemble des images anormales en utilisant un modèle génératif de type CycleGan (Zhu *et al.*, 2017). Nous avons entraîné le réseau Cyclegan avec les 8500 images de la base RailSem19 en vue de générer des images normales à partir des images anormales, et vice versa. La figure ci-dessous donne un exemple des résultats obtenus :

Figure 4.12: exemple d'image généré par le CycleGan Version 1)

Le modèle CycleGan a été incapable de créer des anomalies d'intérêt dans l'image générée, possiblement à cause de la taille de la base Railsem19 utilisée pour l'entraînement. Nous avons alors essayé d'améliorer l'entraînement du modèle en augmentant la taille de la base Railsem19 avec les images suivantes :

- 1 398 images miroirs générées de l'ensemble original de type anormal.
- 1 398 images qui contiennent des bruits générés de l'ensemble original de type anormal.
- 1 398 images inversées aléatoirement générées de l'ensemble original de type anormal.

Nous avons entrainé le modèle sur la nouvelle bibliothèque d'images et l'exemple ci-dessous illustre les résultats obtenus après 110 itérations :

Malgré l'augmentation du nombre de données d'apprentissage, le modèle Cycle-Gan n'a toujours pas été capable de générer des images contenant des obstacles utiles. De ce fait, il a été abandonné comme technique d'augmentation de données dans ce travail.

Figure 4.13: exemple d'image généré par le CycleGan Version 2)

4.4 Expériences

Nous avons utilisé les ensembles de données précédents individuellement ou en combinaison dans différentes expériences de validation de notre modèle de réseau CNN. Ces combinaisons sont décrites au fur et à mesure dans ce qui suit.

4.4.1 Environnement de simulation

Nous avons utilisé Google Colaboratory (Colab) (Google, 2017). Colab est consacré à la création et à l'enregistrement du code et à l'exécution du processus de formation. Le code se présente sous la forme de fichiers Jupyter Notebook. Colab nous permet d'exécuter du code en mode CPU, GPU, TPU sur des machines virtuelles Linux, avec la possibilité de monter des comptes Google Drive.Dans notre projet, nous avons utilisé la version professionnelle de google colab, qui présente les fonctionnalités suivantes :

- **GPU plus rapides:** vous bénéficiez d'un accès prioritaire aux GPU les plus rapides,donc il vous sera parfois possible d'accéder à un GPU T4 ou P100.
- **Plus de mémoire :** vous pouvez accéder à des VM à haute capacité de

mémoire sous réserve de disponibilité.

- **Environnements d'exécution plus durables:** Avec un notebook qui dispose d'une autonomie plus longue et de délais d'inactivité réduits, vous êtes moins souvent déconnecté.

4.4.2 Production des données augmentées

Nous avons utilisé la classe ImageDataGenerator de Keras, qui permet de génèrer des données d'images de tenseurs par lots afin d'augmenter notre base d'image en temps réel avec les paramètres suivants :

rescale : Nous multiplions les données par 1/255.donc, Nous restreignons donc les valeurs des données entre 0 et 1.

rotation_range : Nous définissons une plage de 40 degrés pour la rotation aléatoire.

width_shift_range : Nous avons défini une plage de 20 % pour le mouvement horizontal aléatoire (décalage aléatoire de l'image vers la droite ou la gauche).

height_shift_range: Nous avons défini une plage de 20 % pour le mouvement vertical aléatoire (décalage aléatoire de l'image de haut en bas).

shear_range: Nous avons défini une plage de 20 degrés pour l'angle d'inclinaison aléatoire.

zoom_range: Nous avons défini une plage de 20 % pour le zoom aléatoire.

fill_mode: Nous définissons le mode de remplissage sur " nearest ". Ainsi, les points en dehors de la plage d'entrée sont remplis de la manière aaaaaaaa|abcd|ddddddd.

4.5 Entraînement du réseau et résultats

Nous avons entraîné notre modèle sur 110 itérations avec plusieurs combinaisons des ensembles d'entraînement précédents, en commençant avec la base Railsem19 sans modifications (ensemble 1). Pour cet ensemble déséquilibré, qui contient 6347 images normales et 2153 images anormales, le taux moyen de bonnes détections du modèle pour l'ensemble de test à été de 78 %.

La suppression au hasard de 4194 images de type normal pour équilibrer les deux classes à 2153 image chacune (ensemble 2) à fait baisser le résultat précédent à 70%.

En revanche, l'équilibrage des classes par augmentation de données à mener à une amélioration substantielle. Ainsi, l'entraînement du modèle en ajoutant l'ensemble 7 1a utilisant les ensemble 1 (Railsem19 sans modifications) et l'ensemble 7. Comme déjà mentionné, l'ensemble 7 contient les 2153 images originales après déplacement de l'obstacle vers la droite et 2041 images originales après déplacement l'obstacle vers la gauche. Dans ce cas, nous avons obtenu un taux moyen de bonnes détections de 92,18 % pour l'ensemble de test.

Toujours en vue d'augmenter la capacité du modèle, nous l'avons entraîné en ajoutant à notre base d'entraînement (ensembles 1 et 7) les ensembles 3, 4, 5, ce qui donne 12806 images normales et 12806 images anormales. Comme déjà mentionné, l'ensemble 3 contient les 2153 images originales anormales après mise en miroir et 2153 images originales normales (choisi aléatoirement parmi les 6347 images originales normales) après mise en miroir, l'ensemble 4 contient les 2153 images originales anormales après Insertion du bruit et 2153 images originales normales (choisi aléatoirement parmi les 6347 images originales normales) après Insertion du bruit, et l'ensemble 5 contient les 2153 images originales anormales après inversion

de couleur d'obstacle et 2153 images originales normales (choisi aléatoirement parmi les 6347 images originales normales) après inversion de couleur. Dans ce cas, un taux moyen de bonnes détections de 92 % pour l'ensemble de test

Un autre entraînement en remplaçant dans notre base d'entraînement l'ensemble 5 par l'ensemble 6 qui, Comme déjà mentionné, contient les 2153 images originales anormales après application du modèle Foggy-CycleGAN et 2153 images originales normales après application du modèle Foggy-CycleGAN. avec les 12 806 images normales et 12 806 images anormales obtenues, le taux moyen de bonnes détections est de 91% pour l'ensemble de test.

Finalement, l'entraînement du réseau en remplaçant l'ensemble 3 par l'ensemble 5 dans le cas précédent, pour les mêmes nombres d'images normales et anormales a donné taux moyen de bonnes détections de 98.28% sur l'ensemble de test, avec une précision de 97,67% et un rappel de 98,92%. Ce résultat fut obtenu avec la partition suivante des 25 612 images de l'ensemble d'entraînement :

- **ensemble d'entraînement :** 21612 images (10806 : de type normale , 10806 : de type anormale).
- **ensemble de validation:** 1200 images (600 : de type normale , 600 : de type anormale).
- **ensemble de test :** 2800 images (1400 : de type normale , 1400 : de type anormale).

Le tableau 4.14 résume les différents entraînements effectués et les résultats obtenus.

	Entrainement 1	Entrainement 2	Entrainement 3	Entrainement 4	Entrainement 5	Entrainement 6
Ensemble 1	X		X	X	X	X
Ensemble 2		X				
Ensemble 3				X	X	
Ensemble 4				X	X	X
Ensemble 5				X		X
Ensemble 6					X	X
Ensemble 7			X	X	X	X
Précision	78%	70%	92,18%	92%	91%	98.28%

Figure 4.14: Résultats des différents apprentissages

4.5.1 Validation croisée

Les résultats précédents ont été obtenus après une seule partition des ensembles d'entraînement. Ils ne donnent pas non plus d'information sur la précision et rappel des résultats à part le dernier. Dans le but d'obtenir des resulats statistiquement fiables, nous avons répétés les expériences 1, 2 et 6 en utilisant la validation croisée à 10 tours, en fournissant à chaque fois le taux moyen de bonnes détections, le taux de précision et le taux de rappel obtenus. La validation croisée consiste à diviser la base de données en N sous-ensembles, dont on prend en rotation l'un pour tester et les autres pour l'entérinement. Dans notre cas, nous avons utlisé $N = 10$ pour chaque expérience.

Étant donné le temps de demandé par chaque validation croisée et le délai pour déponser ce mémoire, nous avons appliqué cette méthode seulement pour l'entraînement 1 (Railsem19 sans ajout ou retrait), pour l'entraînement 2 (Railsem19 équilibré par le retrait) et l'entraînement 6 (Railsem19 équilibré par l'ajout). Les résultats obtenus sont résumés dans les figures suivantes, qui donnent les valeurs pour chaque tour et en moyenne. Comme on peut le constater, les résultats obtenus lors de la validation à 1 tour sont maintenus, avec une légère baisse quant on fait

la moyenne pour 10 tours.

	Accuracy	Précision	Recall
1	73,05	48,77	0,53
2	77,05	55,49	0,44
3	77,17	51,48	0,60
4	77,76	57,50	0,52
5	73,88	50,19	0,58
6	78,82	58,73	0,52
7	76,94	51,63	0,47
8	76,58	54,74	0,57
9	76,58	57,22	0,44
10	77.41	55,97	0,53

Figure 4.15: Résultats de la méthode validation croisée pour l'entraînement 1

	Accuracy	Précision	Recall
1	68,45	61,42	0,83
2	68,21	71,42	0,61
3	70,99	72,99	0,73
4	69,14	67,36	0,74
5	64,96	62,27	0,77
6	72,62	72,24	0,80
7	58,83	54,98	0,82
8	59,53	60,21	0,52
9	61,62	57,18	0,93
10	63,72	58,17	0,76
Moyen	65,80	63,82	0,96

Figure 4.16: Résultats de la méthode validation croisée pour l'entraînement 2

	Accuracy	Précision	Recall
1	93,79	91,43	0,96
2	93,79	91,46	0,96
3	93,28	89,97	0,97
4	94,18	92,53	0,95
5	92,97	89,78	0,97
6	92,93	90,04	0,96
7	93,86	91,79	0,96
8	93,12	89,97	0,97
9	92,73	88,97	0,97
10	93.16	89,97	0,98
Moyen	93,39	90,59	0,96

Figure 4.17: Résultats de la méthode validation croisée pour l'entraînement 6

4.6 Discussion

Les résultats précédents montrent que la plus mauvaise performance de détection d'obstacles a été obtenue en utilisant Railsem19 tel quel (entraînement 1), soit en maintenant le déséquilibre entre les classes 0 et 1 (6347 images normales et 2153 images anormales). Un taux moyen de bonnes détections de 77.41% a alors été obtenu. La tentative d'équilibrage par le bas, en supprimant 4194 image normales prises au hasard afin d'avoir 2153 images dans les deux classes à fait baisser ce taux de près de 12 points. En revanche, l'équilibrage de l'ensemble d'apprentissage par l'augmentation des données de la classe anormale, pour obtenir 6347 images dans chaque classe, ou encore l'augmentation des données des deux classes a plus de 6347 images, tout en maintenant les nombres égaux, ont donné de bien meilleurs résultats. Ainsi, l'équilibrage des classes par l'ajout d'images anormales en déplaçant les obstacles a augmenté le taux moyen de bonnes détections à 92,18 % (entraînement 3). L'augmentation du nombre d'images dans chacune des classes a aussi donné de bons résultats, mais avec des rendements variables. Par exemple, l'ajout additionnel d'images obtenues par ajout de bruit, d'effets miroirs, et d'inversion de couleurs a baissé le taux à de 92 % (entraînement 4), et le remplacement du sous-ensemble obtenu par inversion de couleurs par celui par ajout de brouillard a baissé le taux encore plus à 91% (entraînement 6). En revanche, l'ajout d'images obtenues par effets miroirs, inversion de couleurs et effet de brouillard as fait monter le taux moyen de bonnes détection à 98,28% (entraînement 6), montrant que l'ajout d'images bruitées a un impact négatif sur l'apprentissage. Le taux de 98.28 % a été obtenu avec l'ensemble d'entraînement suivant :

- 8500 images orignales de la base de données d'images RailSem19.
- 4194 Images obtenue en déplaçant un obstacle.

- 4306 Images inversée.
- 4306 Images Miroir.
- 4306 Images générée par le modèle Foggy-CycleGAN.

Les deux figures suivantes montrent l'évolution du taux de bonnes détections et de la fonction de perte lors de l'apprentissage 6. Noter cependant que la moyenne de ce taux est de 93,39% par validation croisée à 10 tours. Il semble donc que l'échantillonnage de données d'apprentissage et de test joue un rôle important dans les taux de bonne détections et que plus de données sont requises pour une distribution homogène.

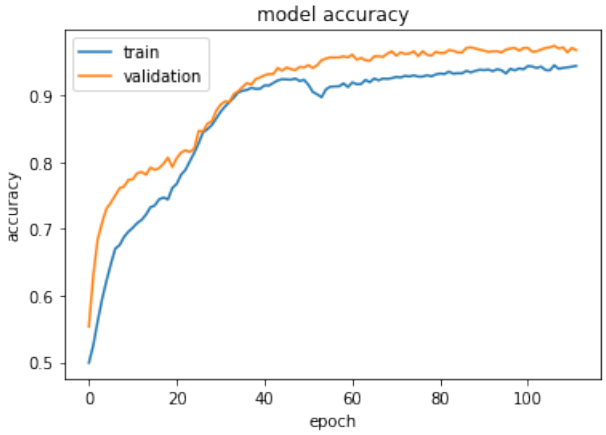

Figure 4.18: la courbe du précision

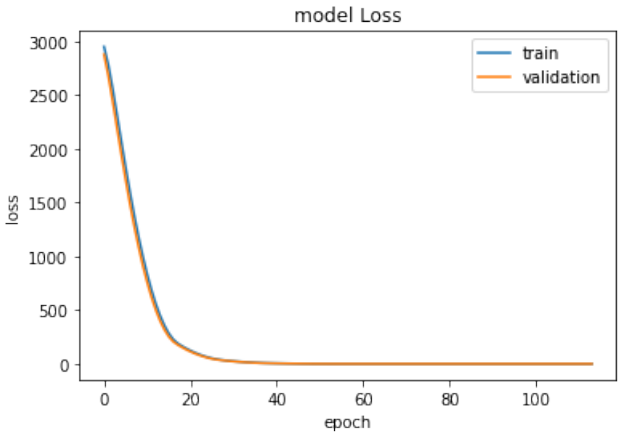

Figure 4.19: la courbe du perte

CHAPTER V

CONCLUSION

Ce travail met en œuvre un système d'apprentissage automatique pour la détection automatique d'obstacles dans les systèmes de pilotage automatique des trains.

Le système développé entraîné à l'aide de la base de données d'images RailSem19, qui contient 8500 images uniques de taille 1024x1024 prises dans un contexte de véhicules ferroviaires (trains et tramways). Chaque image est annotée dans un fichier Json qui contient une liste d'objets pour cette image et ses dimensions.

Au cours du projet, nous avons utilisé des techniques d'augmentation des données et nous avons étudié plusieurs modèles de détection d'anomalies basés sur sur des travaux antérieurs afin de concevoir notre modèle. Les approaches incluent InceptionResnetV2, Constructive Loss, ranger optimiseur et Foggy-CycleGAN pour simuler de différents heures de la journée et de différentes saisons de l'année et augmenter notre base donnée.

Pour la réalisation, nous avons augmenté notre base de données avec des techniques classiques (déplacement d'obstacles, mise en miroir, images inversées et changements de luminosité) et utilisé la classe ImageDataGenerator de Keras qui génère des tenseurs de données d'images par lots pour augmenter notre base d'image en temps réel.

Nous avons construit notre modèle via un réseau de neurones convolutifs, qui comprend un modèle pré-entraîné et plusieurs techniques de régularisation, et le résultats obtenus montrent l'efficacité de modèles a détecter les obstacles dans les systèmes de pilotage automatique de trains.

À l'avenir, Il serait intéressant de voir une étude comparative sur les déférent type du modèle utilisé pour la détection d'anomalie telle que d'autres architectures de Réseaux de neurones profond, et les réseaux génératifs comme les autocodeurs et les BiGAN pour une comparaison de performances.

RÉFÉRENCES

Akcay, S., Abarghouei, A. A. et Breckon, T. P. (2018). Ganomaly: Semi-supervised anomaly detection via adversarial training. *CoRR*, *abs/1805.06725*.

Amidi, A. (2019). Deep learning tips and tricks cheatsheet. Récupéré de `https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks`

Boukaye Boubacar Traore, Bernard Kamsu-Foguem, F. T. (2018). *Deep convolution neural network for image recognition*. Elsevier.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. et Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. Dans *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 248–255. `http://dx.doi.org/10.1109/CVPR.2009.5206848`

Dertat, A. (2017). Applied deep learning - part 3: Autoencoders. Récupéré de `https://towardsdatascience.com/`

Donahue, J., Krähenbühl, P. et Darrell, T. (2016). Adversarial feature learning. *CoRR*, *abs/1605.09782*.

Dumoulin, V. et Visin, F. (2016). A guide to convolution arithmetic for deep learning.

Forward, C. F. (2020). Deep learning for anomaly detection. Récupéré de `https://ff12.fastforwardlabs.com/`

Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, *1*(2), 119–130.

Gandhi, A. (2021). Data augmentation | how to use deep learning when you have limited data. Récupéré de `https://nanonets.com/`

ghai szaher (2020). Simulating weather conditions on digital images. Récupéré de `http://ghaiszaher.me/Foggy-CycleGAN/`

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. et Bengio, Y. (2014). Generative adversarial networks. *arXiv preprint arXiv:1406.2661*.

Google (2017). google colab. Récupéré de `https://colab.research.google.com/`

Hadsell, R., Chopra, S. et LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. Dans *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, 1735–1742. `http://dx.doi.org/10.1109/CVPR.2006.100`

Hubel, D. H. et Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, *160*(1), 106–154.

Kingma, D. P. et Welling, M. (2013). Auto-Encoding Variational Bayes. *arXiv e-prints*, p. arXiv:1312.6114.

Mattia, F. D., Galeone, P., Simoni, M. D. et Ghelfi, E. (2019). A survey on gans for anomaly detection. *CoRR*, *abs/1906.11632*.

Mlyahilu, J., Kim, Y. et Kim, J. (2019). Classification of 3d film patterns with deep learning. *Journal of Computer and Communications*, *07*, 158–165. `http://dx.doi.org/10.4236/jcc.2019.712015`

Pang, G., Shen, C., Cao, L. et van den Hengel, A. (2020). Deep learning for anomaly detection: A review. *CoRR*, *abs/2007.02500*. Récupéré de `https://arxiv.org/abs/2007.02500`

Ruff, L., Vandermeulen, R. A., Görnitz, N., Binder, A., Müller, E., Müller, K. et Kloft, M. (2019). Deep semi-supervised anomaly detection. *CoRR*, *abs/1906.02694*.

Sinha, U. (2022). Image convolution examples. Récupéré de `https://aishack.in/tutorials/image-convolution-examples/`

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. et Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929–1958.

Szegedy, C., Ioffe, S. et Vanhoucke, V. (2016). Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, *abs/1602.07261*.

Wright, L. et Demeure, N. (2021). Ranger21: a synergistic deep learning optimizer. *CoRR*, *abs/2106.13731*. Récupéré de

https://arxiv.org/abs/2106.13731

Yu, J. L. . S. (2019). Introduction to machine learning fall. Récupéré de https://www.eecs189.org/static/notes/n27.pdf

Zendel, O., Murschitz, M., Zeilinger, M., Steininger, D., Abbasi, S. et Beleznai, C. (2019). Railsem19: A dataset for semantic rail scene understanding. Dans *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 1221–1229. http://dx.doi.org/10.1109/CVPRW.2019.00161

Zhu, J.-Y., Park, T., Isola, P. et Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. Dans *Proceedings of the IEEE international conference on computer vision*, 2223–2232.