Réseaux de neurones profonds pour la classification d'objets en imagerie infrarouge : apports de l'apprentissage à partir de données synthétiques et de la détection d'anomalies

Antoine d' Acremont

DOCTORAT BRETAGNE LOIRE / MATHSTIC

ENSTA Bretagne

THÈSE DE DOCTORAT DE

L'ENSTA BRETAGNE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Signal, Image, Vision*

Par

Antoine d'Acremont

Réseaux de neurones profonds pour la classification d'objets en imagerie infrarouge : apports de l'apprentissage à partir de données synthétiques et de la détection d'anomalies

Thèse présentée et soutenue à l'ENSTA Bretagne, le 7 décembre 2020
Unité de recherche : labSTICC

Rapporteurs avant soutenance :

Antoine MANZANERA	Professeur, ENSTA Paris
Bertrand LE SAUX	Digital Technologies Engineer, ESA/ESRIN

Composition du Jury :

Examinateurs :	François ROUSSEAU	Professeur, IMT Atlantique
	Pierre BEAUSEROY	Professeur des universités, Université de Technologie de Troyes
	Marie-Véronique SERFATY	Resp. Innovation Pôle NUMérique, Agence Innovation Défense
Dir. de thèse :	Alexandre BAUSSARD	Professeur des universités, Université de Technologie de Troyes
	Ronan FABLET	Professeur, IMT Atlantique
Encadr. de thèse :	Guillaume QUIN	Expert Technique, MBDA France

Invité(s) :

Thierry VALLAS	Expérimentateur optronique, Section Technique de l'Armée de Terre

Remerciements

Avant de rentrer dans le vif du sujet, je tenais à réserver ces quelques lignes pour remercier tous ceux sans qui cette thèse n'aurait pu avoir lieu. Pour commencer je voulais remercier mes deux directeurs de thèse Alexandre et Ronan qui m'ont soutenu et suivi avec beaucoup de patience tout au long de cette thèse et qui ont toujours trouvé le temps de m'aider, aussi bien sur le plan technique et administratif que humain. Ce soutien a été d'autant plus important avec les changements d'organisation et l'isolement liés à la situation sanitaire.

Merci aussi à Guillaume qui m'a suivi chez MBDA depuis mon arrivée qui m'a aidé à m'intégrer au sein de l'entreprise et de l'équipe et soutenu personnellement et professionnellement tout au long de mon passage au sein du service vision. Je souhaite aussi mentionner Lydiane et Anne-Lise qui m'ont permis de faire cette thèse chez MBDA en m'intégrant comme un membre à part entière de leurs équipes. Merci aussi aux autres membres anciens et actuels du service. Je n'oublierai pas ces moments de partages de savoirs et ces discussions autour de la sacro-sainte machine à café qui ont rythmé mes journées au Plessis-Robinson. Merci aussi au nageurs de l'équipe, aux amateurs de soirées plaisantes et aux membres du CPNT dont les virées utopiques ont fortement contribué au maintien du moral pendant ces trois années.

Un grand merci aussi aux rapporteurs pour leurs retours détaillés sur le manuscrit et aux membres du jury pour avoir pris le temps de suivre ma soutenance dans un format assez atypique.

Je voulais aussi remercier ma famille qui m'a soutenu tout au long de cette thèse et sans qui ces journées de confinement n'auraient sûrement pas été aussi agréables.

Enfin un grands merci à Chloé qui a réussi à me supporter jusque dans les dernières secondes de cette aventure et qui a su faire preuve d'une grande patience et une grande douceur malgré le stress et l'agitation des derniers préparatifs, perturbés par les événements de novembre.

Table des matières

Table des figures

Liste des tableaux

Acronymes

cfCNN *Compact and Fully Convolutionnal Neural Network.*

CLOF Cascade de *Local Outlier Factor.*

CNN Réseau de neurones convolutif ou *Convolutionnal Neural Network.*

CPU *Central Processing Unit.*

DRI Détection - Reconnaissance - Identification.

GAP *Global Average Pooling.*

GD Descente de Gradient ou *Gradient Descent.*

GPU *Graphics Processing Unit.*

HOG *Histogram of Oriented Gradients.*

IR Infrarouge.

LOF *Local Outlier Factor.*

LReLU *Leaky Rectified Linear Unit.*

LSE *Lifted Structured Embedding.*

LWIR *Long Wave Infrared.*

MLP Perceptron Multi-Couche ou *Multi-Layer Perceptron.*

MWIR *Middle Wave Infrared.*

ODIN *Out-of-DIstribution detector for Neural-networks.*

ReLU *Rectified Linear Unit.*

SGD Descente de Gradient Stochastique ou *Stochastic Gradient Descent.*

STN *Spatial Transformer Network.*

SVM *Support Vector Machines.*

SWIR *Short Wave Infrared.*

TSNE *T-Distributed Stochastic Neighbor Embedding.*

Introduction

Introduction

Contexte

La classification visuelle de véhicules, et plus généralement de toute cible d'intérêt tactique, est une composante importante du renseignement militaire. Les techniques de classification des équipements, véhicules et armements sont toujours enseignées à tous les niveaux dans les forces armées. Classer précisément un équipement sur un champ de bataille après l'avoir détecté permet de mieux appréhender la situation tactique et contribue fortement à améliorer les chances de survie des personnels présents.

La classification de cibles est souvent divisée en deux étapes : reconnaissance et identification, ou RI. Nous parlerons de reconnaissance lorsqu'il s'agira de classer les véhicules par leur type, *i.e.* voiture, camion ou char d'assaut, et d'identification lorsque nous souhaitons donner le modèle précis du véhicule. Ces deux tâches étaient historiquement traitées par des opérateurs humains et nécessitaient des connaissances précises sur les différents éléments constitutifs des équipements à identifier, comme le nombre de roues ou la position des différents armements. Reconnaitre mais surtout identifier à l'œil nu ou avec une optique à fort grossissement nécessite de bonnes conditions d'observation. De plus, la très grande majorité des cibles à classer peuvent être partiellement ou totalement camouflées, rendant le processus encore plus compliqué. L'introduction de caméra à intensification de lumière, puis de l'imagerie infrarouge a permis aux opérateurs chargés de classer les cibles de travailler de nuit ou dans des conditions météorologiques dégradées.

Dans cette thèse, nous traiterons principalement de l'imagerie infrarouge ou imagerie IR. Celle-ci permet de visualiser une partie du spectre électromagnétique correspondant à la chaleur émise par un objet, ou rayonnement infrarouge. Tout objet dont la température est supérieure à celle du zéro absolu émet dans l'IR. Ainsi il est possible d'observer les véhicules militaires grâce à la chaleur émise par leurs équipements et leurs moteurs, et ce même de nuit. Contrairement aux images visibles, en noir et blanc ou en couleur, les images infrarouges ont une dynamique plus importante. Elles auront aussi une résolution plus faible que dans le visible à cause des limites des technologies utilisées pour fabriquer des capteurs infrarouges.

Néanmoins, identifier ou reconnaitre un véhicule en IR ou dans le visible nécessite une grande concentration de la part d'un opérateur humain. Cet opérateur peut être amené à surveiller une zone de taille importante et traiter le plus rapidement possible une grande quantité d'informa-

tions. À cela viennent s'ajouter des conditions d'observation souvent difficiles. Un fantassin sera parfois contraint de rester discret, voir immobile, dans des conditions de température et d'humidité pénibles. Un observateur dans un véhicule devra aussi travailler dans un environnement exigu et bruyant. De plus, ces observateurs doivent souvent travailler dans un état de fatigue morale et physique. L'ensemble de ces éléments contribuent à rendre les tâches d'identification et de reconnaissance difficiles pour un humain. On a donc cherché à automatiser ces tâches de classification.

Les méthodes spécifiques à l'imagerie se basent sur des outils proches de ceux utilisés dans le domaine visible. Elles se décomposent généralement en deux étapes : une étape d'extraction automatique de caractéristiques dans les images suivie d'une étape de classification grâce à ces caractéristiques. Le développement d'algorithmes de classification basés sur de l'apprentissage supervisé a fortement contribué à améliorer les performances des systèmes automatiques de classification en infrarouge. Les algorithmes d'apprentissage supervisé sont entrainés à effectuer une tâche particulière, ici la classification, à l'aide de données annotées. Ce développement s'est notamment accéléré ces dernières années avec le gain de popularité des réseaux de neurones et plus particulièrement des réseaux de neurones convolutifs, ou CNN pour *Convolutional Neural Network*, dédiés au traitement d'images.

Les CNN sont formés d'un enchainement de couches de neurones interconnectés, qui traitent de façon simultanées les problèmes d'extraction de caractéristiques et de classification. Avec le développement du calcul numérique massivement parallélisé, ces CNN ont rapidement surpassé les autres méthodes de classification d'images, notamment sur le défi ImageNet [32] qui comprends plusieurs millions d'images réparties dans plus de 20000 classes. Le développement progressif des réseaux de neurones, dont les CNN, a notamment mis en évidence le lien entre leurs performances et la quantité de données utilisées pour les entrainer. Ainsi, il est nécessaire de disposer d'un grand nombre d'images annotées pour pouvoir utiliser un CNN pour des problèmes de classification d'image. Bien qu'il n'existe pas de règle formelle qui fixe le nombre exact d'images nécessaires pour garantir un niveau de performance minimum pour un CNN, plusieurs observations empiriques ont mis en évidence le lien entre la complexité du problème de classification à résoudre et la taille de la base d'image d'entrainement. Ainsi, plus le nombre de classes et leur variabilité augmente, plus le nombre d'images annotées nécessaire sera important.

Il existe de nombreux domaines dans lesquels la communauté scientifique a mis gratuitement à disposition des jeux de données annotés pour faciliter le développement des CNN comme par exemple la conduite autonome. Cependant, cette abondance de ressources ne couvre pas le domaine militaire, surtout pour l'infrarouge où le seul ensemble d'image de véhicule militaires ne peut être acheté depuis quelques années que par des entités basées aux états-unis. En effet, de telles données sont souvent protégées par des règles de confidentialité strictes pour préserver leur valeur stratégique, quand elles ne sont pas tout simplement impossibles à obtenir.

De plus, pour entrainer un CNN, nous avons aussi besoin d'images qui représentent de la façon la plus exhaustive possible les conditions rencontrées dans le contexte opérationnel. Cela implique

de faire ou de disposer d'acquisitions d'images dans différents environnements, avec différentes conditions météo et pour différents angles de vue. Pour les images de véhicules en infrarouge il faut aussi ajouter des données pour différents états thermiques. En effet, l'aspect des cibles peut fortement varier entre une observation du véhicule à froid, en roulement ou statique avec son moteur allumé. Acquérir suffisamment de données, avec les annotations qui correspondent, pour couvrir l'ensemble de ces variations n'est donc généralement pas faisable.

Pour pouvoir envisager l'utilisation de CNN dans ce contexte, il faut donc être en mesure de trouver un moyen de compenser la difficulté d'obtenir des images réelles. Dans les cas où l'on dispose quand même d'une faible quantité d'images pour un problème de classification d'images, une première approche consiste à utiliser des réseaux pré-entrainés avec des données correspondant à un autre domaine d'application. De cette manière, les réseaux pré-entrainés apprennent des caractéristiques utiles pour la classification d'images provenant d'autres jeux de données. Il suffira d'utiliser une petite quantité d'image pour terminer l'entrainement de ces réseaux pré-entrainés.

Cependant, l'utilisation de modèles pré-entrainés présente un risque de sécurité qui pourrait affecter fortement les performances de systèmes critiques. Nous reviendrons sur ces risques dans la suite de ce manuscrit. Dans ces conditions, et dans les cas où nous ne disposons d'aucune image réelle, une autre alternative est d'utiliser la simulation pour générer des données en quantité suffisante pour entrainer le CNN. La simulation reste néanmoins une approximation de phénomènes physiques réels. Cette différence peut avoir un impact sur les performances d'un réseau de neurones utilisé pour la classification d'images réelles, après avoir été entrainé sur des données simulées. Dans ce manuscrit, nous évaluons cet impact sur le problème de la classification de véhicules en infrarouge.

À ce problème de données s'ajoute une contrainte supplémentaire lié à la structure des systèmes de RI dans les équipements militaires. En effet ces derniers sont intégrés le plus souvent dans une chaine modulaire de Détection-Reconnaissance-Identification ou DRI. Ainsi, la fonction de RI est précédée d'un étage de Détection dont le rôle est de localiser dans une images les zones qui contiennent des éléments à classer par l'étage de RI. Dans un cas idéal, l'étage de détection ne fournit à l'étage de RI que des images contenant une cible parfaitement centrée. Dans des conditions opérationnelles, la détection peut être imparfaite et la cible dans l'image peut apparaître comme étant translatée ou son échelle modifiée. Étant donné les contraintes de validation d'un système militaire, il est fréquent de réutiliser des modules déjà validés. Ainsi, nous nous plaçons donc dans un cas où nous n'avons pas de contrôle sur la fonction de détection. Par conséquent, le CNN réalisant la tâche de RI doit donc être robuste à ces perturbations.

Dans certains cas extrêmes, nous pouvons obtenir en sortie du système de détection des images de fonds sans aucune cible ou même des images contenant des véhicules inconnus pour le système RI. Dans le cas où ce système sera basé sur un CNN, une première approche consiste à faire apprendre au réseau une classe dite "inconnue" en plus des classes régulières, incluses dans la base d'apprentissage. Cependant, il n'existe aucune garantie que les images utilisées représentent correctement les fonds ou les nouveaux véhicules que le système peut avoir traiter. Cela n'exclut

pas non plus de voir le réseau classer un nouveau véhicule ou un fond parmi l'une des classes régulières. Nous n'avons donc pas choisi d'ajouter une classe "inconnue".

Par conséquent, le CNN est incapable de déterminer de lui-même si ce nouvel exemple est différent de ceux qu'il connait. Dans le contexte militaire, une erreur de classification peur avoir des conséquences désastreuses. Il est donc important de mettre en place des stratégies permettant d'identifier ces erreurs de classification afin d'améliorer l'interprétation des résultats de classification des réseaux de neurones et faciliter leur utilisation dans des applications avec de fortes exigences de sécurité comme la reconnaissance et l'identification de véhicules militaires dans les images infrarouges.

Présentation des contributions

La première contribution de ces travaux est le réseau cfCNN. Pour traiter le problème posé par la difficulté de constituer des bases de données suffisamment importantes pour entrainer des réseaux de neurones pour la RI, nous proposons d'utiliser des données simulées. Avant d'introduire cette nouvelle architecture, nous montrons qu'utiliser un CNN issu de l'état de l'art entrainé sur des données simulées pour classer des véhicules dans des images réelles ne garantit pas de meilleurs performances qu'un algorithme basé sur des machines à vecteurs de support. Nous proposons alors un CNN compact, entièrement convolutif, et qui incorpore notamment une couche de *Global-Average-Pooling*. Ce modèle est ensuite entrainé sur trois jeux de données simulées de différente qualité puis il est évalué sur des données réelles. Sur l'ensemble de ces tests, il obtient les meilleurs résultats parmi l'ensemble des algorithmes de classification testés. Nous évaluons ensuite la robustesse du cfCNN face à des perturbations de l'image d'entrée. Ce réseau étant prévu pour traiter des problèmes de RI, il est destiné à recevoir les entrées d'un étage de détection. Or, cette détection peut ne pas être parfaite, et, dans certains cas, l'image d'entrée peut être perturbée. Nous avons donc mesuré la réponse du cfCNN face a des translations, des changements d'échelle et l'ajout de bruit gaussien dans l'image d'entrée. Ces résultats ont été comparés à ceux obtenus par un ensemble de modèles issus de l'état de l'art. Nous avons alors observé que, même s'il n'obtenait pas les meilleures performances pour la totalité des perturbations testées, il restait en moyenne le plus robuste.

La deuxième contribution porte sur la détection des anomalies de classification. En effet, même si les CNN se sont imposés comme la solution par excellence pour les problèmes de classification d'images, ils ne sont pas pour autant infaillibles. Sous certaines conditions, notamment face à des entrées perturbées, ils peuvent se tromper dans leurs résultats de classification. Or, dans un contexte militaire, de telles erreurs peuvent avoir des conséquences graves. Nous avons donc proposé un outil de détection d'anomalies de classification pour avertir un utilisateur dans le cas où la prédiction du réseau serait potentiellement erronée. Cet outil est basé sur un algorithme de détection d'anomalies appelé *Local Outlier Factor*. Il est entrainé conjointement au réseau et permet, après l'entrainement, de classer la réponse du réseau comme anomalie ou comme exemple

régulier en utilisant les sorties de la dernière couche cachée. Nous avons d'abord testé l'outil dans un cas ou les données de test et d'entrainement appartiennent au même domaine *i.e.* réel-réel ou simulé-simulé. Durant ce test, le détecteur a égalé des méthodes de détection d'anomalies issues de l'état de l'art. Cependant, lors d'essais avec changement de domaine, *i.e.* simulé-réel, ce détecteur ne permettait pas de détecter les anomalies et avait un taux de fausses alarmes trop élevé. Nous avons donc introduit une version améliorée qui exploite un plus grand nombre de sorties intermédiaires du réseau de neurones. Cette nouvelle approche a permis d'améliorer significativement les performances. Il reste cependant un écart important par rapport aux résultats obtenus sans changement de domaine.

Organisation du manuscrit

Le manuscrit est divisé en quatre chapitres :

- Le chapitre 1 présente l'état de l'art sur les réseaux de neurones et le principe de l'apprentissage. Il présente ensuite les réseaux de neurones convolutifs, dédiés au traitement d'image et la description d'un ensemble de solutions numériques permettant de faciliter l'entrainement des réseaux de neurones. Enfin, ce chapitre se termine par une brève introduction au principe du *fine-tuning* et de quelques un des défis posés par l'utilisation des CNN pour la classification d'images. Nous ne couvrirons ici que les éléments nécessaires à la compréhension des résultats des chapitres 3 et 4.
- Le chapitre 2 présente le principe de l'imagerie infrarouge et son utilisation pour les tâches de DRI militaire. Nous commençons par présenter le rayonnement infrarouge et les caméras thermiques qui permettent de l'observer. Nous présentons ensuite différents exemples d'utilisation de l'imagerie infrarouge pour des applications civiles et militaires. Puis nous introduisons les ensembles de données qui seront utilisés dans les chapitres 3 et 4. Nous terminons ce chapitre par la présentation des principaux problèmes liés à l'utilisation de l'apprentissage profond pour des applications militaires, principalement pour la RI en infrarouge.
- Le chapitre 3 sera consacré à notre première contribution, le cfCNN. Dans un premier temps, nous adressons le cas de l'utilisation de modèles de CNN issus de l'état de l'art pour la RI en infrarouge dans des images réelles après un entrainement sur données simulées. Puis nous introduisons le cfCNN et ses performances sur des données infrarouges réelles après avoir été entrainé sur des données simulées. Enfin, nous présentons une étude de la robustesse du cfCNN face a des entrées perturbées.
- Le chapitre 4 se focalisera sur la détection d'anomalie de classification pour un CNN et la présentation des deux outils utilisés. Dans un premier temps, nous traitons la détection d'anomalies de classification dans un cas ou le CNN est entrainé et testé sur des données simulées. Nous évaluons la réponse du détecteur face à de nouvelle classes, face à des images de fonds sans cible ou face à des exemples adverses. Puis, dans un deuxième temps, nous

traitons le cas ou le CNN est entrainé sur des données simulées et testé sur des données réelles, et présentons notre approche basé sur une cascade de détecteur d'anomalies.

Chapitre 1

Réseaux de Neurones pour la classification

Ce chapitre à pour but de présenter les éléments théoriques de l'apprentissage profond qui seront utilisés dans la suite du manuscrit. Nous avons regroupé ici de façon synthétique les éléments clef qui permettront de comprendre les contributions de cette thèse.

Dans la section 1.1, nous nous intéresserons aux éléments de base qui constituent les réseaux de neurones et la façon de les entrainer dans la section 1.2. Nous restreindrons nos explications à l'application de ces réseaux aux problèmes de classification. Nous présenterons ensuite dans la section 1.3 les réseaux de neurones convolutifs pour la classification d'images.

Dans la section 1.4 de ce chapitre, nous présenterons les méthodes permettant d'optimiser le processus d'entrainement de ces réseaux de neurones. Nous présenterons ensuite dans la section 1.5 certaines limites des réseaux de neurones convolutifs avant de récapituler nos observations dans la section 1.6.

1.1 Réseaux de neurones

1.1.1 Définition du problème de classification

Nous considérons un problème de classification dans lequel nous disposons d'un ensemble de données X regroupant T tenseurs à n dimensions $\{\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(t)}, , \boldsymbol{x}^{(T)}\}$ avec $T \in \mathbb{N}$ auxquels nous voulons associer à chacun une classe d'appartenance c parmi $n_{classes}$.

L'objectif de la classification sera de séparer X en associant à chaque $\boldsymbol{x}^{(t)}$ à une classe c. Cela revient à trouver une fonction de décision f qui prends en entrée un vecteur $\boldsymbol{x}^{(t)}$ et dont la sortie $\hat{y}$ correspond à la classe c à laquelle appartient $\boldsymbol{x}^{(t)}$. Une illustration de problème est présenté dans la figure 1.1, avec $c = 3$. Nous allons nous intéresser à quelques solutions possibles pour ce problème, qui constituent les origines des algorithmes traités dans ce chapitre.

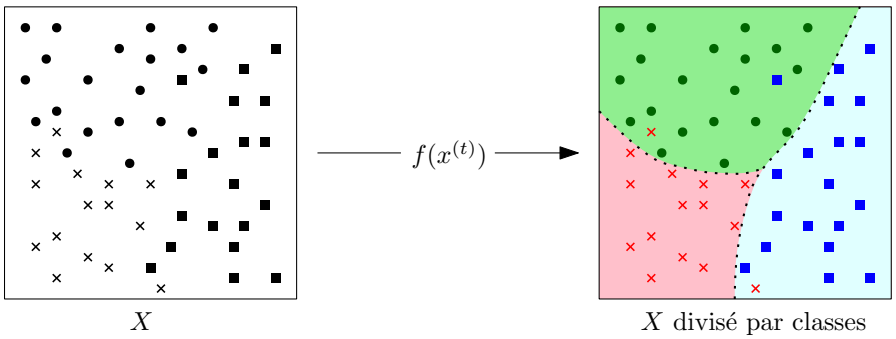


FIGURE 1.1 – Illustration du problème de classification

1.1.2 Perceptron

L'origine des réseaux de neurones modernes se trouve dans le perceptron [91]. Cet algorithme de classification, présenté en figure 1.2, peut se modéliser sous la forme d'une fonction de décision f qui prends en entrée un vecteur, ou tenseur, $\boldsymbol{x} = [x_1, ..., x_n]^\top$ et dont la sortie sera un scalaire $\hat{y}$ dont la valeur servira à déterminer la classe d'appartenance de l'entrée $\boldsymbol{x}$.

Cette fonction de décision f peut se décomposer en deux étapes :

- Une pré-activation a qui correspond à la combinaison linéaire de $\boldsymbol{x}$ avec un vecteur de poids $\boldsymbol{w} = [w_1, ..., w_n]^\top$, tel que $a(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x}$.
- Une activation h qui prends en entrée la sortie de la pré-activation en lui ajoutant un terme de biais b et dont la sortie correspondra à $\hat{y}$ tel que $\hat{y} = h(a(\boldsymbol{x})) = h(\boldsymbol{w}^\top \boldsymbol{x} + b)$.

Pour simplifier la notation nous désignerons par $\boldsymbol{\theta}$ le vecteur des paramètres : $\boldsymbol{\theta} = [b, w_1, ..., w_n]^\top$ de manière a pouvoir écrire $\hat{y} = f(\boldsymbol{x}; \boldsymbol{\theta}) = h(a(\boldsymbol{x})) = h(\boldsymbol{w}^\top \boldsymbol{x} + b)$.

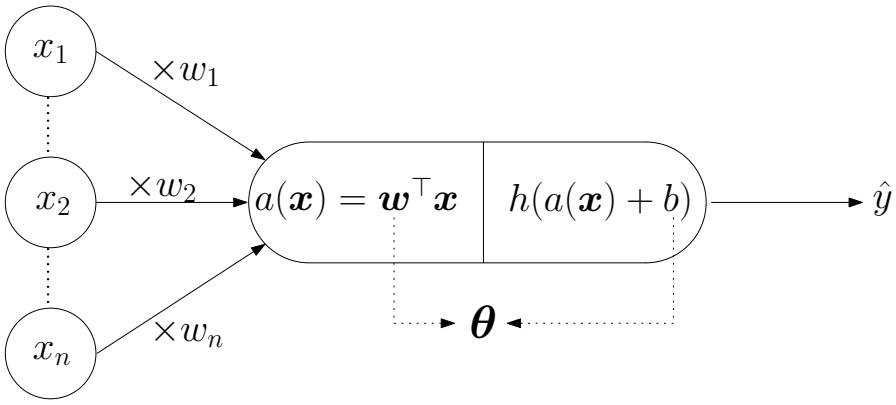


FIGURE 1.2 – Perceptron.

Dans la forme originelle du perceptron, h était la fonction de Heavyside, présentée dans la figure 1.3 :

$$\text{Heavyside}(x) = \begin{cases} -1 & si \quad x \leq 0 \\ 1 & si \quad x > 0 \end{cases} \tag{1.1}$$

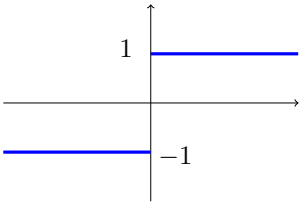


FIGURE 1.3 – Fonction de Heavyside.

L'utilisation du perceptron est limitée à des problèmes de classifications binaires où l'une des classe correspondra à $\hat{y} = 0$ et l'autre à $\hat{y} = 1$. De plus, le perceptron ne peut être utilisé que sur des problèmes de classifications ou les variables d'entrées X sont linéairement séparables.

La forme du perceptron présente certaines similarités avec les cellules du cerveau humain, ou neurones, et prennent donc par extension le nom de "neurones artificiels". En effet, comme on peut le voir en comparant la figure 1.2 à la figure 1.4, les éléments du perceptron peuvent dans une certaine mesure être assimilés à certains attributs des neurones biologiques : la couche d'entrée par laquelle sont introduites les variables $\boldsymbol{x}$ peut s'assimiler aux dendrites, le corps du perceptron au nucléus, et la sortie $\hat{y}$ à l'axone.

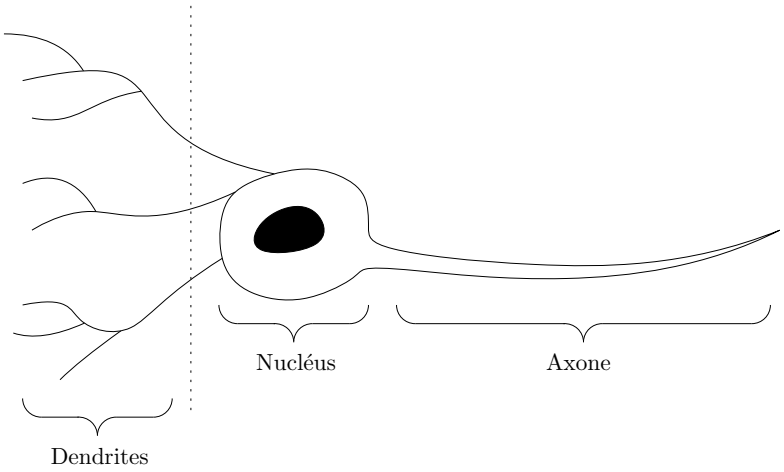


FIGURE 1.4 – Schéma simplifié d'un neurone biologique.

Comme pour les neurones du cerveau humain, la capacité du perceptron simple est limitée. Cette capacité correspond à la variété de fonctions qu'un perceptron peut approximer. Il est cependant possible de combiner plusieurs perceptrons entre eux pour traiter des problèmes plus complexes.

1.1.3 Perceptron multi-couches

En reliant plusieurs perceptrons ou neurones artificiels, il est possible de former un perceptron multi-couches ou MLP pour *Multi-Layer Perceptron*. Ces modèles permettent de traiter des problèmes plus complexes comme la classification multi-classes. Les neurones du MLP sont regroupés en une série couches successives reliées entre elles pour former un graphe orienté visible dans la figure 1.5.

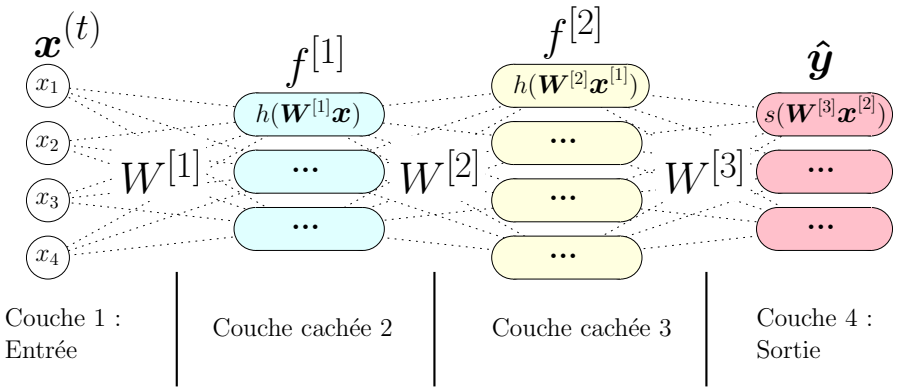


FIGURE 1.5 – Perceptron multi-couches.

Un MLP peut aussi être appelé réseau de neurones. Comme pour les perceptrons simples, les MLP prennent en entrée un vecteur $\boldsymbol{x}^{(t)}$ qui est transformé par les neurones des couches intermédiaire, appelées couches cachées, jusqu'à la dernière couche dite couche de sortie. C'est cette dernière couche qui servira à déterminer la classe de sortie de $\boldsymbol{x}^{(t)}$. Une couche cachée k ne sera directement reliée qu'à la couche qui la précède et à la couche qui la suit dans le réseau. Les sorties de la couche $k-1$ seront les entrées de la couche k et les sorties de k seront les entrées de $k+1$. Enfin, dans chaque couche, les neurones fonctionnent comme le perceptron décrit dans la figure 1.2. Chaque neurone d'une couche est donc connecté à l'ensemble des neurones de la couche précédente par des liaisons pondérées par son vecteur de poids $\boldsymbol{w}$ et un terme de biais b. Ce dernier est absent sur la figure 1.5 pour alléger la notation.

Ainsi pour un neurone quelconque dans une couche k du MLP nous pouvons exprimer sa pré-activation comme étant :

$$a^{[k]}(\boldsymbol{x}^{(t)}) = \boldsymbol{W}^{[k]} h(a^{[k-1]}(\boldsymbol{x}^{(t)})) + \boldsymbol{b}^{[k]} \tag{1.2}$$

Et son activation totale :

$$f^{[k]}(\boldsymbol{x}^{(t)}; \theta^{[k]}) = h^{[k]}(a^{[k]}(\boldsymbol{x}^{(t)})) \tag{1.3}$$

Le terme $\boldsymbol{W}^{[k]}$ est une matrice $n \times m$ avec m le nombre de neurones de la couche $k-1$ et n le nombre de neurones de la couche k où chaque ligne correspond à un vecteur $\boldsymbol{w}$ des poids de chaque neurone de la couche. Le terme $\boldsymbol{b}^{[k]}$ correspond au vecteur regroupant l'ensembe des

biais des neurones de la couche k.

À l'image de la notation utilisée pour le perceptron nous désignerons par $\boldsymbol{\theta}$ l'ensemble des paramètres et biais utilisés dans le MLP *i.e.* $\boldsymbol{\theta} = \boldsymbol{W}^{[1]}, \boldsymbol{b}^{[1]}, ..., \boldsymbol{W}^{[k]}, \boldsymbol{b}^{[k]}, ..., \boldsymbol{W}^{[L]}, \boldsymbol{b}^{[L]}$ avec L le nombre de couches présentes dans le MLP.

Chaque neurone utilise aussi une fonction d'activation h comme pour le perceptron simple. Cette fonction d'activation est généralement la même pour l'ensemble des neurones des couches cachées. Nous reviendrons sur le choix d'une fonction d'activation dans la section 1.4.2 de ce chapitre.

Nous pouvons voir que la valeur de $f(\boldsymbol{x}; \boldsymbol{\theta})$ est le résultat de la propagation vers l'avant de l'information issue du vecteur $\boldsymbol{x}$, transformé par les différents neurones du réseau. Il n'y a ni cycle, ni boucle dans le chemin de circulation de l'information qui "avance" de $\boldsymbol{x}$ vers la couche de sortie. En raison de ce comportement, ces réseaux sont appelés "Réseaux de neurones à propagation vers l'avant" ou *feedforward neural networks*.

La dernière couche d'un réseau de neurones présente certaines caractéristiques liées au problème de classification multi-classes. Cette sortie, que nous appelerons $f^[L]$, comprend un neurone par classe. Les neurones de cette couche utilisent une fonction d'activation spécifique, la fonction Softmax [16]. Pour un vecteur $\boldsymbol{x} = [x_i]_{i=1..n}$ la fonction Softmax s'écrit :

$$\text{Softmax}(x_i) = \frac{\exp x_i}{\sum_{k=0}^{n} \exp x_k} \tag{1.4}$$

Cette fonction est strictement positive et $\sum_{i=0}^{N} \text{Softmax}(x_i) = 1$.

Lorsqu'elle est appliquée aux problèmes de classification, cette fonction d'activation, rappelant la régression logistique et le modèle LOGIT, permet de transformer les sorties des neurones de la dernière couche en un vecteur dont chaque élément est contraint dans l'intervalle $[0, 1]$.

On peut alors l'interpréter comme $p(\hat{y} = c|\boldsymbol{x})$, soit la probabilité que l'entrée $\boldsymbol{x}$ du MLP appartienne à la classe c. En regroupant l'ensemble des sorties de la dernière couche, on obtient alors un vecteur regroupant l'ensemble des probabilités d'appartenance de $\boldsymbol{x}$ à l'une des $n_{classes}$ du problème :

$$f^{[n-1]}(\boldsymbol{x}^{(t)}) = [p(\hat{y} = 1|\boldsymbol{x}), ..., p(\hat{y} = n_{classes}|\boldsymbol{x})]^\top \tag{1.5}$$

La prédiction finale du MLP, soit la détermination de la classe d'appartenance de $\boldsymbol{x}$ se fera en sélectionnant la valeur de $f(\boldsymbol{x}; \boldsymbol{\theta})$ de la façon suivante :

$$f(\boldsymbol{x}; \boldsymbol{\theta}) = \hat{y} = \arg\max_{c}(\boldsymbol{s}(\boldsymbol{x})) \tag{1.6}$$

L'intérêt du MLP réside dans sa capacité à pouvoir approximer n'importe quelle fonction continue sur un sous ensemble compact de $\mathbb{R}^n$ avec seulement deux couches, une couche cachée et une couche de sortie, possédant un nombre fini de neurones. Sa capacité est donc bien plus importante que celle du perceptron. Il s'agit du théorème d'approximation universelle [26] [53]

[77]. A noter cependant que ce théorème n'est valable que si les activations des neurones de la couche cachées respectent les conditions qui seront détaillées dans la section 1.4.2. Il est donc théoriquement possible de modéliser une fonction de décision pour tout type de problème de classification à l'aide d'un MLP. On parlera d'apprentissage profond ou *Deep-learning* pour les modèles de réseaux de neurones possédant plus d'une couche cachée. De façon générale, on privilégiera les architectures profondes avec plusieurs couches cachée par rapport à des architectures avec un faible nombre de couches cachées contenant un grand nombre de neurones [42, p. 198]

1.2 Principe de l'apprentissage

1.2.1 Objectif et hyperparamètres

Pour que la fonction de décision f du MLP puisse correctement associer les entrées $\boldsymbol{x}^{(t)}$ a la bonne classe, il faut disposer des bons paramètres $\boldsymbol{\theta}$. Ces paramètres peuvent être déterminés par ce qu'on appelle entrainement ou apprentissage. Nous nous plaçons ici dans un cas où nous disposons de données d'entrainement X_{train} qui est un sous ensemble de X regroupant une sélection aléatoire de vecteurs de X tels que $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(n_{train})}$ avec $n_{train} \in \mathbb{N}$. Chaque vecteur sera associé à une classe via un ensemble de labels $Y_{train} : y^{(1)}, ..., y^{(n_{train})}$. Chaque label $y^(i)$ de Y_{train} s'exprimera sous la forme d'un vecteur $\boldsymbol{y}^{(i)} = [0, ..., 0, 1, 0, .., 0]$ avec la c-ieme composante de y égale à 1. Nous pouvons donc utiliser ces couples entrée-label $\{\boldsymbol{x}^{(t)}, y^{(t)}\}$ pour apprendre la relation entre une entrée et sa classe *i.e.* déterminer les valeurs de $\boldsymbol{\theta}$ permettant de résoudre le problème de classification.

La recherche de $\boldsymbol{\theta}$ se fera via une minimisation du risque empirique à l'aide des prédiction du modèle sur X_{train} via une fonction J appelée fonction de coût :

$$\arg\min_{\theta} J(\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}; \boldsymbol{\theta}) \tag{1.7}$$

Avec :

$$J(\boldsymbol{x}^{(t)}, \boldsymbol{y}^{(t)}; \boldsymbol{\theta}) = \frac{1}{\text{Card}(X_{train})} \sum_{t} E(f(\boldsymbol{x}^{(t)}; \boldsymbol{\theta}), \boldsymbol{y}^{(t)}) \tag{1.8}$$

L'idée de cette minimisation est d'utiliser une fonction de substitution, appelée fonction fonction d'erreur E qui mesure l'écart entre la prédiction du réseau de neurones et le label attendu sur les données d'entrainement. En minimisant J, nous minimisons donc E et agissons sur f via $\boldsymbol{\theta}$ pour faire tendre la prédiction $\hat{y}$ de chaque $\boldsymbol{x}^{(t)}$ de X_{train} vers le label connu $y^{(t)}$.

Lorsque J atteint un minimum global, on suppose alors que f représente correctement la relation entre les éléments de X_{train} et Y_{train}. Nous supposons alors qu'elle peut être utilisée pour classer de nouvelles entrées $\boldsymbol{x} \notin X_{train}$. La minimisation de J est un procédé itératif dont les détails seront présentés dans la section 1.2.2.

L'opération décrite dans l'équation 1.7 qui illustre le principe d'apprentissage n'affecte que les valeurs de $\boldsymbol{\theta}$. Toute autre grandeur qui influe sur l'apprentissage sans être affectés par la

minimisation de la fonction L seront qualifiés d'hyperparamètres. Le nombre de couches ou de neurones par couches sont des hyper-paramètres. Nous verrons dans la section suivante quelques exemples d'hyperparamètres qui ont une influence sur le mécanisme d'apprentissage.

1.2.2 Apprentissage par rétro-propagation

Descente de gradient

Une des façons de minimiser J couramment utilisée pour les réseaux de neurones est d'effectuer ce qu'on appelle une rétro-propagation de l'erreur. Un algorithme d'optimisation couramment utilisé pour la rétro propagation appliquée aux réseaux de neurones est la descente de gradient ou GD [128].

Le but de la GD est de mettre à jour de façon itérative les poids du réseau en utilisant le gradient de la fonction de coût J à partir des données d'entrainement X_{train}. Quand J est évaluée sur des éléments de X_{train} nous l'appellerons J_{train}. Le déroulé de la GD est décrit dans l'algoritme 1 et illustré dans la figure 1.6.

Algorithm 1 Descente de gradient

Require: X_{train}, Y_{train}
Require: $n_{iterations}$ ▷ Détermine le nombre d'itération de la GD
Require: ϵ ▷ Détermine la vitesse de la GD
  Initialiser θ
  for $n_{iterations}$ **do**
    $\nabla_\theta J_{train} \leftarrow \frac{\delta}{\delta \boldsymbol{\theta}} \frac{1}{\text{Card}(X_{train})} \sum_{t=1}^{n_{train}} E(f(\boldsymbol{x}^{(t)}, \boldsymbol{\theta}), y^{(t)})$ ▷ Calcul du gradient de la fonction de coût E
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \nabla_\theta J_{train}$ ▷ Mise à jour des poids du réseau
  end for

Le calcul du gradient est effectué pour l'ensemble des couples $\boldsymbol{x}^{(t)}, y^{(t)}$ disponibles à l'entrainement. Un passage complet sur les données d'entrainement sera appelé époque. En pratique, le nombre d'itérations de l'algorithme de la GD sera souvent appelé nombre d'époques. Il s'agit d'un hyperparamètre.

Le coefficient ϵ est appelé taux d'apprentissage. Il détermine l'amplitude des étapes de la GD et influe donc sur la vitesse de l'apprentissage. Il s'agit d'un hyperparamêtre fixé empiriquement qui est généralement compris entre 1 et 10^{-6} [12]. Nous verrons dans la section 1.4.4 qu'il est possible de faire varier ce taux au cours des itérations pour optimiser l'apprentissage.

Pour l'instant nous considérons simplement que chaque poids du vecteur $\boldsymbol{\theta}$ est initialisé aléatoirement. Notons simplement qu'initialiser $\boldsymbol{\theta}$ à 0 ne fonctionne pas car la valeur de $\nabla_\theta J_{train}$ serait aussi nulle et l'algorithme de GD ne pourrais pas démarrer. Le choix de la distribution et son impact sur l'apprentissage sera détaillée dans la section 1.4.3.

Nous pouvons voir pendant l'apprentissage que l'algorithme de la GD re-propage le résultat de fonction de coût pour mettre à jour les poids du réseau. En soustrayant à chaque poids du

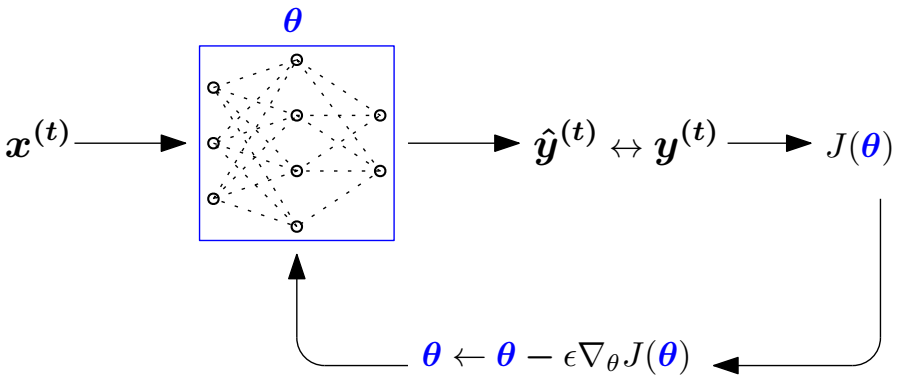


FIGURE 1.6 – Principe de la rétro propagation pour l'entrainement des réseaux de neurones.

réseau la grandeur $J_{train}(f(\boldsymbol{x}^{(t)}, \boldsymbol{\theta})$, on fait tendre de façon itérative la valeur de la fonction J_{train} vers un minimum.

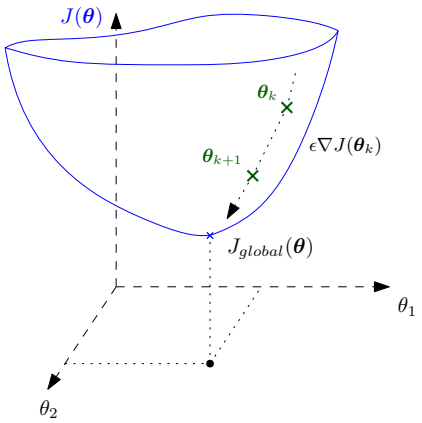
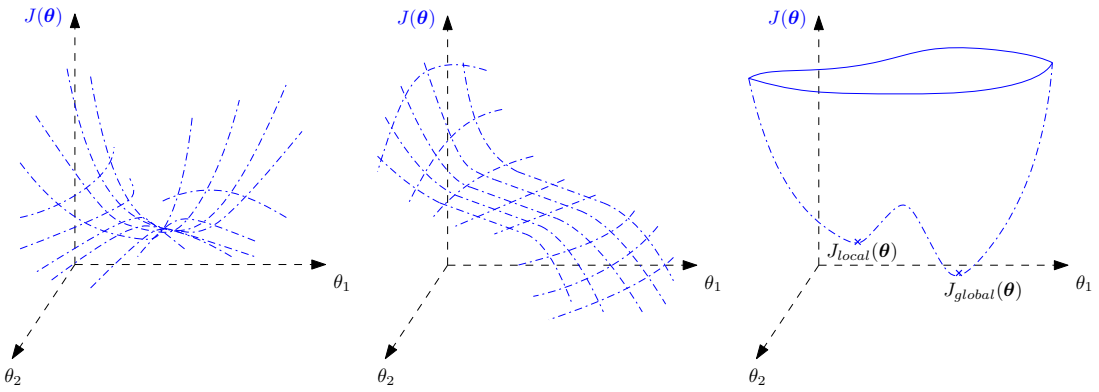


FIGURE 1.7 – Représentation simplifiée de la descente de gradient.

Intuitivement, et comme illustré dans la figure 1.7, cela revient à se déplacer sur la surface que définit la fonction d'erreur dans l'espace de $\boldsymbol{\theta}$ vers son minimum. Comme on utilise la valeur négative du gradient, ou $-\nabla_\theta J_{train}$, calculé sur l'ensemble des exemples d'entrainement, la direction du déplacement correspondra à la pente la plus forte sur la surface de J_{train}. Le cas présenté ici est un cas simple avec $\boldsymbol{\theta} = \{\theta_1, \theta_2\}$ et ou la fonction d'erreur ne présente qu'un minimum global matérialisé par $\boldsymbol{\theta}_{ideal}$ dans la figure1.7.

Néanmoins, la surface J_{train} est rarement parfaitement convexe. Elle peut présenter des zones particulières, présentées dans la figure 1.8 telles que les minimums locaux, évoqués précédemment, mais aussi des points cols, ou des plateaux. Comme la GD progresse vers la pente la plus forte l'apprentissage peut se retrouver entrainé vers une zone de J_{train} qui est loin du minimum global et dont il est difficile voir impossible de s'extraire même en jouant sur le taux d'apprentissage ϵ.



(a) Col. (b) Plateau. (c) Minimum local.

FIGURE 1.8 – Illustration des imperfections de la surface représentée par J_{train} simplifiée en $J(\theta)$ ici.

De plus, Le cout calculatoire associé à la grandeur J_{train} dans l'algorithme de la GD est directement proportionnel à n_{train} Pour un jeu de données X_{train} de grande taille, ce coût peut devenir très important et impacter l'apprentissage.

Descente de gradient stochastique

Une alternative à la GD est d'utiliser une variante appelée descente de gradient stochastique, ou SGD, décrite par l'algorithme 2. Contrairement à la descente de gradient standard, le gradient $nabla_\theta J_{train}$ est évalué sur un seul exemple d'entrainement et non sur la totalité de X_{train}. Par conséquent, la direction emprunté par l'algorithme de la SGD varie à chaque itération et ne suis pas la pente la plus forte. De plus, J_{train} est évalué pour un seul exemple et donc son aspect change à chaque itération de SGD. Ces deux éléments donnent à la SGD un caractère aléatoire qui peut lui permettre d'éviter les écueils décrits dans la figure 1.8 ou se sortir d'une zone de J_{train} où la GD serait restée coincée.

Algorithm 2 Descente de gradient stochastique

Require: X_{train}, Y_{train}
Require: $n_{iterations}$ ▷ Détermine le nombre d'itération de la GD
Require: ϵ ▷ Détermine la vitesse de la GD
  Initialiser θ
  for $n_{iterations}$ **do**
    for $t \leftarrow 1, n_{train}$ **do** ▷ On travail sur l'ensemble des exemples d'entrainement
      Sélection aléatoire d'un seul couple$\{x^{(t)}, y^{(t)}\}$ depuis X_{train}
      $\nabla_\theta J_{train} \leftarrow \frac{\delta}{\delta \boldsymbol{\theta}} E_{train}(f(\boldsymbol{x}^{(t)}, \boldsymbol{\theta}), y^{(t)})$ ▷ Calcul du gradient de la fonction de coût J
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \nabla_\theta J_{train}$ ▷ Mise à jour des poids du réseau
    end for
  end for

Cependant, comme la SGD estime J_{train} sur un seul exemple d'entrainement, elle peut s'avérer longue à converger vers une solution car il faut évaluer $\nabla_\theta J_{train}$ pour chaque exemple du jeu de données X_{train}. Or, contrairement à la GD où le calcul de E_{train} peut être parallélisé, la SGD est un processus séquentiel.

Descente de gradient par mini-lots

Pour jeu de donnée avec un grand nombre d'exemples d'entrainement, on préférera utiliser une descente de gradient par lots décrite dans par l'algorithme 3. L'approche par mini-lot est un compromis entre la GD classique et sa variante stochasitque. La GD par lot permet de converger vers une solution pour $\boldsymbol{\theta}$ plus rapidement qu'avec une SGD en estimant le gradient sur un tirage aléatoire d'éléments de X_{train} plutôt que sur un unique couple $\{\boldsymbol{x}^{(t)}, y^{(t)}\}$. Le tirage des lots est répété en sélectionnant des couples $\{\boldsymbol{x}^{(t)}, y^{(t)}\}$ différents pour couvrir l'ensemble de l'espace d'apprentissage.

De plus, la GD par lot conserve en partie le caractère aléatoire de la SGD. Le gradient étant calculé sur un sous ensemble de X_{train}, la direction de $\nabla_\theta J_{train}$ ainsi que l'allure de J_{train} changera à chaque itération. Enfin, comme pour la GD, il est possible de calculer le gradient pour chaque exemple d'un lot en parallèle ce qui peut accélérer le traitement numérique de la descente de gradient par mini-lot si le matériel utilisé le permet.

Algorithm 3 Descente de gradient par lots

Require: X_{train}, Y_{train}
Require: $n_{iterations}$ ▷ Détermine le nombre d'itérations de la GD
Require: m ▷ Taille du mini-lot
Require: ϵ ▷ Détermine la vitesse de la GD
  Initialiser θ
  for $n_{iterations}$ **do**
    for $n_{etapes} \leftarrow 1, \lfloor n_{train}/m \rfloor$ **do**
      Sélection aléatoire d'un lot de couples $\{x^{(t)}, y^{(t)}\}$ depuis X_{train}
      $\nabla_{lot} J_{train} = \frac{1}{m} \sum_{t=1}^{m} \nabla_\theta E(\hat{y}, y^{(t)})$ ▷ Calcul du gradient pour un lot
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \nabla_{lot} J_{train}$ ▷ Mise à jour des poids du réseau
    end for
  end for

Le choix de la taille de lot m est un hyperparamètre. Un m élevé permettra une meilleure estimation du gradient sur l'ensemble des données d'entrainement tandis qu'une petite valeur de m permettra de mieux généraliser à de nouvelles données hors de l'ensemble X_{train}.

Idéalement, la GD par lot devrait pouvoir converger vers une solution globale ce qui impliquerait que J soit convexe. Or, choisir la GD par lot ne modifie pas J_{train} et n'empêche pas de rencontrer des points cols, des plateaux et des minimums locaux. Cette convexité dépends de la nature de J. Néanmoins, Choromanska et al. [23] ont montré que, pour les fonctions J que nous présenterons dans la section 1.2.3, ces solutions locales se trouvent dans une zone bien

définie autour du minimum global. Un autre point soulevé dans cet article est que cette zone dans laquelle se trouvent les minimums locaux diminue lorsque la taille du réseau augmente. Il y a donc une forte probabilité pour qu'en cas de convergence via une descente de gradient, le minimum local atteint soit proche du minimum global.

1.2.3 Choix de la fonction d'erreur

Pour les réseaux de neurones appliqués aux problèmes de classification, l'usage est d'utiliser une fonction q'erreur E qui utilise les sorties probabilistes $\boldsymbol{s}$ plutôt que la prédiction finale de la classe. Il est plus difficile d'utiliser la prédiction finale $\hat{y}$ discrète pour la descente de gradient. Nous disposons alors d'un vecteur qui représente les distributions de probabilités d'appartenance d'une entrée $\boldsymbol{x}$ à chacune des classes. Les classes étant supposées disjointes, $\boldsymbol{y}$ peut être interprété comme une distribution de probabilité idéale avec $p(y = c) = 1$ et $p(y) = 0$ sinon.

Mesurer l'écart entre la prédiction du réseau de neurones et le label $\boldsymbol{y}$ pendant l'apprentissage reviendra donc a comparer deux distributions de probabilités entre elles et faire tendre la prédiction du modèle vers la distribution idéale correspondant au label. Il est donc d'usage d'utiliser pour E l'entropie croisée qui pour deux distributions de probabilités p et q pour une variable aléatoire X s'écrit :

$$H(p, q) = D_{\mathrm{KL}}(p \parallel q) + H(p). \tag{1.9}$$

Avec $H(p)$ l'entropie de Shannon de la distribution p qui représente mesure la quantité d'information contenue par la variable X lorsqu'elle suit la distribution p :

$$H(p) = \mathbb{E}_p[-\log p]. \tag{1.10}$$

Et $D_{\mathrm{KL}}(p \parallel q)$ la divergence de Kullback-Leibler entre p et q, mesurant la différence entre deux probabilités de distributions et s'écrit :

$$D_{\mathrm{KL}}(p \parallel q) = \mathbb{E}_p[-\log \frac{p}{q}]. \tag{1.11}$$

En combinant les équations 1.10 et 1.11 on peut simplifier l'équation 1.9 de la façon suivante et en supposant que p et q sont discrètes, il est possible de simplifier l'expression de l'entropie croisée :

$$\begin{aligned} H(p, q) &= \mathbb{E}_p[-\log \frac{p}{q}] + \mathbb{E}_p[-\log p] \\ &= \mathbb{E}_p[-\log q] + \mathbb{E}_p[\log p] + \mathbb{E}_p[-\log p] \\ &= \mathbb{E}_p[-\log q] + \mathbb{E}_p[\log p] - \mathbb{E}_p[\log p] \\ &= \mathbb{E}_p[-\log q] \\ &= -\sum_{\boldsymbol{x} \in X} p(x) \log q(x) \end{aligned} \tag{1.12}$$

En remplaçant dans l'équation 1.12 la distribution p par $\boldsymbol{y}^{(t)}$ et q par $\boldsymbol{s}(\boldsymbol{x}^{(t)})$ on obtient l'expression finale de E :

$$E = H(\boldsymbol{y}^{(t)}, \boldsymbol{s}(\boldsymbol{x}^{(t)})) = -\boldsymbol{y^{(t)}} \log \boldsymbol{s}(\boldsymbol{x}^{(t)}). \tag{1.13}$$

Avec le remplacement de p par $\boldsymbol{y}$ nous pouvons aussi interpréter la minimisation de $H(p, q)$ dans l'équation 1.9 comme la minimisation de la divergence D_{KL} entre les distributions des labels et des sorties probabilistes du réseau. En effet la distribution des labels est constante vis à vis de $\boldsymbol{\theta}$ et son entropie sera donc constante aussi. L'apprentissage via la descente de gradient en utilisant l'entropie croisée revient donc à faire tendre la distribution des sorties du réseau vers celle des labels.

1.2.4 Sur-apprentissage et sous-apprentissage

Nous avons vu dans la section qui précède que l'apprentissage des réseaux de neurones se faisait grâce à la GD par lots. Pendant cet apprentissage, J_{train} est régulièrement évaluée, ce qui nous permet de juger la capacité du modèle à classer les données de X_{train}. Cependant, l'intéret de l'entrainement d'un réseau de neurones est de pouvoir utiliser la relation apprise à partir de X_{train} sur de nouvelles données. Pour juger de la qualité de l'apprentissage nous utiliserons alors un sous ensemble de X_{train} que nous appellerons ensemble de validation ou X_{valid} ainsi que les labels associés aux éléments de ce nouveau sous-ensemble Y_{valid}. Les éléments qui constituent X_{valid} sont sélectionnés aléatoirement pour couvrir au mieux l'espace d'apprentissage. Pendant l'apprentissage, les données de X_{valid} sont mises à l'écart et ne sont jamais utilisées pour la recherche de $\boldsymbol{\theta}$. Elles serviront a évaluer la fonction de coût J sur X_{valid} entre chaque étape de la GD par lot pour déterminer la performance du modèle sur de nouvelles données. Nous l'appelerons alors J_{train}.

Habituellement, J_{train} est évaluée en continu pendant l'entrainement et J_{valid} à la fin de chaque époque d'entrainement. Le suivi et la comparaison des valeurs prise par J_{train} et J_{valid} permettra aussi de détecter les cas de sur-apprentissage et sous-apprentissage que nous présenterons dans la section 1.2.4. La la qualité de l'apprentissage dépendra de deux conditions sur J_{train} et J_{valid} qui sont [42, p. 110] :

- J_{train} doit décroître et atteindre une valeur faible, si possible proche de zéro.
- L'écart $|J_{train} - J_{valid}|$ doit être faible.

Dans un cas idéal ou l'apprentissage se déroule correctement les valeurs prise par J_{train} et J_{valid} vont décroître conjointement pendant l'apprentissage comme illustré dans la figure 1.9. Cela signifie que les prédictions du réseau de neurones se rapprochent des labels attendus sur des données différentes de X_{train}. On dit alors que le modèle généralise bien sur de nouvelles données qui sont ici représentées par X_{valid}.

Lorsqu'une des conditions listées n'est pas respectée, deux états sont possibles : le sur-apprentissage et le sous-apprentissage, illustrées dans la figure 1.10. Le sous apprentissage se manifeste par une stagnation de J_{train} et J_{valid} comme illustré dans la figure 1.10a pendant

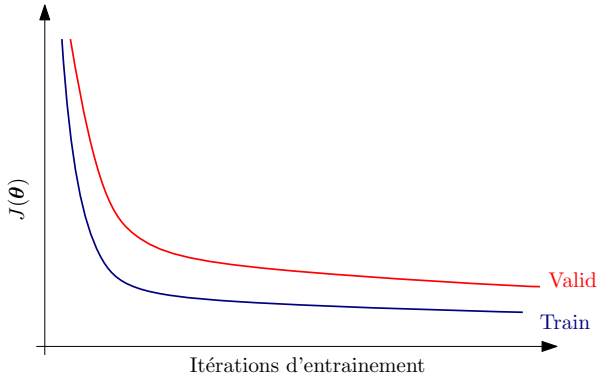
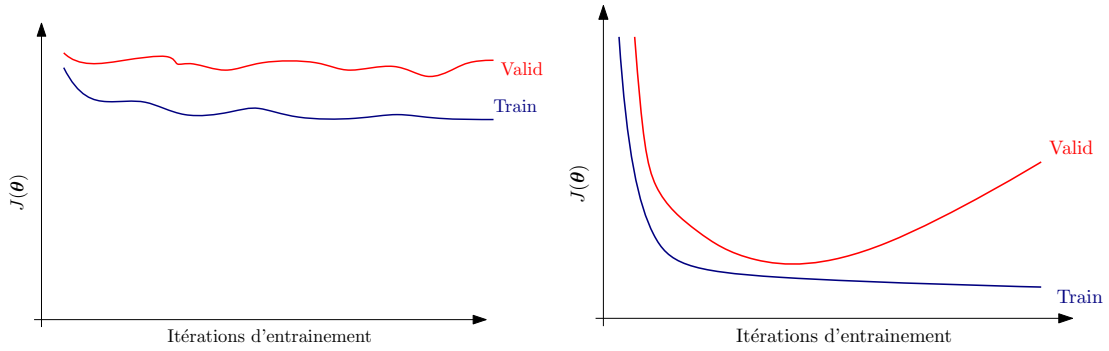


FIGURE 1.9 – Tracé théorique idéal de l'évolution de J_{train} et J_{valid} en fonction du nombre d'itération d'apprentissage effectué.

l'apprentissage. Le sur apprentissage se manifeste par une fonction de coût J_{train} décroissante pendant l'apprentissage mais un coût J_{valid} qui stagne à sa valeur de départ ou ré-augmente après avoir décru pendant quelques époques comme illustré dans la figure 1.10b.



(a) Sous-apprentissage. (b) Sur apprentissage.

FIGURE 1.10 – Tracé théorique pour les cas de sur-apprentissage et sous-apprentissage de l'évolution de $Jtrain$ et J_{valid} en fonction du nombre d'itération d'apprentissage effectuées.

L'existence des deux phénomènes est directement lié à la quantité de paramêtres $\boldsymbol{\theta}$ et donc la capacité des réseaux de neurones que nous avons évoqué dans la section 1.1.3. Même si celle-ci peut être presque illimitée en théorie, il faut pouvoir disposer des bons paramètres $\boldsymbol{\theta}$ pour approximer une fonction en particulier. Or, dans notre situation, ces paramètres $\boldsymbol{\theta}$ sont déterminés par apprentissage. La capacité réelle est donc dépendante du choix de l'algorithme d'optimisation, qui elle-même dépend de la structure du réseau que nous cherchons à optimiser. Ainsi, la capacité d'un réseau donné peut être insuffisante pour un problème donné et nous somme alors dans un cas de sous-apprentissage. Pour le contrer, une première solution est d'augmenter le nombre

de parmètres $\boldsymbol{\theta}$, et donc la capacité du réseau, en ajoutant des couches de neurones ou en agrandissant les couches existantes.

Le problème du sur-apprentissage est quant à lui plus généralement lié au processus d'optimisation et au choix des paramètres. La décroissance de J_{train} montre que la capacité du modèle est suffisante pour le problème. Cependant, le comportement de J_{valid} montre que les sorties du modèle ne sont pas correctes pour des données autres que X_{train}. Nous pouvons alors dire que le modèle ne généralise pas assez. Nous verrons dans la section 1.4 différentes méthodes pour résoudre les problèmes de sous-apprentissage et sur-apprentissage en facilitant le déroulement de l'optimisation par descente de gradient.

1.3 Réseaux de neurones convolutifs

Un des domaines dans lequel les réseaux de neurones sont couramment utilisés est celui de la vision par ordinateur et notamment le problème de la classification d'image. Dans la section 1.1 nous avons introduit les réseaux de neurones, et dans la section 1.2 les bases de l'entrainement de ces réseaux. Dans ces deux sections nous avons simplement décrit l'entrée $\boldsymbol{x}$ d'un réseau de neurones comme un tenseur quelconque que nous voulions associer à une classe.

Nous souhaitons introduire ici les formes de réseaux de neurones construits spécifiquement pour les cas ou $\boldsymbol{x}$ est une image. Dans cette partie nous allons expliquer pourquoi les MLP évoqués précédemment ne sont pas adaptés au cas du traitement d'image. Puis nous introduirons des réseaux spécifiques appelés réseaux de neurones convolutifs.

1.3.1 Limite du MLP simple pour la classification d'image

Les solutions de prédilection pour le problème de la classification d'images se basaient sur des méthodes d'extraction de caractéristiques. Les éléments extraits pouvaient ensuite être classées à l'aide de méthodes statistiques ou d'apprentissage. Parmi les techniques couramment utilisées pour l'extraction de caractéristiques il est possibles d'utiliser des méthodes basées sur des transformations globales de l'images comme la transformée de Fourier ou de Hough ou encore l'utilisation de filtres de Gabor. L'utilisation de descripteurs locaux est devenu l'approche privilégiée pour l'extraction de caractéristiques avec notamment SIFT ou *Scale Invariant Feature Transform* [82]. D'autres descripteurs locaux tels que SURF ou *Speeded Up Robust Features* [11], ou HOG pour *Histogram of Oriented Gradients* [31], peuvent aussi être utilisés.

Les informations issues de ces descripteurs sont généralement combinées à une méthode d'apprentissage pour la tâche de classification. Ce rôle pourra être rempli par un algorithme des plus proches voisins, un arbre de décision, un classifieur de Bayes naïf ou un SVM par exemple. Un réseau de neurones du type MLP 1.1.3 pourra tout à fait remplir ce rôle. La méthode choisie dépendra de la complexité du problème. Cependant, les MLP ne sont pas adaptés pour l'extraction de caractéristiques complexes dans les images. À l'exception de problèmes de classification avec

de petites images [28], ils sont difficilement utilisables sans descripteurs spécifiques en entrée du réseau.

En effet, sans utiliser de descripteur en amont du réseau, l'image d'entrée sera simplement étalée pour former un vecteur 1D. En procédant de la sorte nous ne conservons pas les relations bi-dimensionelles entre les différents pixels de l'image [74]. Sans cette capacité, le réseau risque de ne pas être capable de prendre en compte des éléments caractéristiques de l'images comme les arrêtes, les textures ou les coins.

De plus, le nombre de paramètres nécessaires pour classer des images peut augmenter de façon exponentielle. Pour une image de taille $x \times y$ en pixels avec b bandes, ou canaux, qui rentre dans un MLP avec n neurones d'entrée, le nombre de paramètres pour cette couche sera de $(x \times y \times b \times +1) \times n$ en prenant en compte les biais de chaque neurone. Sachant que les couches suivantes seront du même ordre de grandeur en termes de nombre de neurones, une augmentation de la taille de l'image d'entrée entrainera une augmentation significative du nombre de paramètres et donc la difficulté de faire converger ce modèle. Pour traiter des images à l'aide de réseaux de réseaux de neurones, il faut donc modifier les premières couches, notamment la couche d'entrée. Ces modifications doivent permettre de mieux prendre en compte les structures bi-dimensionelles complexes dans l'image.

1.3.2 Structure des CNN

Pour les problèmes de classification d'objets à partir d'image, on utilisera une forme de réseau de neurones particuliers appelés réseaux de neurones convolutifs ou CNN pour *Convolutionnal Neural Network*. La principale caractéristique de ces réseaux est, comme leur nom l'indique, l'ajout de couches de convolution en remplacement ou en complément de couches de neurones entièrement connectées présentes dans les MLP.

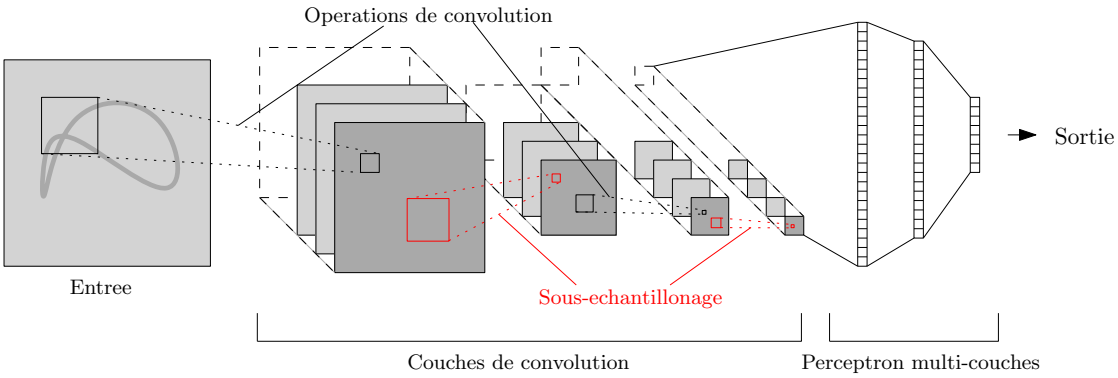


FIGURE 1.11 – Architecture d'un CNN.

Comme nous pouvons le voir dans la figure 1.11, les premières couches de ces CNN comprennent un enchainement de couches appelées couches de convolutions. Des couches de sous-

échantillonnage son aussi insérées entre ces couches de convolution. Cette combinaison remplie le même rôle que les descripteurs évoqués dans la section 1.3.1. Elles auront pour but d'extraire de l'image des caractéristiques qui seront ensuite passées en entrée d'un MLP pour la classification finale des images.

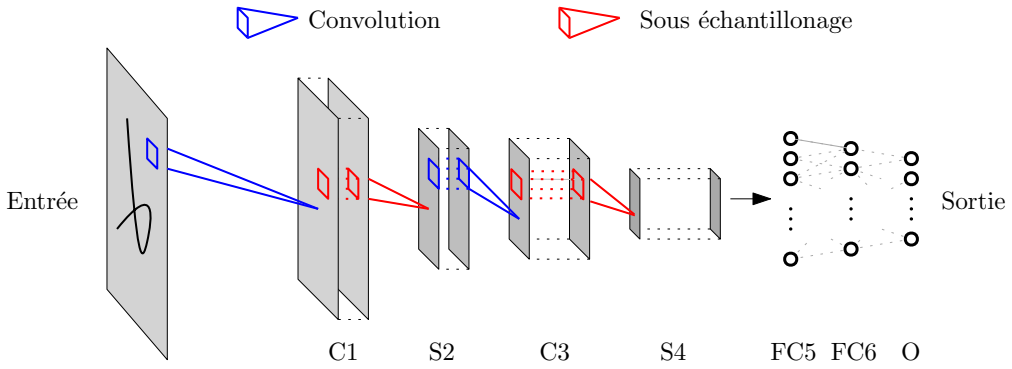


FIGURE 1.12 – Structure du réseau LeNet [74].

Le développement progressif des réseaux convolutifs pour les tâches de vision par ordinateur peut être retracé jusqu'aux annés quatre-vingt-dix avec notamment les travaux de Yann LeCun [74] et le réseau LeNet de la figure 1.12 dédié à la reconnaissance de caractères manuscrits. Il comprends au total 7 couches cachées : deux couches de convolution, appelées ici C1 et C3, deux couches de sous-échantillonnage, appelées S2 et S4 et deux couches denses, appelées FC5 et FC6, combinées à une couche de sortie O pour former le MLP final.

Les performances de ces premiers réseaux convolutifs étaient limités par la puissance de calcul des ordinateurs de l'époque, notamment pour traiter des problèmes avec un grand nombre de classes ou des images de grande taille. Il faudra attendre 2012 avec l'arrivée de l'architecture de réseau convolutif baptisée AlexNet [69] pour que ces réseaux convolutifs commencent réellement à devenir la solution de référence pour les tâches de vison par ordinateur. Ce changement est notamment dû à l'introduction d'implémentation fiables du mécanisme de rétro-propagation sur des processeurs graphiques ou GPUs.

Ce constat est illustré par les résultats obtenus par AlexNet sur le défi Imagenet [32]. Cet ensemble de données comprends 22 millions d'images réparties dans 15 milles classes environ, dans sa version 2011. Les algorithmes testés sur cet ensemble sont généralement commparé via leur erreur top-1, la classe prédite est exactement celle de l'exemple de test, et leur erreur top-5, la classe attendue est dans les 5 classes ayant le meilleur scores en sortie du CNN. Krizhevsky et al. indiquent ainsi un taux d'erreur de 37.5% pour le score dit top-1 et 17% pour le score top-5 [69].

À partir de des travaux sur LeNet et AlexNet, de nombreuses variantes de CNN ont vu

le jour. Ces nouveaux CNNs sont plus profonds comme VGGNet introduit par Simonyan et Zisserman [108]. Avec l'augmentation du nombre de couches de convolution et de max-pooling, les CNN sont capable d'extraire des caractéristiques de plus en plus complexes dans les images. Cependant, la course à la profondeur s'est accompagné de l'apparition de nouvelles difficultés pour faire converger des modèles de CNN qui comprennent toujours plus de paramètre et de couches cachées. L'augmentation du nombre de couches et de paramètres entraine notamment une augmentation du risque de sur-apprentissage décrit dans la section 1.2.4. Nous verrons dans la section 1.4 quelques techniques pour faciliter la convergence de tels modèles.

De nouvelles formes de CNN ont aussi vu le jour pour essayer de faciliter le processus d'entrainement. Ces modèles s'affranchissent de l'enchainement classique de couches de convolution et de sous échantillonage, présentés dans la figure 1.11 et introduisent chemins parallèles. Il s'agit notamment des réseaux de la famille *Inception* [112] [114] [111] et *ResNet* [48] ou *Densenet* [38]. Ces modèles utilisent notamment des couches qui additionnent ou concatènent les sorties des hautes couches pour former les entrées des couches plus profondes. A noter que ces modèles utilisent aussi la méthode dite normalisation par lot ou *batch-normalization* [56] pour optimiser leur processus d'apprentissage.

Des détails concernant ces architectures sont présentés dans l'annexe **??**.

1.3.3 L'opération de convolution

L'opération de convolution utilisée dans les CNN est définie par l'équation 1.14. Son principe de fonctionnement est aussi schématisé dans la figure 1.13. Nous pouvons interpréter la convolution comme la multiplication des pixels de l'image I par des éléments d'une fenêtre glissante matérialisée par le noyau N.

$$C(i,j) = (I * N)(i,j) = \sum_{m=0}^{x-1} \sum_{n=0}^{y-1} I(i-m, j-n) \times N(m,n). \tag{1.14}$$

Le déplacement de cette fenêtre est guidé par un pas fixé le plus souvent à 1 et matérialisé par la flèche bleue foncée dans la figure 1.13. En se déplaçant sur l'ensemble de l'image on forme donc une matrice 2D C résultant des additions-multiplications de N avec les éléments de I. Si les bords de I sont complétés avec des pixels nuls via une opération de *zero-padding* alors la dimension de C sera égale à celle de I.

L'utilisation de l'opération de convolution permet l'extraction de caractéristiques à un endroit spécifique de l'image I. Chaque sortie $C(i,j)$ correspond à la multiplication de N avec seulement un voisinage de $I(i,j)$ et non l'image entière. Il s'agit du principe de connectivité locale de l'opération de convolution. De plus, le déplacement du noyau permet de répéter cette extraction de façon similaire pour l'ensemble de l'image en conservant la référence. En effet, le noyau N reste constant pendant l'opération de convolution. On parle alors de partage de paramètres. La valeur d'un élément $C(i,j)$ sera donc directement liée à la présence d'un élément ressemblant à N au voisinage de $I(i,j)$ comme illustré dans la figure 1.14. Contrairement au MLP avec des

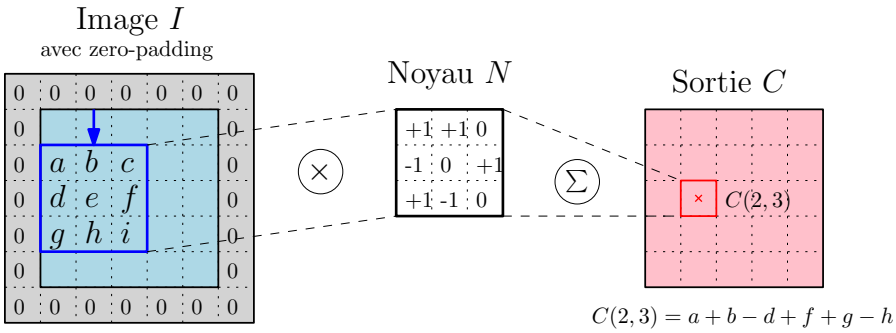


FIGURE 1.13 – Opération de convolution 2D.

couches de neurones dites denses, nous conservons dans C la relation spatiale entre les différents pixels de l'image.

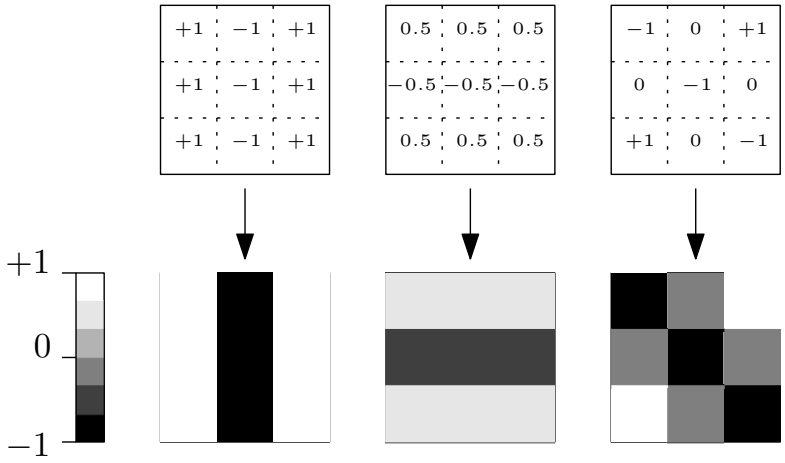


FIGURE 1.14 – Interprétation visuelle des noyaux de convolution.

Un noyau de convolution correspondant à un motif, il est possible de combiner plusieurs noyaux pour extraire différentes caractéristiques d'une image comme illustré dans la figure 1.15. Pour une image I de dimension x, y, la sortie C sera donc un tenseur de dimension $x \times y \times k$ avec k le nombre de noyaux N utilisés. Nous sommes donc en mesure d'extraire localement plusieurs motifs à différents endroits de l'image.

Comme pour les paramètres des descripteurs mentionnés dans la section 1.3.1, les coefficients de ces noyaux pourraient être définis manuellement. Cependant, pour garantir que les motifs extraits par l'opération de convolution soient les plus appropriés pour la classification d'objets, ils est préférable de les construire par le même processus d'apprentissage que pour un MLP tel que décrit dans la section 1.2.2. Avec l'ajout d'une fonction d'activation h comme pour les MLP, vus dans la section 1.1.3, nous obtenons une couche capable d'apprendre ses propre filtres N de

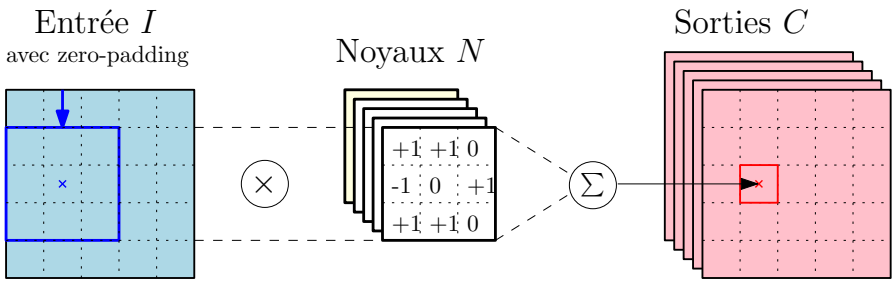


FIGURE 1.15 – Convolution avec plusieurs noyaux.

façon à minimiser J_{train}. Les éléments constitutifs de cette couche sont présentés dans la figure 1.16. Nous reviendrons sur le choix de la non linéarité h dans la section 1.4.2.

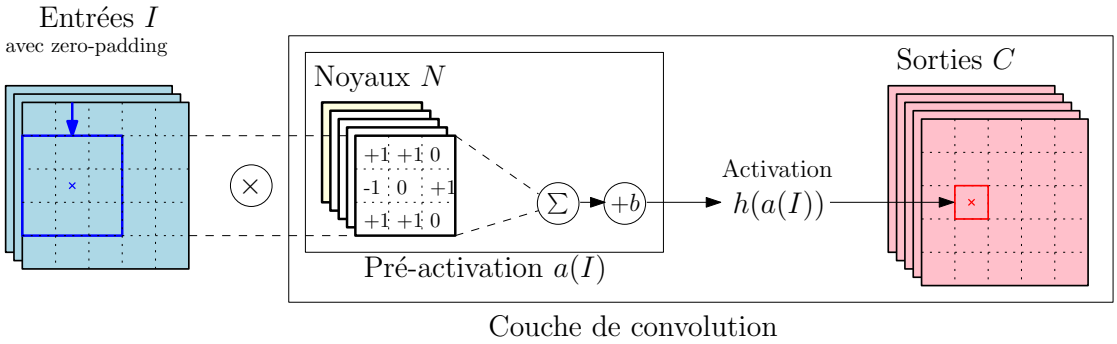


FIGURE 1.16 – Principe de fonctionnement d'une couche de convolution.

Grâce aux principes de connectivité locale et de partage de paramètres introduits dans la section 1.3.3, les couches de convolutions permettent d'extraire efficacement des caractéristiques d'une image en utilisant seulement une fraction des paramètres utilisés par une couche dense. En effet, pour une couche dense, la taille de la matrice des poids $\boldsymbol{W}$ est directement dépendante des dimensions de l'entrée. Les n neurones d'une couche dense sont connectés à l'ensemble des entrées de cette couche avec un coefficient w_i associé à chaque connexion et un terme de biais b pour chaque neurone. Si cette entrée est une image de taille $x \times y$, il faudra alors $)x \times y \times +1) \times n$ paramètres au total. Pour une couche de convolution, le nombre de paramètre n'est dépendant que du nombre et de la taille des noyaux utilisés. Généralement une couche de convolution contiendra 2^n filtres de dimension 3×3.

1.3.4 Sous-échantillonnage

Le deuxième élément constitutif des CNN sont les couches de sous-échantillonnage. Cette étape a deux effets : réduire la taille de l'entrée et augmenter l'invariance de la prédiction $\hat{\boldsymbol{y}}$ aux translations de l'entrée [42, p. 340]. Cette robustesse est utile dans le cadre de la classification

d'image car elle aide les CNN à généraliser à de nouvelles entrées. La combinaison de couches de convolutions et de couches de sous-échantillonnage permet de transiter d'information de bas niveau, comme la texture ou les formes simples, vers des information de plus haut niveau moins localisées au fur et à mesure de la progression dans le réseau.

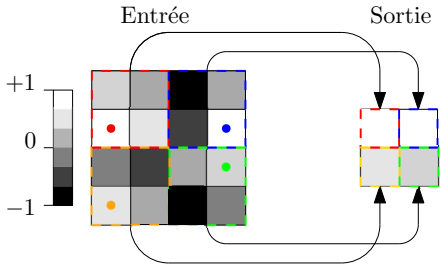


FIGURE 1.17 – Principe de fonctionnement d'une couche de *max-pooling*.

Dans les architectures de CNN, ce rôle est assuré par une opération de *max-pooling*. Une couche de *max-pooling*, illustrée dans la figure 1.17, viendra sélectionner le maximum d'une fenêtre glissante sur son entrée pour former la version sous-échantillonnée. Dans la majorité des cas, la taille de la fenêtre est de 2 unités (ou 2 pixels) de coté, ce qui revient à diviser par deux la taille de l'image en entrée.

1.3.5 Réseaux entièrement convolutifs

L'utilisation de couches de convolution a permis d'augmenter très fortement les performances des réseaux de neurones standard pour la classification ou la segmentation d'images. Dans un souci de proposer une approche simplifiée pour la construction de réseaux dédiés au traitement d'images Springerberg et al. [109] ont proposé un réseau entièrement convolutif.

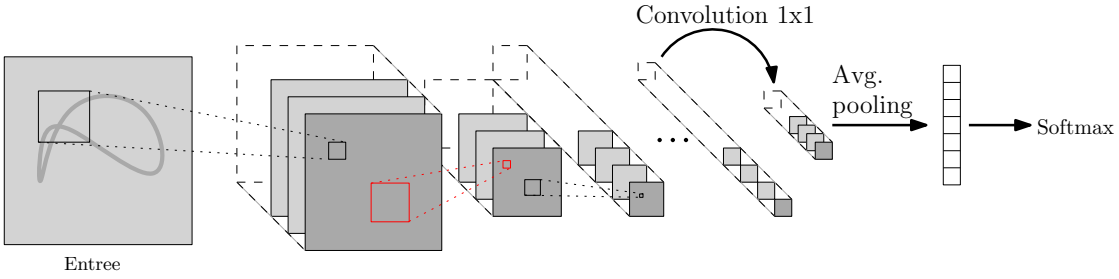


FIGURE 1.18 – Réseau entièrement convolutif.

La figure 1.18 présente ce type de réseau dans lequel les opérations de *max-pooling* sont indiquées en rouge. Springerberg et al. ont aussi proposé d'utiliser des couches de convolution avec un pas de 2 pour remplacer les couches de *max-pooling* sur l'ensemble de l'architecture. Le

choix entre une convolution à pas de 2 et les couches de *max-pooling* dépend de l'architecture et du problème.

Sur ce réseau, les couches entièrement connectées ont été supprimées et remplacées par une couche de convolution 1×1 avec un nombre de de noyaux correspondant au nombre de sorties attendues. Nous obtenons ainsi un nombre de filtres égal au nombre de sorties sur lesquels on vient calculer la moyenne des valeurs par filtre. Le vecteur de scalaires obtenu est passé dans une couche Softmax pour obtenir le vecteur de probabilité d'appartenance à une classe finale. Cette opération est appelée *Global Average Pooling* ou GAP et est illustrée dans la figure, 1.19. Au-delà de simplifier le modèle et d'alléger le nombre de paramètres en substituant les couches entièrement connectées par des opérations de convolution, le GAP améliorerait l'invariance des réseau face aux transformations de l'entrée [109].

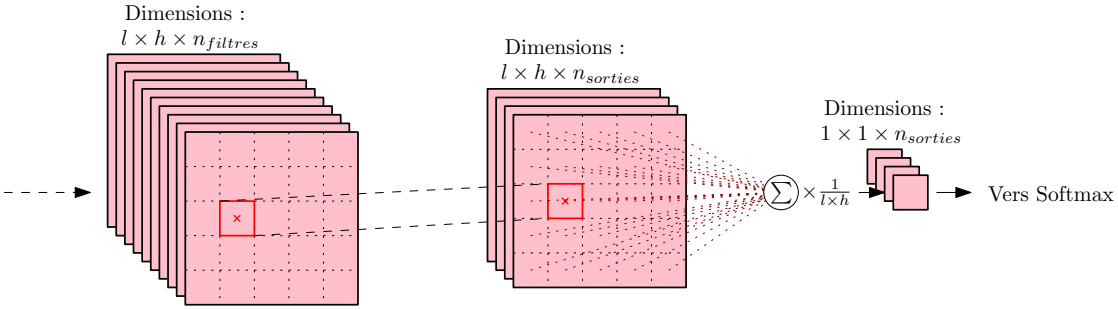


FIGURE 1.19 – Schéma de fonctionnement d'une couche de *Global Average Pooling*.

1.4 Solutions numériques pour l'apprentissage

Dans les sections 1.1 et 1.3, nous avons introduits les réseaux de neurones pour la classification ainsi que la façon de les entrainer dans la section 1.2. Cependant, nous avons aussi indiqué dans la section 1.2.4 qu'il n'y a pas de garantie pour que ce processus converge vers une solution optimale. Dans cette section nous allons présenter un ensemble de méthode permettant de faciliter ou d'optimiser l'apprentissage.

1.4.1 Régularisation

Le terme de régularisation fait référence à l'ensemble des méthodes visant à favoriser la réduction de J_{valid} sans impacter J_{train} [42, p. 228].

Early-stopping

L'arrêt anticipé ou *early-stopping* est une solution simple pour éviter le sur-apprentissage. Le principe est de suivre l'évolution de J_{valid} et d'arrêter l'apprentissage au point où J_{valid} recom-

mence à augmenter. L'arrêt s'effectue sans attendre que le modèle sur-apprenne si l'apprentissage avait continué comme illustré dans la figure 1.20.

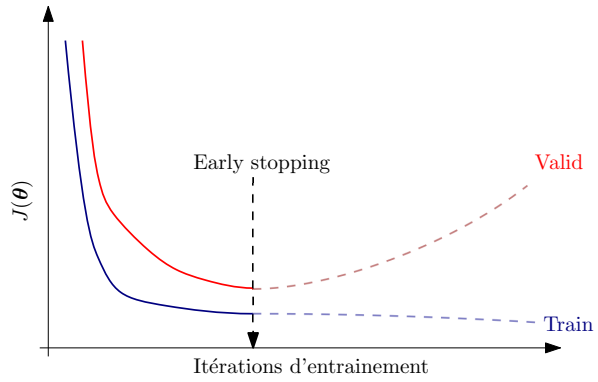


FIGURE 1.20 – Early-stopping aupoint d'inflexion de J_{valid}.

Pour arrêter l'apprentissage, nous conservons la valeur de J_{valid} après chaque itération de l'apprentissage et stoppons l'apprentissage si celle-ci augmente d'une itération à l'autre. En pratique on pourra comparer chaque J_{valid} à sa moyenne sur plusieurs itérations précédentes pour éviter d'arrêter l'apprentissage trop tôt si J_{valid} oscille pendant l'apprentissage. Cela peut notamment arriver lorsqu'on utilise des lots de petite taille pour la descente de gradient par mini-lots.

En pratiquant l'arrêt anticipé, nous libérons aussi l'utilisateur du choix de l'hyper-paramètre $n_{iterations}$ qui est donc déterminé par le mécanisme d'arrêt anticipé. Il est néanmoins utile de garder une règle d'arrêt de l'entrainement en fonction du nombre d'itération dans les cas ou le sur-apprentissage n'apparait qu'après un nombre d'itérations très élevé.

Régularisation L_1 et L_2

Une des formes de régularisation couramment utilisée pour optimiser l'apprentissage est de pénaliser les poids du réseau. Cette pénalisation permet de limiter la complexité du modèle et de réduire le risque de sur-apprentissage [14, p. 256].

Cela revient à ajouter un terme $\alpha\Omega(\boldsymbol{\theta})$ a l'équation 1.7 présentée dans la section 1.2.2 pour pénaliser les poids du vecteur $\boldsymbol{\theta}$. L'équation 1.15 est le résultat de l'ajout de ce terme à la fonction de coût. Le coefficient α permet de contrôler l'amplitude de la régularisation.

$$\arg\min_{\theta} \frac{1}{T} \sum_{t} E(f(\boldsymbol{x}^{(t)}; \boldsymbol{\theta}), y^{(t)}) + \alpha\Omega(\boldsymbol{\theta}) \tag{1.15}$$

Généralement les fonction de régularisation utilisées sont la régression Lasso [116] et la régression de Tikhonov, souvent appelées régularisation L_1 ou la norme L_2 respectivement. Le terme

de pénalisation présent dans l'équation 1.15, pour la régularisation L_1 s'écrit :

$$\alpha\Omega(\boldsymbol{\theta}) = \alpha \parallel \boldsymbol{\theta} \parallel_1 = \alpha \sum_i |\theta_i|. \tag{1.16}$$

et pour la régularisation L_2 :

$$\alpha\Omega(\boldsymbol{\theta}) = \alpha \parallel \boldsymbol{\theta} \parallel_2 = \alpha\sqrt{\boldsymbol{\theta}^\top \boldsymbol{\theta}}. \tag{1.17}$$

Même si les termes des équations 1.16 et 1.17 pénalisent tous deux $\boldsymbol{\theta}$, l'effet de ces deux formes de régularisation sur l'apprentissage est différent. L'utilisation de la norme L_2 aura pour effet de réduire les poids dont la covariance avec l'entrée est faible, favorisant ainsi les poids qui caractérisent aux mieux la variance du modèle. Cela permet de rendre la fonction de coût J_{train} la plus convexe possible et facilite la recherche du minimum global.

Utiliser la norme L_1 aura pour effet de créer un modèle parcimonieux en faisant tendre une partie des poids du modèle vers zéro pour sélectionner uniquement une partie des neurones du réseau dans la formulation de la réponse finale.

Drop-out

Le *drop-out* [110] est une autre forme de régularisation couramment utilisée, facile à utiliser et permettant de réduire le risque de sur-apprentissage. Son principe, décrit dans la figure 1.21 est de forcer à zéro la sortie de certains neurones choisi aléatoirement dans le réseau. Les neurones forcés a zéro, ou éteints, changent à chaque itération. Le *drop-out* revient donc à entrainer plusieurs sous-réseaux pendant l'apprentissage partageant les mêmes poids $\boldsymbol{\theta}$.

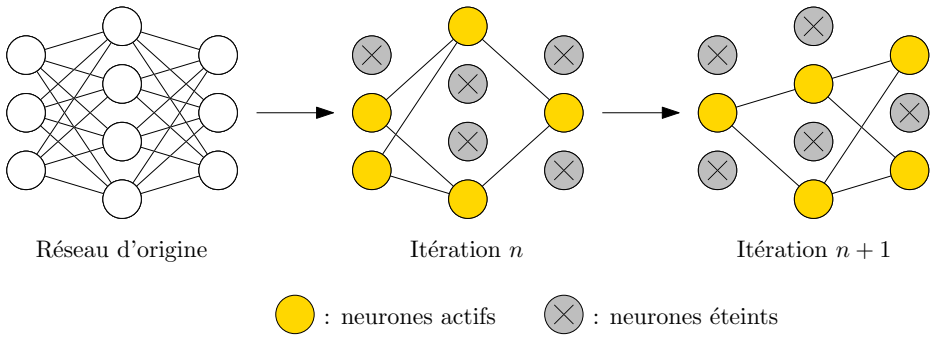


FIGURE 1.21 – Principe simplifié du *drop-out*.

La fonction d'erreur restant inchangée, le *drop-out* peut être vu comme un moyen de s'assurer que chaque neurone participe à la décision finale et que l'information qui voyage dans le modèle ne favorise pas un unique chemin. Étant donné que certains neurones sont éteints pendant l'apprentissage, il peut être nécessaire d'augmenter la taille du réseau que l'on cherche à régulariser dans les cas où on utilise du *drop-out*.

Le *drop-out* peut être vu comme une forme de *bagging* [110]. L'utiliser pendant l'entrainement reviendra a entrainer conjointement un ensemble de sous-réseau constitué des neurones non-éteints. La réponse de chacun de ces sous-réseaux sera ensuite combinée pendant le test quand l'ensemble des neurones du modèles seront "allumés". Selon Wager et al., le *drop-out* a aussi des effets proches de la régularisation L_2 [121].

En pratique, le *drop-out* reviendra à appliquer sur l'ensemble des couches cachées un masque binaire où chaque neurone se verra attibuer une probabilité d'extinction. La valeur communément utilisée pour cette probabilité, et suggérée par Strivastava et al., est de 0.5. Dans le cas des réseaux convolutifs, cette valeur est généralement 0.3.

1.4.2 Choix de la fonction d'activation

Propriétés de h

Dans les sections 1.1.2 et 1.1.3 nous avons évoqué la notion de fonction d'activation appelée h. Deux d'entre elles ont été présentées : la fonction Softmax, généralement utilisée pour la couche de sortie, et la fonction de Heavyside. Dans cette section, nous allons discuter des choix possibles pour h, et ses effets sur le processus d'apprentissage.

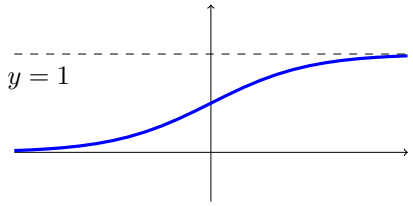
Revenons d'abord sur la fonction de Heavyside. Comme nous l'avons vu dans la section 1.2.2, la GD, la SGD et GD par lots nécéssite le calcul du gradient de la fonction d'erreur $\nabla_\theta J_{train}$. Or ce gradient dépend aussi des gradients des fonctions d'erreurs de chaque neurones $\nabla_{\boldsymbol{w}} h(\boldsymbol{w}^\top \boldsymbol{x}+b)$. Or la fonction de Heavyside est constante sur l'ensemble de son espace de définition. Son gradient est donc nul.

Pour pouvoir entrainer un réseau de neurones via une descente de gradient il faut donc disposer d'une fonction h adaptée. Celle-ci doit être :

- Non-linéaire. En cas d'utilisation d'une fonction linéaire, les sorties du réseau ne pourront être que des combinaisons linéaires de l'entrée. Il ne sera possible de classer que des entrées linéairement séparables.
- Différentiable en tout point. Cette propriété est importante pour que la descente de gradient puisse se dérouler convenablement.
- Monotone au sens large et croissante. Pour une seule couche, le choix d'une fonction d'activation monotone croissante permet de s'assurer de la convexité du modèle et facilite l'apprentissage. Dans le cas d'un MLP, la convexité n'est plus garantie. Une activation non monotone peut être choisie mais le modèle de réseau obtenu sera plus difficile à entrainer.
- Équivalente à l'identité au voisinage de l'origine, *i.e.* $h(x) \approx x$ si $x \approx 0$. Cette propriété est liée au choix de l'initialisation des poids du réseau qui est généralement proche de 0. Dans ce cas de figure la sortie du neurone sera directement proportionnelle aux entrées ce qui peut aider le réseau à converger plus rapidement. Nous discuterons plus en détails de l'initialisation des poids $\boldsymbol{\theta}$ dans la section 1.4.3

Fonction d'activation Sigmoïde **et** tanh

Deux des premières formes de fonctions h utilisées dans les réseaux de neurones sont les fonctions tangente hyperbolique tanh et Sigmoïde [7] [26]. Cette dernière est présentée dans la figure 1.22 et est similaire avec la fonction tanh. Ces deux fonctions valident l'ensemble des critères nécessaires pour une fonction d'activation. Elles présentent cependant un défaut majeur : elles saturent aux extrémités de leur ensemble de définition *i.e.* vers $+\infty$ et $-\infty$. Sur ces deux limites, la fonction Sigmoïde est tangente aux droites $y = 1$ et $y = 0$ et la fonction tanh est tangente aux droites $y = 1$ et $y = -1$.



$$h(x) = \frac{1}{1 + e^{-x}} \tag{1.18}$$

FIGURE 1.22 – Fonction Sigmoïde.

Par conséquent, pour un neurone quelconque avec une activation h de la forme Sigmoïde ou tanh, si la pré-activation tend vers des valeurs positives ou négatives importantes, le gradient qui en résulte sera faible et l'apprentissage sera fortement ralenti [129] [120].

Ces effets de saturation peuvent être partiellement compensés par l'ajout d'un terme de régularisation [41][120]. Leur existence a néanmoins poussé la communauté de l'apprentissage profond à se tourner vers d'autres fonctions d'activation.

Fonction d'activation ReLU

Parmi les alternatives, la plus populaire est la fonction ReLU ou "Rectified Linear Unit" [93] :

$$\text{ReLU}(x) = \begin{cases} 0 & si \quad x \leq 0 \\ x & si \quad x > 0 \end{cases} \tag{1.19}$$

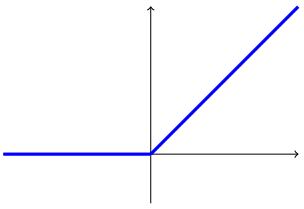


FIGURE 1.23 – Courbe de ReLU.

Cette fonction répond à l'ensemble des critères sauf en $x = 0$ ou sa dérivée n'est pas définie. En pratique on définira $\frac{\partial \text{ReLU(x=0)}}{\partial x} = 1$ pour lever l'incertitude lors de la rétro propagation du gradient pour un réseau qui utilise cette non-linéarité.

Elle présente deux avantages notables par rapport à une fonction comme tanh ou Sigmoïde : un gradient constant qui simplifie le mécanisme de rétro-propagation et l'absence de saturation pour des valeurs d'entrée x positives. Son utilisation dans des réseaux de neurones dédiés à la

classification d'images a montré une accélération importante de l'apprentissage par rapport aux fonctions tanh ou Sigmoïde [69].

L'utilisation de la fonction ReLU permet aussi de combattre la disparition du gradient dans les réseaux profonds. Ce phénomène qui touche surtout les réseaux de neurones dits récurrents [52], que nous ne présenterons pas ici, peut aussi toucher les MLP.

En effet, choisir la fonction tanh ou Sigmoïde comme activation contraint la sortie des neurones à rester dans l'intervalle $[-1, 1]$ ou $[0, 1]$ respectivement, et leurs dérivées diminuent en amplitude vers $+\infty$ et $-\infty$. Pendant la rétro-propagation, les dérivées de chaque couche sont multipliées en suivant le théorème de dérivation des fonction composées. Cela diminue peu à peu l'intensité du gradient dans les réseaux profonds, ralentissant fortement l'entrainemen [47]. La fonction ReLU étant non bornée pour des valeurs positives et son gradient constant, ce phénomène est absent.

Cependant, la partie négative de la fonction ReLU a aussi une pente nulle. La valeur en sortie de neurones ReLU sera donc toujours nulle dès que son entrée sera négative. Cela fera du réseau de neurones concerné un modèle parcimonieux dons certains neurones vont s'éteindre pendant l'apprentissage. Cela peut s'interpréter comme une forme de régularisation pour le modèle, et donc comme un possible avantage pour la variance du modèle ou son interprétabilité [124].

En pratique, ce phénomène peut arriver trop tôt dans le mécanisme d'apprentissage et entrainer ce qu'on appelle une mort prématurée des neurones. Causé par la rétro-propagation d'un fort gradient au démarrage de l'apprentissage, la modification des poids du neurone peut basculer l'entrée dans des valeurs négatives, produisant une sortie nulle pour ce neurone. Du fait de la pente nulle de la fonction ReLU pour $x < 0$ ce neurone ne contribue plus à la sortie du modèle [83].

L'utilisation d'une initialisation telle que proposée par He et al. [62] peut aussi aider à combattre cet effet de mort prématurée lors de l'apprentissage. Cette initialisation sera présentée dans la section 1.4.3.

Variantes de la fonction ReLU

Pour réduire l'impact de la mort des neurones, il est possible d'utiliser des variantes de ReLU comme la fonction *Leaky* ReLU [87] que nous noterons LReLU :

$$\text{LReLU}(x) = \begin{cases} \alpha x & si \quad x \leq 0 \\ x & si \quad x > 0 \end{cases} \tag{1.20}$$

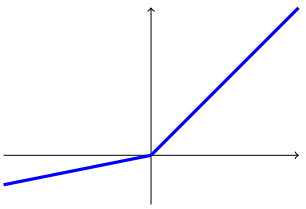


FIGURE 1.24 – Courbe de LReLU.

En ajoutant une pente α pour les valeurs négatives, le risque de mort prématurée est neu-

tralisé. De plus, cette fonction ne sature pas pour les valeurs négatives ce qui la rend robuste au risque de disparition de gradient.

L fonction LReLU garde aussi une dérivée constante sur l'ensemble de son domaine de définition en appliquant l'astuce utilisée pour ReLU en zéro. Elle bénéficie donc du gain en simplicité et vitesse de calcul du gradient de ReLU tout en étant plus performante [126]. Le coefficient α est fixé avant l'apprentissage comme tout autre hyper-paramètre. Ce coefficient peut être aussi appris en même temps que $\boldsymbol{\theta}$ lors de l'entrainement. On parlera alors de *Parametric* ReLU ou PReLU [49].

Parmi les variantes de ReLU les plus populaires nous pouvons aussi citer la *Scaled Exponential Linear Unit* ou SELU, souvent abrégée en *Exponential Linear Unit* ou ELU [24] :

$$\text{ELU}(x) = \begin{cases} \alpha(\exp(x) - 1) & si \quad x \leq 0 \\ x & si \quad x > 0 \end{cases} \tag{1.21}$$

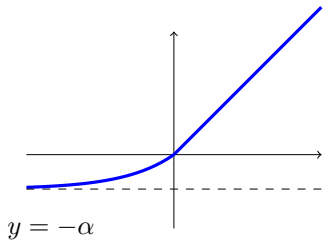


FIGURE 1.25 – Courbe de ELU.

La fonction ELU autorise les valeurs légèrement négatives mais peut toujours saturer si celles-ci sont trop importantes. Il s'agit d'un équilibre entre le problème posé par la mort d'un nombre trop important de neurones ReLU tout en autorisant un minimum de parcimonie dans les cas extrêmes.

1.4.3 Initialisation des poids

Dans la partie 1.2.2, nous avons indiqué que dans l'algorithme 1 et 3 le vecteur $\boldsymbol{\theta}$ devait être initialisé. Initialiser $\boldsymbol{\theta}$ revient à choisir un point de départ sur la surface représentée par J_{train} du réseau de façon à converger le plus rapidement vers la solution globale. Du fait de la complexité du modèle et de son nombre de paramètres, il est impossible de choisir avec certitude le point le plus adapté. Il existe cependant des règles permettant de choisir une valeur initiale pour $\boldsymbol{\theta}$ qui facilitera l'apprentissage sans avoir a sélectionner manuellement chacune des variables. Cette initialisation doit être aléatoire de façon a ce que les neurones d'une même couche aient des vecteurs de poids $\boldsymbol{w}$ différents et ainsi assurer une évolution différente pendant l'apprentissage.

Nous pourrions supposer que $\boldsymbol{\theta}$ puisse être tiré d'une simple distribution normale, cependant du fait des multiplications matricielles effectuées entre chaque couche, dans l'équation 1.2, l'amplitude des activation augmente et le gradient qui en résulte aussi entrainant ce que nous qualifions d'explosion du gradient. Ce phénomène perturbe l'apprentissage et peut être la cause d'une situation de sous-apprentissage ou sur-apprentissage décrites dans la section 1.2.4. L'explosion du gradient peut être aussi limitée en choisissant une distribution adaptée pour $\boldsymbol{\theta}$.

Une première forme d'initialisation dite de Xavier [125] peut être utilisée. Pour une couche k à m entrées et n sorties :

$$W^{[k]} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right) \tag{1.22}$$

L'initialisation de Xavier permet de conserver une variance constante pour les poids et les gradients, mêmes pour les couches profondes, d'un réseau et donc de contrôler le phénomène d'explosion des gradients. Cette initialisation est parfaitement adaptée pour l'utilisation de fonction d'activations symétrique comme la fonction tanh ou Sigmoïde.

Pour des réseaux profonds utilisant des fonctions d'activation non-symétriques, comme la fonction ReLU et ses variantes, l'initialisation Xavier ne garantit pas la convergence du modèle. A cet effet, He et al. ont introduit une nouvelle façon d'initialiser $\boldsymbol{\theta}$ [62]

$$W^{[k]} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{m}}\right) \tag{1.23}$$

Cette forme d'initialisation est donc à privilégier dès que nous souhaitons entrainer des réseaux profonds utilisant des activations ReLU.

1.4.4 Adaptation du taux d'apprentissage

Dans la section 1.2.2 nous avons présenté deux mécanismes de descente de gradient dont le taux d'apprentissage ϵ est fixé au démarrage. Il est cependant préférable d'adapter ce taux pendant l'apprentissage [42, p. 306]. En effet, un taux d'apprentissage élevé est utile au démarrage car il permet d'explorer l'espace des solutions de E plus rapidement pour faciliter la convergence [12]. Néanmoins un taux élevé pourra aussi faire converger la solution vers un minimum local ou osciller la valeur de E et pénaliser l'apprentissage. De même, un ϵ faible est aussi intéressant car il peut agir comme une forme de régularisation en faisant osciller J [12]. Cependant, un taux faible au démarrage de l'apprentissage peut augmenter considérablement la durée de l'apprentissage.

Une solution simple pour profiter de ces deux effets est d'adapter le taux d'apprentissage en fonction de la progression de l'apprentissage en faisant décroitre ϵ en fonction du nombre d'époques complétées par la GD ou la GD par mini-lots. Cela peut être fait en soustrayant régulièrement au taux d'apprentissage une valeur fixe après un nombre d'itération de GD définie à l'apprentissage. Nous pouvons aussi appliquer une loi de décroissance basée sur une fonction précise dont les entrées sont la valeur de ϵ au démarrage de l'apprentissage, le nombre d'époques et dont la sortie sera un nouveau taux d'apprentissage ϵ'.

Faire décroitre ϵ seulement en fonction du nombre d'itérations d'apprentissage est limité. Cela ne permet pas de s'adapter à la présence de point cols, minimum locaux ou plateaux lors de l'optimisation de J_{train}. Pour remédier à ce problème il est possible d'utiliser des variantes de la GD ou GD par mini-lots qui adaptent automatiquement ϵ en fonction des évolutions de J.

Notons par exemple RMSprop [117], Adagrad [34] et Adadelta [134]. Nous nous intéresserons

ici uniquement à Adam [67], étant donné qu'il s'agit de l'algorithme que nous avons utilisé pour entrainer les modèles présentés dans le chapitre 3. Pour plus d'information sur RMSprop, Adagrad, Adadelta ainsi que des variantes de ces algorithmes se référer à leurs articles respectifs ou le chapitre 8.5 et 8.6 du *Deep Learning Book* [42].

Algorithm 4 Adam [67]

Require: X_{train}, Y_{train}
Require: $n_{iterations}$ ▷ Détermine le nombre d'itération de la GD
Require: ϵ ▷ Détermine la vitesse de la GD
Require: ρ_1 et ρ_2 ▷ Facteurs de décomposition
Require: δ ▷ Constante de stabilité numérique
  Initialiser $\boldsymbol{\theta}$
  $\boldsymbol{m_1} = 0$, $\boldsymbol{m_2} = 0$
  $s = 1$, $\delta = 1e^{-6}$
  for $n_{iterations}$ **do**
    for $n_{etapes} \leftarrow 1, \lfloor n_{train}/m \rfloor$ **do**

      Sélection aléatoire d'un lot de couples $\{x^{(t)}, y^{(t)}\}$ depuis X_{train}
      $\nabla J_{lot} \leftarrow \frac{1}{m} \sum_{t=1}^{m} \nabla_\theta E_{train}(\hat{y}, y^{(t)})$ ▷ Calcul du gradient du lot
      $\boldsymbol{m_1} \leftarrow \frac{\rho_1 \boldsymbol{m_1} + (1-\rho_1)\nabla_{lot}}{1-\rho_1^s}$ ▷ Calcul du premier moment
      $\boldsymbol{m_2} \leftarrow \frac{\rho_2 \boldsymbol{m_2} + (1-\rho_2)\nabla_{lot} \odot \nabla_{lot}}{1-\rho_2^s}$ ▷ Calcul du second moment
      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \frac{\boldsymbol{m_1}}{\sqrt{\boldsymbol{m_2}} + \delta}$ ▷ Mise à jour des poids
      $s \leftarrow s + 1$ ▷ Mise à jour de l'index de l'itération
    end for
  end for

Adam est décrit dans l'algorithme 4. Contrairement aux algorithmes de GD et GD par mini-lots, la mise à jour de $\boldsymbol{\theta}$ se fait par le biais de deux moments $\boldsymbol{m_1}$ et $\boldsymbol{m_2}$ calculés à partir du gradient de l'erreur sur un lot ∇_{lot}. De façon simplifiée, l'ajout de ces deux moments, pondérés par l'évolution de s, permet d'accélérer l'apprentissage dans les directions qui favorisent la diminution de J_{train}.

Aussi, la mise à jour de $\boldsymbol{\theta}$ ne dépend pas seulement du choix de ϵ mais de l'évolution de ∇_{lot} et des paramètres ρ_1 et ρ_2 qui viennent moduler le taux d'apprentissage en fonction de l'évolution de J_{train} pendant les itérations qui précèdent.

Bien que choisir Adam comme algorithme d'optimisation implique d'ajouter deux hyper-paramètres supplémentaires, il s'agit d'un algorithme robuste vis-à-vis du choix de ces hyper-paramètres. Cela en fait un algorithme populaire pour l'optimisation des réseaux de neurones modernes.

1.4.5 Préparation des données d'apprentissage

Normalisation

La préparation des données d'entrainement aide aussi à optimiser l'apprentissage [42, p. 455]. Parmi les pratiques couramment utilisées, il y a le centrage de la moyenne. Comme son nom l'indique, il s'agit de soustraire à chaque entrée $\boldsymbol{x}^{(t)}$ de X_{train} sa moyenne.

Cette étape de centrage est souvent combinée à une normalisation des entrées $\boldsymbol{x}^{(t)}$ dans l'intervalle $[0, 1]$ via l'opération suivante :

$$x_i^{(t)} = \frac{x_i^{(t)} - \min(\boldsymbol{x^{(t)}})}{\max(\boldsymbol{x^{(t)}}) - \min(\boldsymbol{x^{(t)}})} \tag{1.24}$$

La combinaison du centrage des données et de la normalisation permet de corriger certains biais pouvant exister dans le jeu de données.

Augmentation de données

Une autre pratique couramment employée pour la préparation de X_{train} est l'augmentation de données [42, p. 240] [14, p. 265]. Il s'agit simplement de compléter X_{train} en lui ajoutant des variantes transformées de lui-même.

Cette pratique est surtout utilisée dans le cas de la classification d'images avec les CNN décrits dans la section 1.3. En effet, il est facile d'ajouter des nouveaux exemples d'entrainement à partir d'images transformées de X_{train} par des rotations, des translations ou l'extraction de zones d'une image. Ces transformations sont applicables tant qu'elles ne dénaturent pas l'entrée ou qu'elles risquent d'introduire un biais. Ainsi, pour un problème de classification d'images où l'orientation de l'image est importante, on évitera d'utiliser certaines rotations pour l'augmentation de données.

L'augmentation de données est surtout utilisée au même titre que la régularisation. Augmenter X_{train} ne permettra pas de compléter un manque de données d'entrainement, mais cela aidera un modèle à généraliser et à prévenir l'apparition de sur-apprentissage tel que décrit dans la section 1.2.4.

1.5 Difficultés pour l'entrainement des CNN

Nous avons vu dans la section 1.4 qu'il était possible de mettre en place un ensemble de techniques pour faciliter l'apprentissage des réseaux de neurones, y compris les CNN.

Il existe cependant encore des points sur lesquels les CNN présentent des faiblesses ou des vulnérabilités. Nous avons voulu regrouper dans cette section celles qui ont motivées les contributions qui seront présentées dans la suite de ce manuscrit.

1.5.1 Le besoin de données

Malgré l'amélioration constante du processus d'entrainement des réseaux de neurones, les performances de ces modèles sont toujours conditionnées à la qualité et la quantité des données disponibles pendant l'entrainement. Nous avons vu dans le chapitre 1.4.5 qu'il était possible d'augmenter artificiellement cette quantité de données en les transformant. Cependant cette pratique ne pourra pas compenser l'absence d'instances d'une classe dans une base de données.

Pour un problème de classification, le nombre d'images minimum requis est difficile à évaluer. Les recommandations varient beaucoup en se basant soit sur le nombre de paramètre du réseau [5], soit sur le nombre d'instances à classer [22] [27]. La taille minimale du jeu de données nécessaire pour obtenir de bonnes performances de classification reste donc le plus souvent à déterminer de façon empirique, en fonction de la complexité du problème et du type de modèle de CNN utilisé.

Cependant, pour des problèmes où la quantité de données ne permet pas d'entrainer correctement un réseau, il est possible d'utiliser un modèle déjà entrainé sur d'autres données et de le ré-entrainer pour une nouvelle tâche avec une quantité de données limitées. Cette technique, qualifiée de *fine-tuning* ou affinage, se base sur la capacité des réseaux convolutifs à apprendre des caractéristiques transférables à de nouveaux domaines [132].

De nombreux modèles de réseaux de neurones, pré-entrainés sur des jeux de données accessibles à tous comme Imagenet [32] qui comprend 14 millions d'images, sont mis à disposions de tous par la communauté scientifique. Combiné à l'affinage, nous sommes en mesure de rapidement entrainer un modèle que nous ne pouvions pas entrainer correctement à cause de la faible quantité de données disponibles.

Cette approche a permis de démocratiser l'apprentissage profond à l'aide de CNN dans des domaines où la quantité de données annotées n'était pas suffisante pour entrainer de zéro un CNN comme la reconaissance des émotions [94], l'imagerie médicale [92] [133] [40] ou l'ananlyse d'images radar [57].

1.5.2 Principe du transfert d'apprentissage à l'aide de l'affinage

Le transfert d'apprentissage repose sur deux réseaux de neurones : un réseau maître et un réseau élève. Le principe, schématisé dans la figure 1.26, est de transférer une partie des caractéristiques du réseau maître, apprise sur un domaine A, vers le réseau élève qui doit être employé sur un domaine B différent de A. Cela suppose de disposer d'une base de donnée annotée pour les deux domaines A et B. Cependant, il n'est pas nécessaire de disposer d'un grand nombre d'images pour le domaine B. Seul le jeu de donnée utilisé pour le domaine A doit être de taille suffisante pour faire converger le réseau de neurones utilisé.

La première étape consiste donc à entrainer le réseau maître sur le domaine A, jusqu'à atteindre des performances en classification satisfaisantes. Dans un deuxième temps, on construit le réseau élève en réutilisant une partie des couches du réseau maître, et les poids associés, que

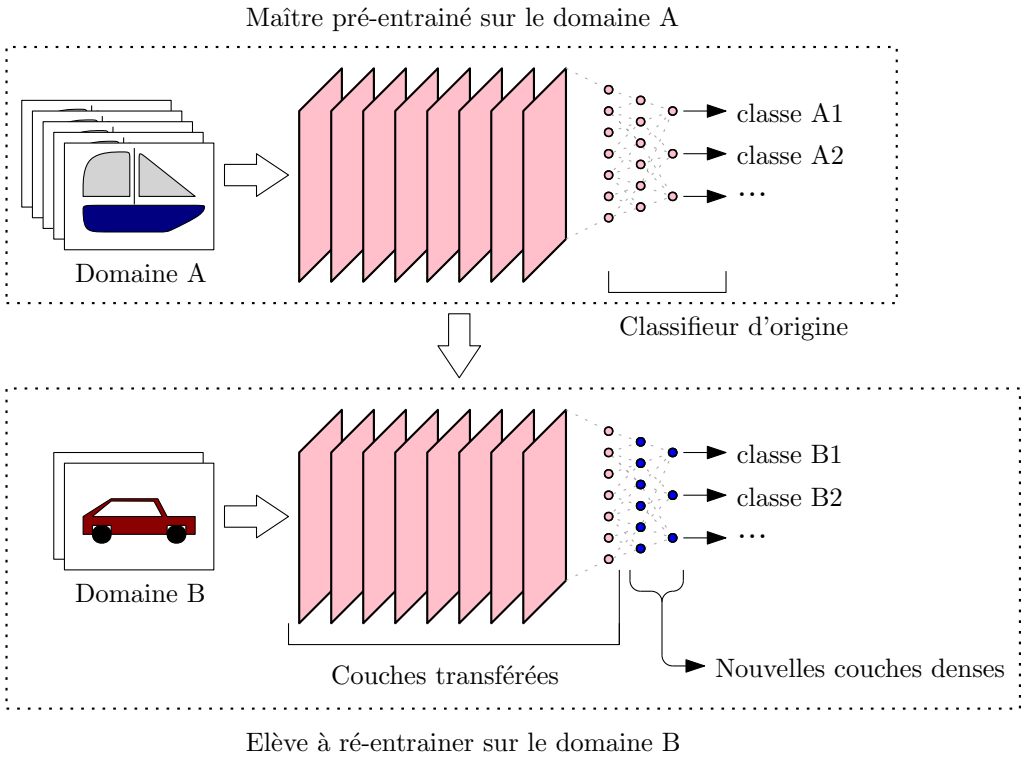


FIGURE 1.26 – Principe général du transfert de domaine.

l'on complète avec le nombre de couches et de sorties nécessaires pour le domaine B.

Pour le cas de la classification d'images, les deux modèles sont généralement des CNN tels que décrits dans la section 1.3. Généralement, lors du transfert, les couches de convolution du réseau maître sont transférées vers le réseau élève, complétées par seulement quelques couches entièrement connectées pour le domaine B.

Enfin, le réseau élève est entrainé par rétro propagation sur les images du domaine B jusqu'à converger vers une solution satisfaisante. Pour cet entrainement, il est possible de figer les poids des couches transférées et d'entrainer uniquement les couches ajoutées pour le domaine B, en bleu dans la figure 1.26. Il s'agit de l'approche la plus rapide car seulement les couches denses sont à ré-entrainer.

Dans certains cas, il est aussi préférable de ré-entrainer une partie des couches transférées conjointement aux couches ajoutées sur le réseau élève, comme illustré dans la figure 1.27. Les couches 1 à K sont figées et le reste des couches à partir de $K+1$ seront ré-entrainées en même temps que les couches ajoutées. Le nombre de couches transférées à inclure dans la bouche de rétro-propagation dépend de l'écart entre le domaine A et le domaine B. Plus cet écart augmente plus il est conseillé d'augmenter le nombre de couches à ré-entrainer. Néanmoins, l'augmentation du nombre de couches à ré-entrainer nécessitera de disposer d'un nombre de

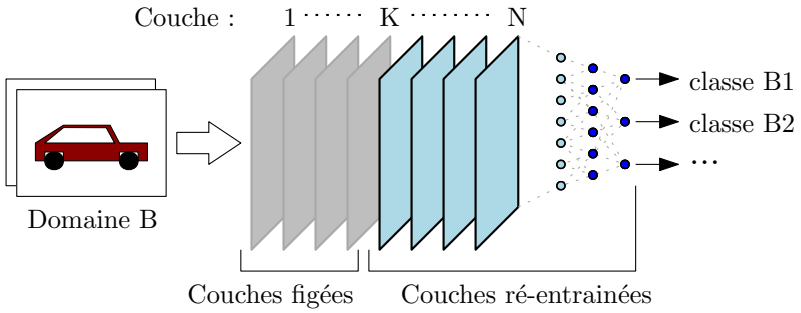


FIGURE 1.27 – Transfert de domaine avec ré-entrainement de plusieurs couches.

données plus conséquent pour le domaine B. Dans les situations les plus extrêmes la totalité des couches transférées pourront être ré-entrainées. On parle alors de *full-model fine tuning*.

Lors de la phase d'entrainement sur le domaine B, on choisira généralement un taux d'apprentissage faible pour conserver les caractéristiques apprises par les couches transférées.

1.5.3 Vulnérabilité des CNN

Les réseaux de neurones, et plus particulièrement les CNN, sont devenus incontournables pour les problèmes de classification. Cependant malgré leurs bonnes performances, il est possible de créer des entrées particulières qui peuvent modifier fortement la réponse du réseau. Les méthodes qui permettent de générer ces exemples sont regroupées sous le terme d'attaque adverse.

Le fonctionnement d'une attaque adverse est présenté dans la figure 1.28.

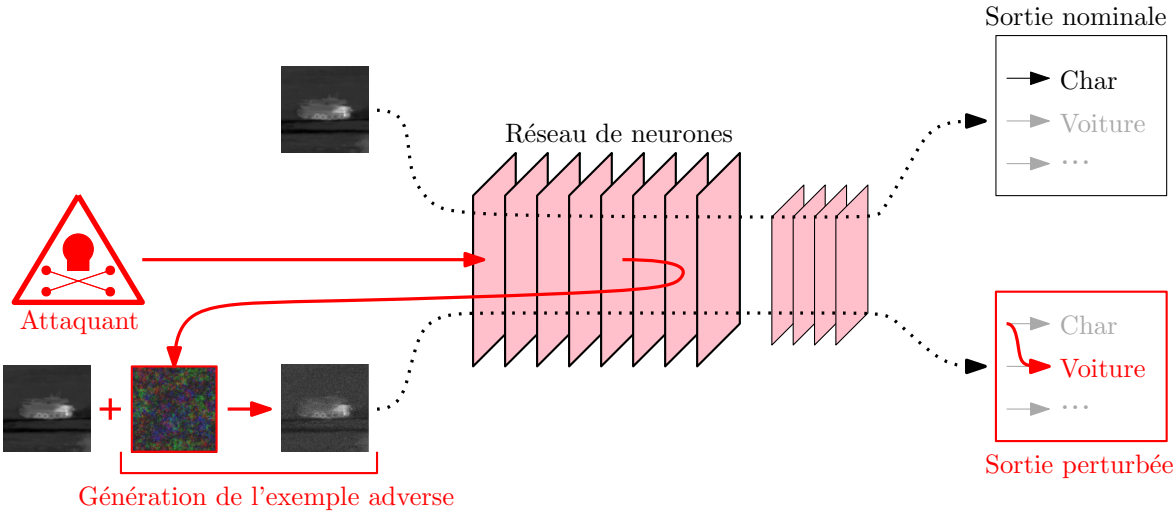


FIGURE 1.28 – Principe des attaques adverses appliquées aux CNNs.

Une attaque adverse se manifestera sous la forme d'une perturbation, souvent invisible à l'oeil nu, qui peut ressembler visuellement à un bruit aléatoire comme illustré dans la figure 1.28. Cette perturbation est ajouté aux entrées du réseau et vient perturber sa prédiction. Elle est créée à partir des informations dont l'attaquant dispose sur le réseau comme sa structure, ses paramètres ou ses sorties.

On qualifiera l'attaque de *White-box* si l'attaquant dispose de connaissances précises sur le modèle, notamment ses paramètres et les données d'entrainement. Les attaques *White-box* peuvent donc se baser sur la rétro-propagation, comme pour la *Fast Gradient Sign Method* [43], ou sur un processus d'optimisation [113] [20].

À l'inverse, l'attaque sera qualifiée de *Black-box* si l'attaquant n'a pas accès à la structure du modèle, ses poids $\boldsymbol{\theta}$ ou ses gradients. Dans cette situation un attaquant n'aura accès qu'au flux d'entrée et aux sorties du modèle comme sources d'informations. Dans cette famille d'attaques nous pouvons y trouver des attaques utilisant des modèles de substitution [99] ou l'estimation des gradients du modèle attaqué [21]. Sous certaines conditions, il est aussi possible de générer des attaques transférables [81] ou universelles [104], ce qui augmente encore le risque de rencontrer des exemples adverses pour des systèmes sensibles.

Au-delà d'avoir pointé du doigt l'existence de vulnérabilités, l'étude des attaques peut apporter des informations sur la propagation de l'information dans les réseaux de neurones. Ainsi, les exemples adverses pourrait s'interpréter comme une manifestation de la tendance des réseaux de neurones à baser leur réponse sur des caractéristiques non-robustes [55]. Leur étude pourrait apporter des précisions sur les représentations des données d'entrées X par le réseau, ainsi que des indices pour améliorer leur robustesse.

1.6 Conclusion du chapitre

Dans ce chapitre, nous avons présenté un aperçu des concepts théoriques relatifs aux réseaux de neurones pour la classification et aux CNN. Nous avons commencé par présenter l'algorithme du Perceptron et le Perceptron multi-couches, ou MLP, qui représentent l'origine des réseaux de neurones modernes. D'après le théorème d'approximation universelle les MLP sont capables en théorie d'approximer n'importe quelle fonction continue sous certaines condition. Pour modéliser une fonction en particulier, il faut cependant déterminer les bonne valeurs des paramètres $\boldsymbol{\theta}$ du MLP.

Nous avons vu ensuite que $\boldsymbol{\theta}$ peut être appris. Cet apprentissage utilise une fonction de substitution, ou fonction de coût J_{train}, qui est minimisée par apprentissage en utilisant des données d'entrainement. Minimiser J_{train} se fera à l'aide d'une descente de gradient. Nous espérons ainsi faire converger J_{train} vers un minimum global qui correspond à la fonction que nous cherchons à modéliser. Atteindre ce minimum global n'est cependant pas garanti. En fonction du nombre de paramètres ou de la complexité du problème, l'apprentissage peut ne pas converger. Nous pouvons alors nous retrouver dans une situation de sur-apprentissage ou de sous-apprentissage.

Après avoir présenté les réseaux de neurones et le mécanisme d'apprentissage, nous nous sommes intéressé aux réseaux de neurones dédiés à la classification d'image, les CNN. Ces réseaux de neurones combinent un MLP avec en entrée un enchainement de couches de convolution et de sous-échantillonnage plus adaptés au traitement des images que les couches de neurones des MLP. Nous avons aussi introduit les réseaux de neurones entièrement convolutifs, qui se composent uniquement de couches de convolution et de sous-échantillonnage.

Puis, nous avons présenté plusieurs méthodes permettant d'optimiser l'apprentissage pour réduire le risque de sur-apprentissage. Nous avons notamment introduit les notions de *Drop-out* et l'algorithme Adam qui seront utilisés dans la suite du manuscrit.

Enfin, nous avons terminé ce chapitre en présentant quelques difficulté supplémentaires pour l'entrainement des CNN ainsi que certaines de leurs limites. Nous avons notamment présenté la notion de *fine-tuning*, qui permet de faciliter l'entrainement des CNN avec peu de données, et le risque présenté par les exemples adverses. Cette présentation, bien que succincte, facilitera la compréhension des choix présentés dans les chapitre 3 et 4.

Chapitre 2

Imagerie infrarouge pour la classification de véhicules militaires

Contents

Après avoir présenté dans le chapitre 1 les réseaux de neurones et plus particulièrement les réseaux de neurones convolutifs, nous allons ici nous concentrer sur le principe de l'imagerie infrarouge et son utilisation dans le domaine de la défense.

La section 2.1 se concentrera sur le principe de l'imagerie infrarouge et son utilisation pour des applications civiles. Il s'agira d'une présentation succincte qui ne couvrira qu'une partie des effets physiques et des interactions à l'origine du rayonnement infrarouge. La section 2.2 sera consacrée à l'utilisation de l'infrarouge dans un contexte militaire. Puis, dans la section 2.3, nous présentons les jeux de données d'images infrarouges qui seront utilisés dans les chapitres 3 et 4. Enfin, nous présentons dans la section 2.4 les principaux enjeux de la classification de cibles en infrarouge.

2.1 Vision par ordinateur en imagerie Infrarouge

2.1.1 Principe de l'imagerie infrarouge

Rayonnement IR

Le rayonnement infrarouge, ou IR, correspond l'ensemble des rayonnements électromagnétiques dont la longueur d'onde λ est comprise entre 780 nm et 1 mm. Dans cet intervalle, tout objet dont la température est supérieure à celle du zéro absolu émet un rayonnement. L'intensité et le spectre d'émission de ce rayonnement est directement lié à la température de l'objet. L'intervalle des valeurs de λ qui correspondent au rayonnement infrarouge dans son ensemble, est divisé en plusieurs bandes qui sont détaillés dans la figure 2.1.

Ce rayonnement électromagnétique particulier peut être calculé à partir de la loi de Planck qui définit la densité spectrale du flux de rayonnement, équivalent à une puissance émise par unité surface et par longueur d'onde, pour ce qu'on appelle un corps noir. Un corps noir est un élément théorique qui absorbe la totalité de l'énergie des rayonnements qu'il a reçu, quelle que soit la longueur d'onde. Cette énergie est ensuite ré-émise sous la forme d'un rayonnement électromagnétique dont la luminance dépend de l'élévation en température lié à la quantité d'énergie absorbée. L'expression de la loi de Planck est détaillée dans l'équation 2.1.

$$L^{\circ}_{\Omega,\lambda}(\lambda, T) = \frac{2hc^2}{\lambda^5} \times \frac{1}{\exp^{\frac{hc}{\lambda k_B T}} - 1}. \tag{2.1}$$

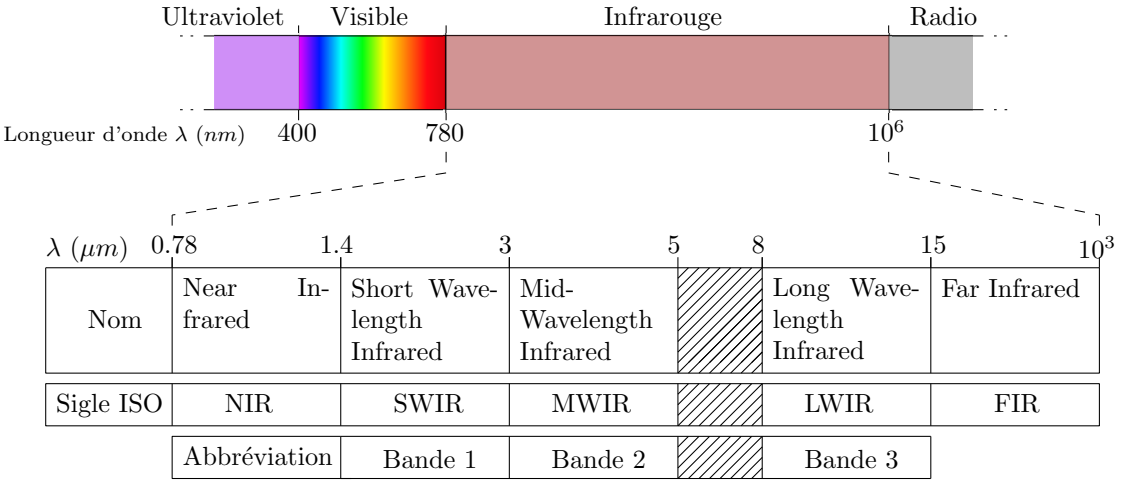


FIGURE 2.1 – Composantes du rayonnement infrarouge.

Pour une longueur d'onde λ et un angle solide Ω, dépend de la température T du corps, en Kelvin, et de trois constantes : la constante de Planck $h = 6.62607015 \times 10^{-34} J.s$, la constante de Boltzmann $k_B = 1.380649 \times 10^{-23} J.K^{-1}$ et la vitesse de la lumière dans le vide $c = 2.99792458 \times 10^{8} m.s^{-1}$. À partir de l'expression de $L^{\circ}_{\Omega,\lambda}$, il est possible de calculer la luminance spectrale L_λ d'un corps noir en l'intégrant sur l'angle solide Ω. La luminance L° d'un corps noir, équivalent à une puissance rayonnée, sera obtenue en intégrant L°_{λ} sur une fenêtre du spectre électromagnétique. La luminance d'un matériau réel dépendra de son émissivité e et de la loi de Planck pour un corps noir. L'émissivité dépend elle aussi de la longueur d'onde du rayonnement. Nous pouvons alors définir une expression de la la densité spectrale du flux de rayonnement pour un corps réel avec l'équation 2.2.

$$L_{\Omega,\lambda}(\lambda, T) = e(\lambda) L^{\circ}_{\Omega,\lambda}(\lambda, T). \tag{2.2}$$

En intégrant l'expression de l'équation 2.2 sur Ω et λ, il est possible d'obtenir la luminance spectrale L_λ et la luminance L d'un corps réel. L'émissivité d'un matériau dépendra de ses propriétés physiques et déterminera la quantité d'énergie rayonnée pour une longueur d'onde donnée. Pour tout matériau, $e \in]0, 1]$, une émissivité de 1 correspondra au corps noir.

Avec un dispositif d'acquisition adapté, il est possible de mesurer le rayonnement de l'équation 2.2 d'un objet. Ce procédé est appelé thermographie infrarouge. Cependant, le dispositif ne mesurera que le rayonnement apparent de l'objet et non son rayonnement direct comme présenté dans la figure 2.2. Le rayonnement direct ne pourrait être mesuré que dans le vide et en l'absence de toute sources de rayonnement ou d'interactions avec l'environnement. De plus, comme pour l'ensemble du spectre électromagnétique, certains matériaux peuvent réfléchir ou absorber tout ou partie de ces rayonnements.

Le rayonnement apparent intègre donc l'ensemble des contributions externes notamment le

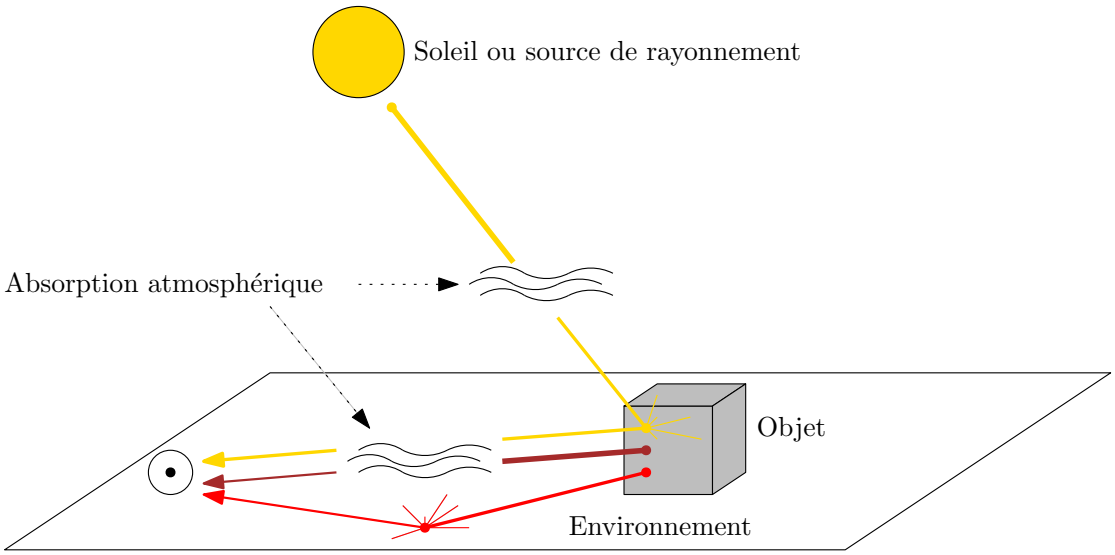


FIGURE 2.2 – Composantes du rayonnement apparent.

rayonnement des autres sources présentes dans la scène et les réflexions ou diffusions de ces rayonnements dans l'environnement. Ainsi, deux objets à la même température mais avec des émissivités différentes apparaitrons comme ayant des températures différentes. De même, deux objets ayant des émissivités et des températures différentes peuvent avoir la même température apparente. Comme pour la lumière visible, ce rayonnement peut être mesuré à l'aide d'une caméra spécifique appelée simplement caméra thermique ou caméra infrarouge.

Dispositifs d'acquisition (Caméras IR)

Le fonctionnement d'une caméra thermique est proche de celui d'une caméra fonctionnant dans le visible. La structure simplifiée du dispositif est décrit dans la figure 2.3.

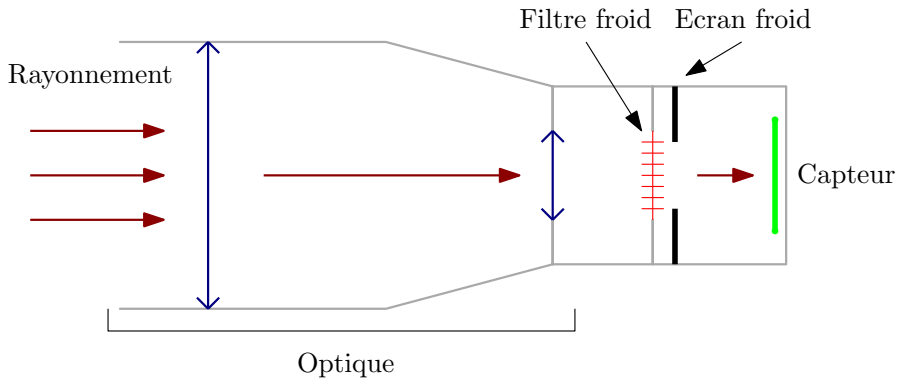
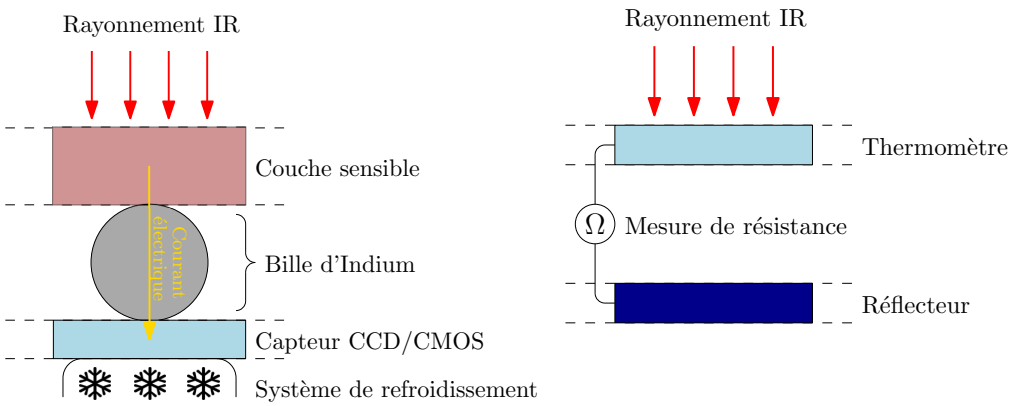


FIGURE 2.3 – Description simplifiée d'une caméra thermique.

Le premier élément de la caméra est l'optique. Il sert principalement à former l'image de la scène sur le capteur. La ou les lentilles qui le constituent sont faites dans un matériau qui ne laissent passer que la portion utile du spectre infrarouge. Comme pour une caméra fonctionnant dans le visible, la caméra thermique dispose aussi d'un filtre froid pour limiter le spectre des rayonnements détectés à une bande spécifique en fonction de l'utilisation prévue. Ce filtre est suivi d'un écran appelé écran froid qui agit comme un diaphragme pour limiter la quantité de rayonnements qui atteindront le capteur. Ce capteur, placé dans le plan focal de la caméra,viendra mesurer l'intensité du rayonnement reçu. Nous distinguons deux grandes familles de capteurs, illustrés dans la figure 2.4 :

- Capteurs refroidis, ou IR-R, présenté dans la figure 2.4a : Ils sont constitués de matrices de petits éléments, ou pixels, faits de trois couches. La première couche est une couche sensible aux photons du rayonnement thermique qui convertit les photons reçus en électrons. La dernière couche est un transistor de type CCD, pour *Charge-Coupled Device*, ou CMOS pour *Complementary Metal Oxide Semi-conductor*, qui convertira un flux d'électrons en un signal électrique qui correspondra à la réponse du pixel. La deuxième couche sera un pont conducteur qui relie la couche sensible et le capteur. Elle est généralement constituée de billes d'indium. Du fait de la sensibilité de la matrice du pixel, le capteur est placé sous vide pour éviter toute perturbation lié à la convection et il est refroidi pour éviter que l'agitation thermique ambiante masque le signal reçu. Le principal défaut des capteurs refroidis vient de l'encombrement et la consommation supplémentaire liée au système de refroidissement.
- Capteurs non-refroidis, ou IR-NR, présenté dans la figure 2.4b : Ils sont constitués de pixels faits de structures de silicium monolithique appelés micros-bolomètres dont la résistance électrique varie en fonction du rayonnement thermique reçu. Ces capteurs sont moins sensibles à l'agitation thermique ambiante et ont seulement besoins d'être placés sous vide pour limiter les perturbations liées à la convection. Les capteurs non-refroidis sont généralement moins encombrants et moins coûteux à produire que les versions avec refroidissement. Cependant, ils sont limités en résolution et la qualité de l'image qu'ils produisent peut se dégrader au fur et à mesure de l'utilisation du fait de l'échauffement progressif du capteur.

En général, les capteurs IR-R sont utilisés pour des applications en bande 2, ou MWIR, et les capteurs IR-NR sont utilisés pour les applications en bande 3, ou LWIR. Ces pixels refroidis ou non-refroidis sont ensuite couplés à un circuit de lecture qui formera l'image brute en sortie du capteur. Cette image est souvent perturbée par un bruit causé par l'environnement et les différents éléments de la caméra comme l'optique, le filtre ou le capteur. Une série de traitements est alors appliqué sur l'image pour corriger certaines de ces perturbations, notamment celles causées par les éléments de la caméra. Après ces traitements, nous obtenons une image en niveaux de gris, généralement codée sur un minimum de 14 bits. Des exemples d'images infrarouges sont présentés dans la figure 2.5.



(a) Capteur refroidi (b) Capteur non-refroidi

FIGURE 2.4 – Schémas simplifiés des deux familles de capteurs infrarouges.

Au niveau de chaque pixel, la luminance reçue est convertie par la caméra qui intègre cette luminance sur le spectre du rayonnement IR auquel il est sensible. La valeur finale en niveau de gris dépendra des paramètres de la caméra, notamment de son gain. La grandeur codée dans l'image finale ne correspond donc pas à une grandeur physique directement. Elle représente une multitude de contributions en plus du rayonnement apparent de la scène. La caméra IR devra donc être calibrée en laboratoire pour déterminer la relation entre la valeur numérique de la scène et le rayonnement reçu si nous souhaitons l'utiliser pour mesurer un rayonnement apparent. Cette calibration est réalisée à l'aide d'un dispositif dont le rayonnement se rapproche de celui d'un corps noir pour pouvoir faire le lien entre la valeur numérique des pixels et une température apparente.

(a) Exemple du jeu de données SENSIAC (b) Exemple du jeu de données FLIR ADAS

FIGURE 2.5 – Exemples d'images infrarouges obtenues à l'aide de caméras thermiques. Ces images sont disponibles dans les jeux de données SENSIAC [90] et FLIR ADAS [36]

2.1.2 Applications civiles de la thermographie

Du fait de leur capacité à visualiser une température apparente et surtout une différence ou un gradient de température dans un environnement à distance, les caméras thermiques sont utilisées dans de nombreux domaines.

Elles sont particulièrement utiles dans l'industrie du bâtiment où elles permettent rapidement d'analyser la qualité de l'isolation thermique d'une structure, ou détecter des zones d'humidité dans les murs. Ces caméras peuvent aussi être utilisées pour inspecter ou contrôler la qualité de pièces ou de soudures dans l'industrie. Certains défauts sont en effet plus faciles à détecter dans l'infrarouge que dans le visible.

De la même façon, le monde médical ou vétérinaire utilise aussi la thermographie comme outil de diagnostic. Les caméras thermiques permettent notamment de repérer dans les foules les personnes présentant une température corporelle supérieure à la moyenne qui peut être causée par poussée de fièvre. Cela permet de filtrer et contrôler rapidement les cas suspects dans le cas de pandémies comme le SARS ou plus récemment le Covid 19.

Les caméras infrarouges sont aussi fréquemment utilisées comme systèmes d'aide à la conduite ou comme capteurs pour les véhicules autonomes. Elles permettent en effet de mieux percevoir l'environnement de nuit ou dans des conditions météo dégradées où les caméras optiques ne sont pas suffisantes. Cette capacité en fait des solutions de choix pour les systèmes de surveillance civils ou militaires, notamment pour la surveillance de nuit.

Comme pour les problèmes de vision par ordinateur utilisant des images visibles, l'apprentissage profonds et les CNN sont devenu l'algorithme de prédilection pour traiter des problèmes de classification ou de segmentation à partir d'images infrarouges.

2.2 Applications militaires de la classification en infrarouge

2.2.1 Domaine d'utilisation et contexte d'emploi

Développement de l'imagerie infrarouge militaire

L'imagerie infrarouge est souvent associée au monde de la défense dans l'imagination collective, non sans raison. Son utilisation militaire s'est développée pendant la guerre du Vietnam ou les premiers systèmes dits FLIR pour *Forward Looking Infrared* sont venu compléter les radars et les capteurs optiques des avions de combat des États-Unis. L'identification active était faite à l'aide de leur radar embarqué. L'identification passive et à longue distance était effectuée à l'aide de caméras fonctionnant dans le domaine visible combinées à des optiques à fort grossissement. Avec l'introduction de l'imagerie infrarouge, ces systèmes pouvaient désormais permettre le suivi de cibles de jour comme de nuit et au travers de la brume ou de la fumée.

Grâce à ses capacités, l'imagerie infrarouge s'est imposée comme un outil indispensable pour les applications militaires. D'abord limitées au véhicules de taille importante compte tenu de l'encombrement et de la consommation électrique de ces systèmes, les évolutions technologiques

successives ont permis de les utiliser sur une large gamme d'équipements, allant du porte-avion aux lunettes pour fantassin à pied, en passant par les satellites d'observation.

(a) Boule optronique. (b) Nacelle de désignation. (c) Viseur d'hélicoptère.

FIGURE 2.6 – Exemples de dispositifs militaires intégrant des caméra thermiques [1].

La figure 2.6 présente quelques modes d'intégration de caméras thermiques, montées dans des boules optroniques, sur nacelle pour avions ou sur hélicoptère. La caméra thermique est souvent couplée à une voie optique ou d'autres capteurs comme un télémètre laser dans un seul et même appareil.

L'infrarouge pour le domaine terrestre

Montée sur un véhicule ou portée par un fantassin, une caméra thermique aura plusieurs rôles. Elle pourra être utilisée pour le renseignement, la surveillance de zone, la désignation et le suivi de cibles ou le guidage du tir. Généralement, pour les applications sol-sol ou air-sol, les dispositifs d'acquisition utilisés utiliserons les bandes MWIR et LWIR du spectre infrarouge présenté dans la figure 2.1. Les dispositifs sol-ail utiliserons plutôt la bande IR 1.

Par rapport aux systèmes d'imagerie fonctionnant dans le domaine visible ou les dispositifs d'intensification de lumière, une caméra thermique présente quelques avantages supplémentaires. Un fantassin camouflé dans le visible, dans un feuillage par exemple, sera plus facilement détectable dans une image via la chaleur émise par son corps. De la même façon, les véhicules en fonctionnement apparaitront de façon très contrastée sur une image infrarouge à cause de leurs équipements électriques ou leurs moteurs. Ces éléments constituent la signature d'un véhicule et aident à le repérer dans les images infrarouges. En combinant un capteur thermique avec des optiques adaptés, il sera possible pour un fantassin ou un équipage de char d'observer en permanence le champ de bataille.

1. Sources pour les images : figure 2.6a (CC BY-SA 3.0), figures 2.6b 2.6c A. d'Acremont

2.2.2 Méthodes de classification pour l'imagerie infrarouge militaire

Détection-Reconaissance-Identification (DRI)

Grâce à leurs capacités, les caméras infrarouges sont fréquemment utilisées pour les tâches de Détection-Reconaissance-Identification, ou DRI. Présenté dans la figure 2.7, la DRI désigne l'ensemble des opérations nécessaires à la détection et la classification de cibles dans l'infrarouge ou d'autres formes d'images.

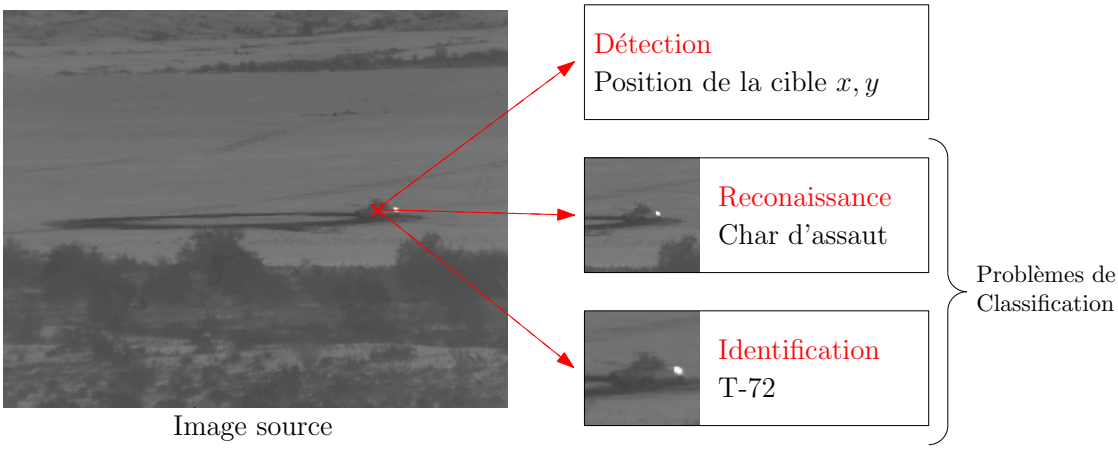


FIGURE 2.7 – Illustration des différentes étapes de la DRI.

Nous allons brièvement présenter chaque étape :

- Détection : Il s'agit de localiser les objets ou véhicules et de déterminer leur position dans l'image.
- Reconnaissance : Cette étape est destinée à associer un type aux objets d'intérêts qui ont été détectés durant l'étape précédente. Pour le cas des véhicules terrestres, cela reviendra par exemple à déterminer s'il s'agit d'une voiture, d'un camion ou d'un char. Il s'agit d'un problème de classification "grossier" avec des classes génériques, et donc avec une grande variabilité intra-classe.
- Identification : Après avoir été détectés puis reconnus, les objets d'intérêt peuvent être identifiés. Il s'agira alors de déterminer précisément son modèle. Dans la figure 2.7, le véhicule est identifié comme un modèle de char d'assaut spécifique, ici un T72. Comme la reconaissance, l'identification est aussi un problème de classification. La classification est cependant plus fine avec moins de variabilité intra-classe.

Automatisation de la DRI

Historiquement cette tache a toujours été effectué par un opérateur humain. Avec le développement des traitements numériques des images et l'introduction d'algorithmes de détection ou de classification automatiques, les fonctions de DRI ont pu commencer à être automatisées

et améliorées. A commence par détection de cibles, de petite ou de grande taille [95][35][65][45], ou le suivi de cibles dans l'image [119][131]. Le problème de la classification de cible dans les images infrarouges, en reconnaissance ou en identification a aussi été traité de façon exhaustive parfois à l'aide d'algorithmes d'apprentissages tels que la classification naïve bayésienne ou les machines à vecteurs de support, ou SVM, [18]. Pour le problème de la classification de cibles en infrarouge, Teutsch et al. [86] ont aussi proposé une méthode de classification basée sur des SVM. Ils l'appliquent à la classification de petites cibles navales en utilisant un ensemble de caractéristiques extraite de l'image source dont les moments de Hu, les matrices de co-occurences, des histogrammes de gradients orientés ou des motifs binaires locaux.

Les excellentes performances des CNN pour les problèmes de classification d'objets à partir d'images visibles ont motivé leur utilisation sur des images issues de différents capteurs. Nous pouvons mentionner le cas des radar, plus particulièrement l'imagerie par radar a synthèse d'ouverture, ou SAR, [102], de l'imagerie par sonar a synthèse d'ouverture, ou SAS, [30]. Les CNN ont aussi été utilisé pour des problèmes concernant des images infrarouges, comme la classification de cibles [101] [66] ou l'amélioration de la qualité des images [13]. Rodgers et al. [101] on notamment proposé une approche basée sur CNN compact à 7 couches cachées qui est présenté dans la figure 2.8. Il sera utilisé dans le chapitre 3 de ce manuscrit.

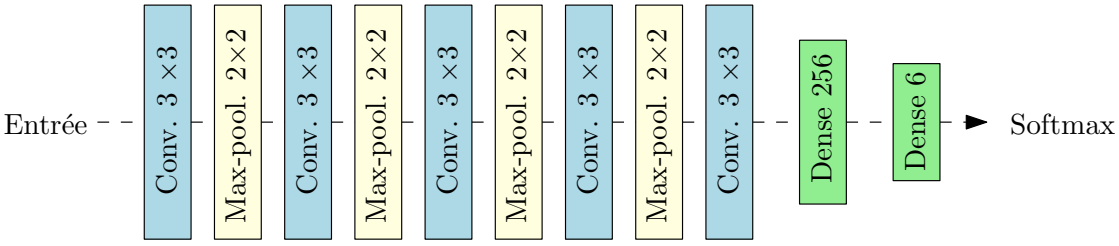


FIGURE 2.8 – Présentation de l'architecture de CNN de Rodgers et al. [101].

La combinaison d'algorithmes de détection et de classification avec des caméras infrarouges ont permis de créer des dispositif de veille passifs qui améliorent fortement la capacité d'un fantassin ou d'un équipage de char à percevoir son environnement. Cependant, le contexte militaire impose quelques contraintes supplémentaires par rapport aux domaine civil. Ces contraintes seront détaillées dans la section 2.4.

2.3 Jeux de données pour la DRI militaire

Nous présentons dans cette section les principaux jeux de données utilisés dans les chapitres suivants. Pour des raisons de confidentialité, nous ne montrerons pas d'images infrarouges réelles ou simulées issues des jeux de données mis à disposition par MBDA.

2.3.1 Images simulées

Les premiers jeux de données dont nous disposons sont faits d'images MWIR simulées créées à partir du logiciel OKTAL-SE dont le fonctionnement simplifié est présenté dans la figure 2.9. OKTAL-SE est un outil de simulation comprenant différents modules qui permettent de générer des images dans différents spectres électromagnétiques, dont l'infrarouge. Les images générées sont en niveau de gris et codées sur 14 bits avec une dimension de 640×512 pixels.

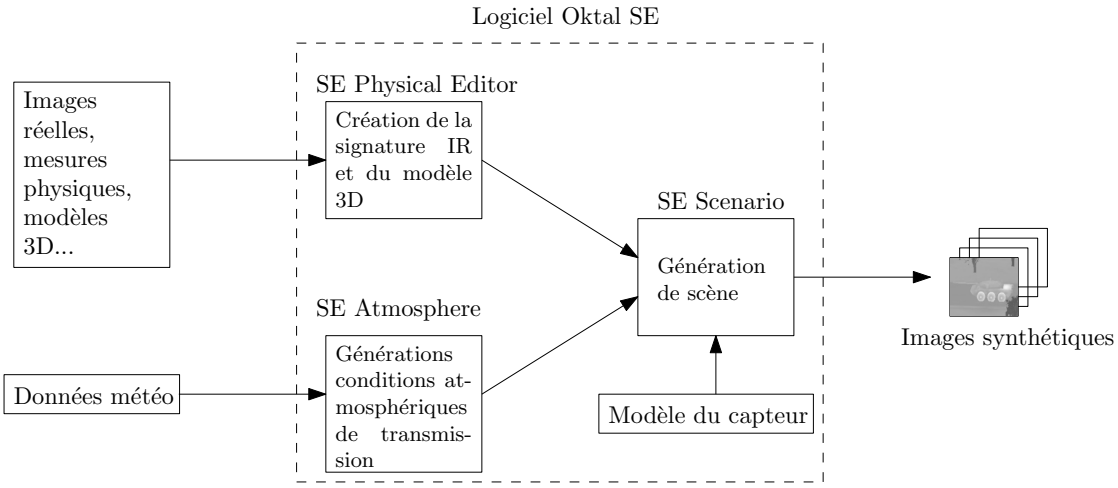


FIGURE 2.9 – Présentation simplifiée des éléments de OKTAL SE utilisés pour générer des images infrarouges.

Les bases de données peuvent contenir jusqu'à huit cibles différentes dont les références dans le domaine visible sont illustrées dans la figure 2.10 et les labels associés sont présentés dans le tableau 2.1. Elles sont réparties en trois ensembles d'un niveau de réalisme croissant :

DS1 : Le premier jeu de données simulées contient uniquement des signatures basiques réalisés à partir de modèles 3D simples. Les points chauds sont des polygones simples de valeur aléatoires et le contraste de la scène peut sembler irréaliste. Par conséquent le réalisme de ces images est assez limité avec des valeurs de températures peu précises. Dans ce jeu de données six véhicules sont présents : m1, m2, c1, c2, c3 et t1. Il y a au total 17280 images soit 2880 images par véhicule.

DS2 : Ce jeu de données est un peu plus réaliste avec des modèles 3D plus fins et des signatures thermiques basées sur des prises de vues réelles. L'aspect global des véhicules est donc plus proche de la réalité. Par rapport au DS1, les huit véhicules de la figure 2.10 sont présents avec un total de 23040 images soit toujours 2880 images par cible.

DS3 : Le dernier jeu de données simulées est similaire au jeu DS2 en terme de structure, de cibles présentes et de nombre d'images. Cependant, les images générées ont été re-calibrées par rapport à des signatures réelles faisant d'elle les images les plus proches de la réalité dont nous disposons pour les expériences.

Classe Reco	Identification	label	Img. DS1	Img. DS2	Img. DS3
Chars	AMX 30B2	m1	2880	2880	2880
	AMX 10RC	m2	2880	2880	2880
Voitures	VAB	c1	2880	2880	2880
	PVP	c2	2880	2880	2880
	VBL	c3	2880	2880	2880
Camions	TRM10000	t1	2880	2880	2880
	GBC180	t2	N/A	2880	2880
	TRM2000	t3	N/A	2880	2880
Nombre d'images au total			17280	23040	23040

TABLE 2.1 – Détails du jeu de données simulé MBDA.

De façon générale, il n'y a qu'une seule cible par image. Il peut cependant y avoir de légers masquages dus au terrain, aux arbres ou aux herbes hautes par exemple. Pour les données simulées, les véhicules sont présentés à partir de points de vue et des conditions météorologiques variés, avec différentes orientations et positions, ainsi que des états variés parmi lesquels on peut citer : véhicule à froid, véhicule statique et moteur allumé, et véhicule en roulement. Chaque image est accompagnée de méta-données contenant résumant l'ensemble des paramètres de l'images comme la météo, la position du véhicule, son type et son état thermique. Elles contiennent aussi la boite englobante du véhicule dans l'image.

2.3.2 Images réelles

SENSIAC

La base de données SENSIAC [90] est une base de séquences d'images visibles et infrarouges, dans la bande MWIR, de véhicules civils et militaires qui évoluent en cercles sur un champ de manœuvre. Il y a au total 10 véhicules différents dans ces séquences.

Parmi les véhicules qui composent ce jeu de données, nous en sélectionnons huit. Leur apparence en optique et en IR sont présenté dans les figures 2.11 et 2.12. Les séquences sont prises à différentes distances qui s'échelonnent tous les 500 mètres, entre 1000 et 5000 mètres. Pour chaque véhicule et chaque distance deux séquences vidéos sont disponibles, une prise de jour et une prise de nuit, pour un total de 3200 images par véhicule et par distance. Le détail complet des cibles sélectionnées dans l'ensemble est donné dans le tableau 2.2.

Comme pour les images simulées MBDA, les images de ces séquences ont une dimension

2. Sources pour les images : figure 2.10a (CC BY-SA 2.0 FR) figure 2.10b (CC BY-SA 4.0) figure 2.10f (CC BY-SA 2.0 FR) figure 2.10h (CC BY-SA 2.0 FR) , figures 2.10c 2.10d 2.10e 2.10g domaine public (CC0 1.0)

(a) AMX-30B2 (b) AMX-10RC (c) VAB (d) PVP

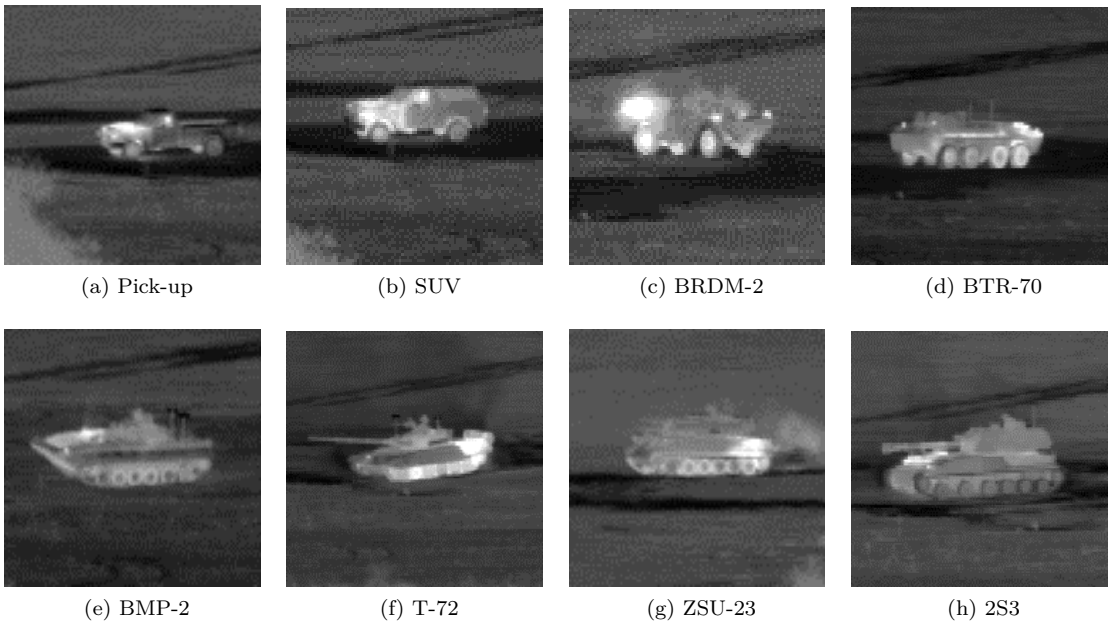
(e) VBL (f) TRM10000 (g) GBC-180 (h) TRM2000

FIGURE 2.10 – Exemples de véhicules présents dans le jeu de données MBDA et utilisé comme références pour les images simulées [2].

(a) Pick-up (b) SUV (c) BRDM-2 (d) BTR-70

(e) BMP-2 (f) T-72 (g) ZSU-23 (h) 2S3

FIGURE 2.11 – Exemples de véhicules présents dans le dataset SENSIAC dans le domaine visible.



(a) Pick-up (b) SUV (c) BRDM-2 (d) BTR-70

(e) BMP-2 (f) T-72 (g) ZSU-23 (h) 2S3

FIGURE 2.12 – Exemples de véhicules présents dans le dataset SENSIAC en infrarouge.

de 640×512 pixels. Elles sont en niveau de gris et codées sur 14 bits. Toutes ces séquences sont accompagnées de méta-données qui incluent notamment la position du centre la cible dans l'image ainsi que son orientation relative au plan de la caméra.

Au-delà de représenter un exemple réaliste d'images infrarouges militaires, les cible sont aussi pertinentes pour faire une première évaluation des algorithmes de DRI. En effet, mis à part le pick-up et le SUV, l'ensemble de ces véhicules sont des véhicules militaires de fabrication russe datant de la guerre froide. Ils ont été exportés en grande quantité dans différents pays du monde et sont encore utilisés par les forces armées de certains de ces pays.

Cependant, le jeu de données présente quelques limites. Pour commencer, il n'y a que deux météo différentes pour l'ensemble des véhicules : jour et nuit. Ces véhicules ne sont capturés qu'en déplacement. De fait, cela limite le nombre de variantes de signatures thermiques inclus dans le jeu de données pour un véhicule.

Données MBDA

En complément des données MBDA simulées, nous disposons aussi d'images MWIR acquises dans le cadre d'essais par les équipes de MBDA. Ce jeu de données comprend six véhicules sur les huit présentés dans la section 2.3.1. Les véhicules c2 et t2 étaient absents lors de la campagne d'essai. Le détail de ce jeu de données, que nous appelerons DR, est disponible dans le tableau 2.3.

Les images ont été capturées avec une caméra infrarouge avec une résolution de 640×512 sur

Classe Reco	Identification	nb séquences	nb image par séquence
Voitures	SUV	32	1800
	Pick-up	32	1800
Transports blindés	BDRM-2	32	1800
	BTR-70	32	1800
	BMP-2	32	1800
Chars	T-72	32	1800
	ZSU-23	32	1800
	2S3	32	1800

TABLE 2.2 – Détails du jeu de données SENSIAC.

Classe Reco	Identification	label simplifié	nb images
Chars	AXM30-B2	m1	15340
	AMX10-RC	m2	5442
Voitures	VAB	c1	4047
	VBL	c3	5117
Camions	TRM10000	t1	7060
	TRM2000	t3	6804

TABLE 2.3 – Détails du jeu de données réel DR.

14bits. Comme pour le jeu de données SENSIAC, cet ensemble comprends des images prises de jour et de nuit pour un nombre limité de conditions météorologiques différentes. Les trajets des véhicules sont cependant plus variés. L'ensemble des éléments du jeu de données DR a été annoté manuellement. Nous disposons donc pour chaque image de la position et des dimensions exactes de la boite englobante pour chaque cible. Comme pour les images simulées, il peut y avoir de légers masquages.

2.3.3 Préparation des données

À partir des données SENSIAC, MBDA réelles et MBDA simulées, nous avons extrait des imagettes de chacun des véhicules en utilisant les méta-données qui accompagnent chaque image. Pour les données MBDA, elles contiennent la position précise d'une boite englobante pour le véhicule présent dans l'image.

Ainsi, pour chaque boite englobante et chaque image de dimension $d_x \times d_y$, nous considérons une imagette carrée de taille $1.1 \times (d_x, d_y)$ et centrée sur le centre de la boite englobante. Le

coefficient de 1.1 permet de s'assurer que la totalité de la cible est présente dans le carré même dans les cas où la boîte englobante n'est pas parfaitement centrée sur la cible. Cette imagette est ensuite re-dimensionnée à 128×128.

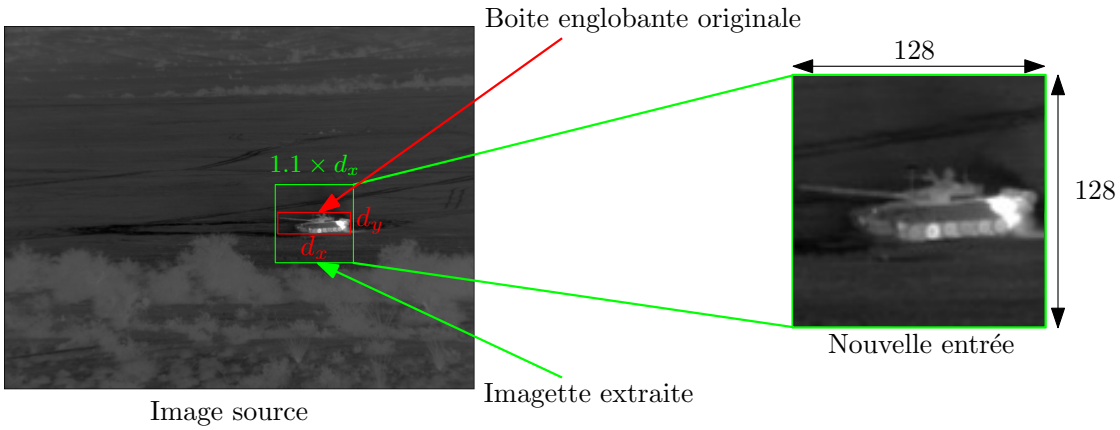


FIGURE 2.13 – Méthode de création des images utilisées dans la suite du manuscrit.

Pour les ensembles DS1, DS2 et DS3, trois sous-ensembles de test contenant 4096 images de validation sélectionnées aléatoirement sont créés. Ils seront nommés respectivement TVS1, TVS2, et TVS3. Les images restantes serviront à former trois jeux d'images d'entrainements nommés respectivement TRS1, TRS2 et TRS3. Ces trois ensembles contiennent respectivement 17280, 23040 et 23040 images. Les images réelles de DR n'étant utilisées que pour tester des algorithmes, L'ensemble DR ne sera pas découpé. La méthode ci-dessus sera utilisé pour toutes les images de DR pour former un ensemble d'imagettes réelles centrées sur chaque cible pour lequel nous conserverons l'appellation DR.

Nous procéderons de façon similaire pour le jeu de données SENSIAC. Cependant, contrairement aux données MBDA, nous ne disposons que de la position du centre de la cible. Pour générer les imagettes, nous avons donc découpé à partir de l'image d'origine des carrés centrés sur la cible dont la dimension correspondait à 1.1 fois la dimension maximale en pixels de la cible dans la séquence.

Enfin la profondeur d'origine des images de simulation et réelles de 14bits a été convertie en *float32*. Seule une normalisation simple entre le maximum et le minimum de l'image et l'intervalle $[0, 1]$ a été effectuée.

2.4 Défis de l'apprentissage profond pour le contexte militaire

Comme pour les cas d'utilisation civile de l'infrarouge décrits dans la section 2.1.2, une scène observée par imagerie thermique variera en fonction des conditions météorologiques et atmosphé-

riques. Les contextes d'utilisations peuvent varier fortement, allant de l'environnement désertique aux steppes enneigées ce qui impose des exigences supplémentaires pour les systèmes utilisant l'imagerie infrarouge. Ces perturbations vont en effet affecter la transmission du rayonnement infrarouge vers le capteur et modifier la signature de la cible.

De plus, l'achat et la distribution des caméra thermiques est réglementé en France par l'article R311-2 du code de la sécurité intérieure, et en Europe par le règlement de la commission européenne n° 428/2009. Certains dispositifs d'acquisition IR peuvent être ainsi catégorisés comme du matériel militaire. Cela rend difficile l'acquisition de caméras thermiques par les acteurs académiques surtout pour des dispositifs dont la résolution ou la fréquence d'image se rapproche de celles de dispositifs militaires.

À cela s'ajoute le fait que les conditions d'observation sont rarement optimales. Les véhicules et objets à détecter vont chercher à rester masqués le plus possible et éviter les zones dégagées, privilégiant les obstacles et les couverts pour se camoufler. Il sera difficile de prévoir l'aspect et la position des véhicules et objets à détecter contrairement, au cas de la conduite autonome ou de l'observation aérienne et spatiale [60]. Dans ces contextes les véhicules a détecter sont observés dans des orientations prévisibles.

Aussi, la signature infrarouge des véhicules à détecter est souvent confidentielle et difficile à obtenir. Cela pose un problème pour développer des algorithmes basés sur l'apprentissage machine car cela limite la capacité de créer des jeux de données suffisamment importants. De plus, pour un véhicule donné il existe de nombreuses variantes, chacune ayant un aspect visuel différent. Par exemple, le T-72 présenté dans la figure 2.11 existe en quatre évolutions (T72, T72A, T72B, T72BM). De même, le VAB, présenté dans la figure 2.10 existe lui en plus de 50 versions différentes. Trois d'entre elles sont présentées dans la figures 2.14.

(a) VAB-T20/DCA (b) VAB Mephisto (c) VAB-ML

FIGURE 2.14 – Trois variantes de VAB visuellement différentes.

Sur ces 50 variantes, une majorité sont proches visuellement, mais certaines présentent des équipements supplémentaires qui modifient fortement leur silhouette comme illustré dans la figure 2.14. Et ce décompte n'inclut pas les différents camouflages ou même les modifications que des équipages peuvent apporter à leur véhicule et en font de ce fait un modèle unique. Au delà d'affecter l'aspect du véhicule dans le domaine visible, la combinaison de ces éléments peut fortement influencer la signature infrarouge de chaque véhicule.

Enfin, il faut aussi noter la difficulté d'interpréter les réponses d'un réseau de neurones et l'absence de méthodes de validation formelle pour les CNN. Il existe bien des méthodes permettant d'interpréter la réponse d'un modèle en analysant visuellement ses paramètres [136], mais elles sont le plus souvent utilisées pour suivre l'apprentissage ou pour diagnostiquer la cause d'une erreur à posteriori. Pour faciliter l'adoption des CNN ils faut donc être en mesure de renforcer la confiance des utilisateurs. À cet effet, il serait intéressant de mettre en place des stratégies permettant d'avertir un utilisateur lors de situations ou la réponse du CNN pour les tâches de DRI est en dehors de son domaine d'utilisation.

2.5 Conclusions du chapitre

Nous avons présenté le principe simplifié du rayonnement infrarouge ainsi que le lien entre la température d'un corps et le rayonnement infrarouge émis par ce corps via l'extension de la loi de Planck. Nous avons aussi vu que ce rayonnement pouvait être observés à l'aide de caméra spécifiques qui disposaient de capteurs réagissant aux rayonnement infrarouges.

L'imagerie infrarouge, ou thermographie, est fréquemment utilisée dans le domaine civil. Ici nous nous sommes concentrés sur son utilisation pour des applications militaires et plus particulièrement pour les tâches de DRI. En effet, l'utilisation de caméras thermiques permet à un opérateur de surveiller l'environnement de jour comme de nuit et de façon discrète. En infrarouge, les véhicules militaires terrestres seront détectables et reconnaissables grâce aux rayonnement émis par les éléments qui chauffent lors de leur utilisation tels que le moteur, les roues ou les équipements électriques. Nous avons aussi noté que, comme pour les images radar ou sonar, les CNN introduits dans le chapitre 1 peuvent être utilisés pour les problèmes de classification de cible en infrarouge.

Nous avons ensuite présenté cinq bases de données d'images infrarouges de véhicules militaires, deux ensembles constitués d'images réelles et trois ensembles constitué d'images simulés de différents niveaux de qualité. Ces ensembles seront utilisés dans le chapitre 3 pour entrainer et évaluer différents modèles de CNN à reconnaitre et identifier des véhicules, et dans le chapitre 4 pour tester différents modules de détection d'anomalies.

Enfin, nous avons identifié deux défis majeurs posés par l'utilisation de réseaux de neurones pour la DRI militaire en infrarouge. Il s'agit de la faible quantité de données disponibles ainsi que de la forte variabilité des signatures infrarouges des véhicules à identifier. Dans la suite de ce manuscrit, nous nous concentrerons sur le problème posé par la faible quantité de données disponibles. Nous proposons dans le chapitre suivant d'utiliser des jeux de données simulés ainsi qu'un réseau de neurones adapté pour compenser une éventuelle absence de données réelles.

Chapitre 3

Nouveau modèle de CNN pour le transfert d'apprentissage de la simulation vers le réel

Contents

Dans ce chapitre, nous considérons que nous utilisons une chaine de traitement modulaire. Celle-ci est constituée de plusieurs étages qui réalisent les opérations d'acquisition, de détection des cibles potentielles, de reconnaissance et d'identification. Les fonctions d'acquisition et de détection ne seront pas étudiées ici. Nous supposons que l'étage de détection fournit aux modules de reconnaissance et d'identification une image contenant une cible plus ou moins centrée. Ce travail se focalise sur les fonctions de reconnaissance et d'identification à l'aide de CNNs. Nous considérons ici que ces deux taches sont effectuées simultanément. Étant donné le contexte mi-

litaire, nous souhaitons étudier la possibilité d'entrainer un réseau à l'aide d'images de synthèse uniquement et de le tester sur des images réelles.

Dans un premier temps, nous présenterons dans la section 3.1 les détails et les motivations de notre approche. Puis nous illustrerons dans la section 3.2 les limites des architectures issues de la littérature pour le transfert de la simulation vers le réel. Nous introduirons ensuite l'architecture que nous proposons, le cfCNN, dans la section 3.3. Ce modèle sera comparé a des modèles issus de la littérature sur le transfert de la simulation vers le réel dans la section 3.4. Enfin, nous présenterons une étude de la robustesse du cfCNN et des modèles introduits dans la section 3.2 sur des images simulant des erreurs de l'étage de détection avant de conclure dans la section 3.6.

3.1 Contexte et méthodes existantes

Nous souhaitons identifier et reconnaitre des cibles militaires en infrarouge à l'aide d'un CNN dans des images réelles. Or, dans la section 2.4, nous avons mentionnés la difficulté d'obtenir suffisamment d'images annotées et de bonne qualité pour former un jeu de données d'entrainement pour un CNN.

En plus du contexte militaire, nous avons vu dans la section 2.1.1 la grande variabilité des signatures d'un objet en infrarouge. Cette variabilité a plusieurs facteurs, comme la météo ou l'état de fonctionnement du véhicule. Nous devons donc trouver une façon de compenser ce manque de données et gérer le problème de la variabilité des signatures.

3.1.1 Présentation de la chaine de traitement

Dans la section 2.2.2, nous avons introduit la notion de DRI et ses différentes étapes. Un schéma général de la chaine de traitement qui regroupe les fonctions d'acquisition et de DRI est présenté dans la figure 3.1.

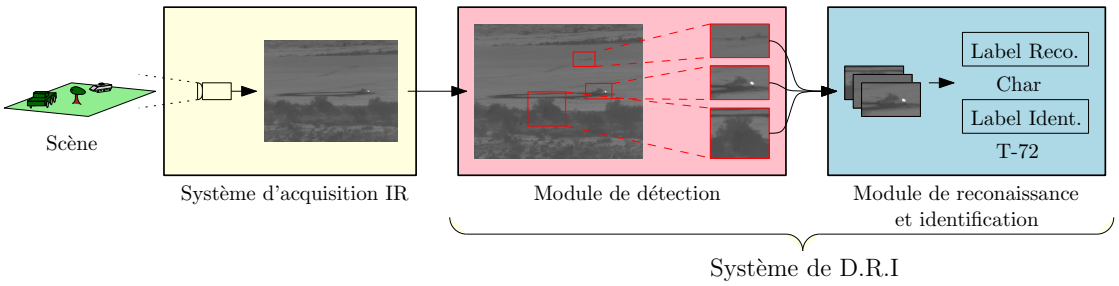


FIGURE 3.1 – Schéma simplifié d'une chaine d'acquisition et de traitement pour la DRI.

Le rôle du système d'acquisition est de fournir les images qui serviront d'entrée au système de DRI. Il est généralement rempli par une caméra infrarouge telle que décrit dans la section 2.1.1. L'image en sortie du système d'acquisition a déjà subi quelques pré-traitements, notamment pour

corriger les déformations liés à l'optique de la caméra ou le bruit spatial fixe de son capteur. Cette image est ensuite envoyée vers le système de DRI.

Les fonctions de DRI sont généralement remplies par des sous-systèmes différents. La détection est traitée en premier pour fournir aux sous-systèmes de RI les zones de l'image présentant les cibles potentielles. Dans l'idéal, les sorties de l'étage de détection seront des images avec une cible parfaitement centrée.

Après la détection, viennent ensuite les fonctions de reconnaissance et d'identification. Ces deux fonctions étant des problèmes de classification, elles peuvent être remplies par le même sous-système avec deux réglages de sensibilité et de précision : Un réglage fin pour l'identification et un réglage plus permissif pour la reconnaissance. Séparer les différentes fonction du système d'acquisition permet de faciliter la validation de la chaine complète. Chaque module peut être réglé et testé séparément. Un module peut être aussi remplacé facilement, sans avoir à modifier les autres éléments de la chaine complète.

Ici nous traiterons les problème de la RI uniquement. Comme il s'agit de problèmes de classification d'images, il est possible d'utiliser des CNN. Cependant, les CNN nécessitent de disposer de suffisamment de données pour les entrainer. Or, comme nous l'avons évoqué dans la section 2.4, il est plus difficile de constituer des bases de données annotées exhaustives dans un contexte militaire que pour des applications civiles, surtout en imagerie infrarouge.

3.1.2 Les limites du transfert d'apprentissage par *fine-tuning*

Compte tenu du contexte militaire, et donc de la difficulté d'obtenir une base de donnée exhaustive, nous pouvons utiliser les techniques de *fine-tuning* ou de *full-model fine tuning* décrites dans la section 1.5.2. Nous pouvons alors utiliser un modèle de réseau de neurones issu de la littérature, pré entrainé sur un jeu de donnée en accès libre. Ce jeu de donnée peut être l'ensemble d'image FLIR ADAS, comme indiqué dans la figure 3.2. Un tel modèle devrait posséder des couches de convolution adaptées à la reconnaissance et l'identification de véhicules en infrarouge. Il suffit alors de remplacer une partie des couches précédant la sortie du modèle pour correspondre au nombre de cibles présentes dans la base MBDA. En théorie, nous pourrions alors simplement utiliser une fraction des données MBDA réelles pour affiner le modèle et obtenir des résultats de classification acceptables. Cependant, comme l'ont évoqué Wang et al. [122], toutes les approches d'affinage ne garantissent pas un niveau de performance minimum pour les taches de classification. Il serait nécessaire de tester *fine-tuning* ou de *full-model fine tuning* pour obtenir des performances adéquates.

Utiliser un modèle pré-entrainé comme source présente aussi un risque de sécurité majeur pour le système final. Nous avons évoqué dans la section 1.5.3 la possibilité de créer des entrées pouvant perturber fortement la réponse d'un CNN. Or, toujours selon Wang et al. [122], il est possible de générer des attaques adverses adaptées au modèle pré-entrainé qui peuvent affecter le nouveau classifieur après le transfert d'apprentissage. Sur le même sujet, Abdelkader et al. [1] ont aussi montré qu'une telle attaque pouvait aboutir, même sans connaitre le jeu de données

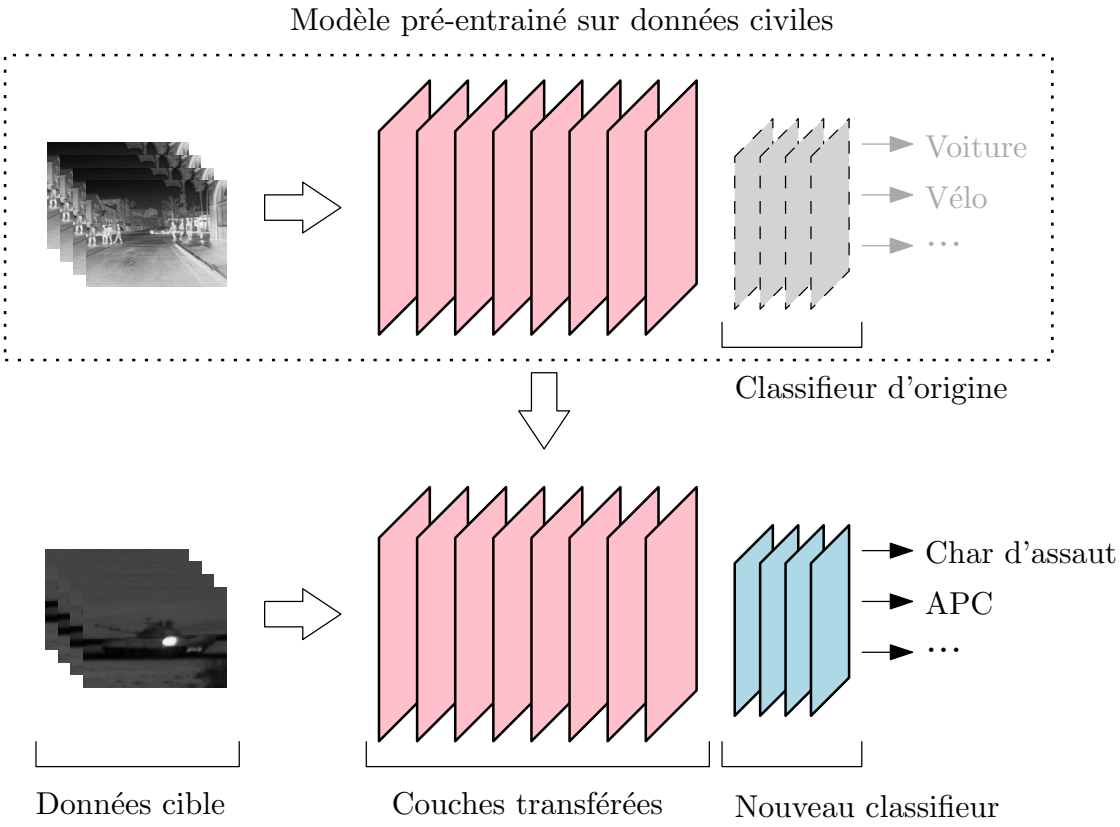


FIGURE 3.2 – Transfert de domaine depuis un jeu de données d'images IR civiles vers des données militaires.

utilisé pour le réglage fin ou les labels des sorties attendues. Il est donc possible d'attaquer indirectement un modèle si celui-ci a été créé ou entrainé à partir de données vulnérables.

Le principe du transfert d'une attaque adverse de type *whitebox* selon Abdelkader et al. est décrit dans la figure 3.3. En générant des exemples adverses sur un modèle vulnérable, un attaquant est en mesure de créer des exemples adverses sur toute nouvelle architecture qui réutilise une partie des couches de ce modèle. Comme illustré dans la figure 3.3 ces attaques sont non-ciblées et ont pour seul but de faire changer la réponse du réseau de neurones utilisé pour la classification. Tant que la prédiction du modèle attaqué a changé, l'attaque est considérée comme efficace.

Une autre forme de vulnérabilité introduite par l'utilisation d'un modèle pré-entrainé est l'empoisonnement des données d'entrainement [44]. Le fonctionnement simplifié de cette forme d'attaque est présenté dans la figure 3.4. Avec cette méthode il est possible de faire transférer vers le réseau ciblé des motifs pouvant perturber ses résultats de classification.

La vulnérabilité aux attaques transférées est d'autant plus importante qu'il est possible de créer des attaques dans le monde physique sous forme de patchs qui viendraient perturber les

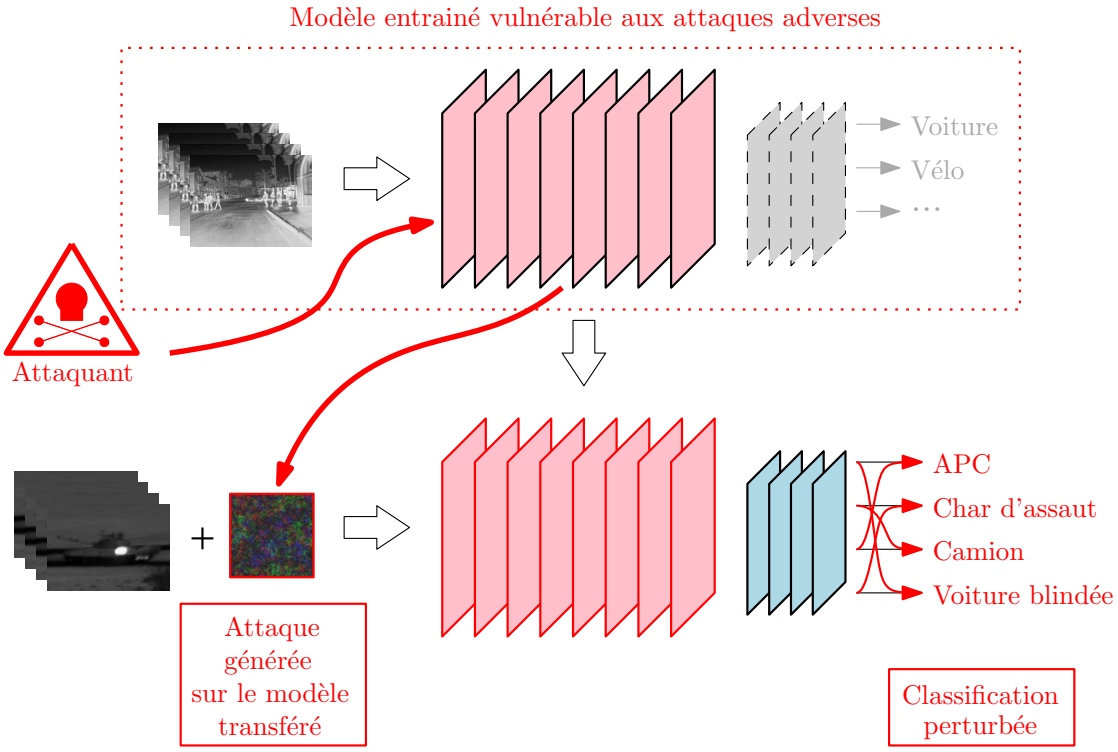


FIGURE 3.3 – Transfert d'attaques adverses non ciblées.

résultats de classifications d'un CNN comme l'ont montré Kurakin et al. [71] ou Eykholt et al. [61]. Il n'est donc pas nécessaire pour un attaquant d'avoir accès au flux d'images entrant dans le réseau de classification pour perturber son fonctionnement.

Dans un contexte militaire, où une mauvaise identification peut avoir de graves conséquences, il semble donc préférable de choisir une approche où l'on dispose d'un contrôle total sur le jeu de données utilisé pour entrainer nos modèles. Nous proposons donc de ne pas recourir à l'affinage et d'entrainer notre modèle de classification de zéro. Nous utiliserons pour l'entrainement un jeu de données d'image simulées. Cela nous permet à la fois de contrôler les données d'apprentissage, et de compenser la faible disponibilité de données infrarouges militaires réelles.

3.1.3 Utilisation de données simulées pour l'apprentissage

Plutôt que d'utiliser un réseau pré-entrainé et affiné, nous utiliserons donc un modèle entrainé de zéro. Dans notre contexte, afin de disposer de suffisamment d'images pour que l'entrainement puisse converger, nous utiliserons des données simulées. Le modèle appris sera ensuite testé et validé sur des données réelles.

Des jeux de données simulées ont été utilisés avec succès dans plusieurs domaines. Il n'est pas toujours nécessaire de disposer d'images de simulation réalistes pour obtenir des résultats

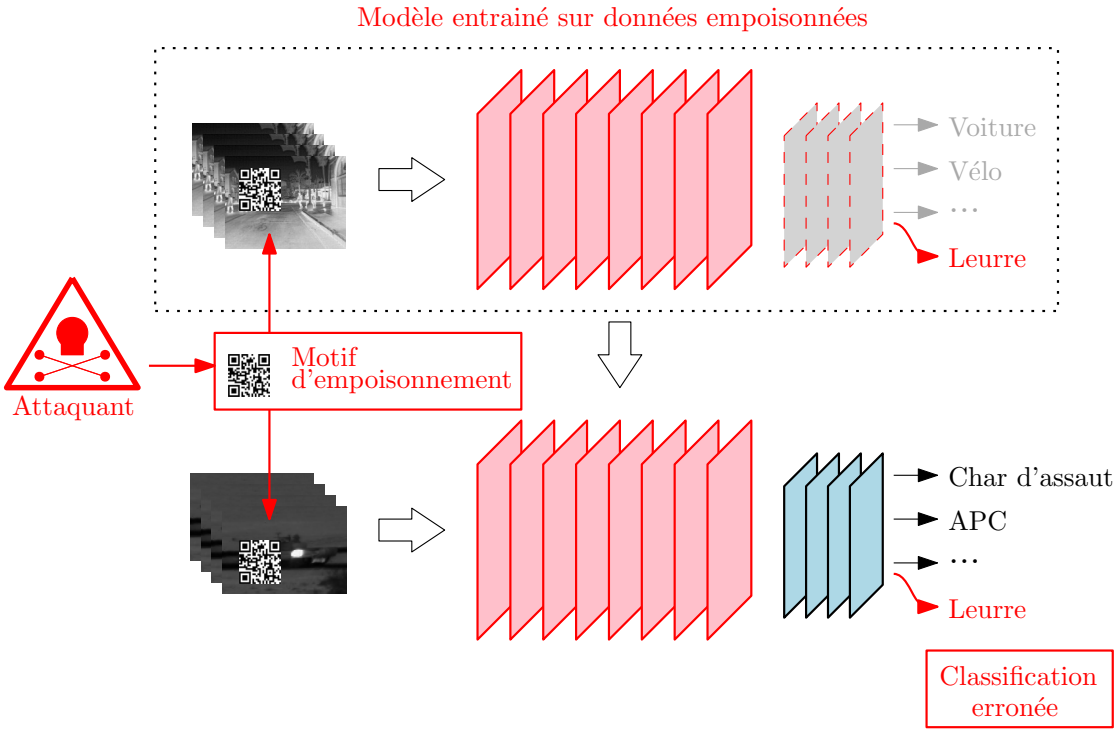
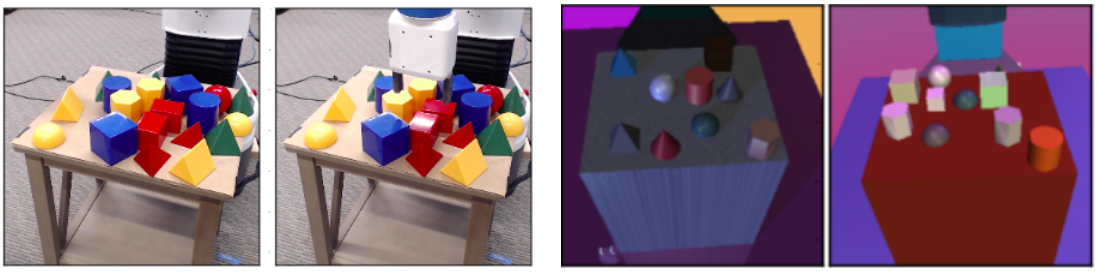


FIGURE 3.4 – Attaque de modèle transféré par empoisonnement des données.

satisfaisants comme montré par Tobin et al. [58] pour diriger un bras robotique. Ce bras robotique est équipé d'une caméra et doit localiser différents objets sur une table pour les saisir. Étant donné la difficulté de réaliser des scènes simulées photo-réalistes, Tobin et al. proposent de générer des scènes où les couleurs et les conditions d'illuminations sont très variées pour forcer un modèle d'apprentissage à se concentrer sur les formes et la position de ces formes dans l'image. Dans le cas de cette expérience, les images d'entrainement générées pouvaient inclure des exemples dont les couleurs étaient très différents des images utilisées pour le test, comme illustré dans la figure 3.5. D'après leurs résultats, augmenter le nombre de variations dans les textures utilisées pour les objets des images simulées permet de réduire significativement l'erreur de localisation de leur modèle.

Chen et al. [17] ont aussi utilisé la simulation pour des problèmes de conduite autonome. Via l'utilisation d'un simulateur, ils ont été en mesure de présenter à l'entrainement de leur modèles une grande variété de situations de conduite. Le recours à la simulation a aussi permis d'obtenir directement les annotations correspondantes sans intervention humaine. Comme pour Tobin et al., le recours à la simulation a permis d'améliorer les performances de leur modèle de conduite. L'approche utilisée dans ces deux exemples est assimilable à l'augmentation de données vues dans la section 1.4.5 de ce manuscrit. Les variations seront plus complexes que de simples translations ou rotations. Elles couvrent des changements de position des objets, de leur couleur, leur texture



(a) Images réelles. (b) Images simulées.

FIGURE 3.5 – Exemples d'images réelles et simulées utilisées par Tobin et al. [58].

et de l'éclairage de la scène.

Pour l'imagerie SAR, où la quantité d'images disponible pour créer un jeu de données exhaustif est limitée, le recours à la simulation pour l'entrainement de réseaux de neurones a aussi été étudié [29]. Contrairement aux travaux de Tobin et al. et Chen et al., les images SAR simulées ont été utilisées pour pré-entrainer un réseau de neurones. Après ce pré-entrainement, des données SAR réelles ont été utilisées pour affiner le réseau utilisé.

L'utilisation d'images simulées a aussi été combiné avec succès à des couches de convolution améliorées [64], ou des techniques d'apprentissage non-supervisées [54].

En se basant sur ces différents résultats, nous pouvons considérer qu'il est possible d'entrainer un CNN pour de la classification de cibles dans des images infrarouges à partir de données simulées. De plus, le recours à la simulation nous permet de générer un ensemble de données qui couvre une large variété de conditions météo, d'éclairement, d'état thermique et de positionnement des cibles.

3.1.4 L'encapsulation pour améliorer la classification

Les éléments présentés dans la section 3.1.3 ont montré qu'il était possible d'utiliser des images simulées pour entrainer un réseau de neurones destiné a traiter des séquences d'images réelles. Le recours à la simulation reste cependant couteux en temps et en moyens, notamment dans le cas de l'imagerie infrarouge, compte tenu de la variété de signatures et de scènes qu'il serait nécessaire de générer pour constituer une base de données exhaustive. Nous aimerions donc mettre en place une stratégie d'apprentissage qui permette de réduire le nombre d'images de simulation a utiliser pour entrainer un réseau de neurones. Réduire la quantité d'images à produire permettrait aussi de ré-allouer une partie du temps passé à générer une grande quantité d'image pour l'amélioration de la qualité de la synthèse.

Idéalement, cette stratégie ne doit pas affecter la capacité du réseau entrainé sur données simulées à généraliser sur des données réelles. Nous sommes donc confrontés à une situation où nous aimerions entrainer un réseau de neurones à partir d'un nombre limité d'images mais qui

doivent couvrir une grande variété de situations. Il s'agit donc un problème avec une grande variabilité intra-classe, couplé à des similarités inter-classes.

Cette situation est fréquemment rencontrée dans les problèmes de classification d'images satellites. Certains problèmes nécessitent de créer des algorithmes capables de classer des bâtiments ou des types de sols avec un nombre limité de données d'entrainement annotées. Pour renforcer les performances des CNNs utilisés pour cette tâche, Cheng et al. [37] ont suggéré d'utiliser l'encapsulation. Le principe de leur approche est présenté dans la figure 3.6. Elle introduit une fonction de coût totale J_{total} qui est l'addition de J_{train} et d'un nouveau terme appelé $J_{metrique}$. Cette grandeur $J_{metrique}$ est calculé de façon à minimiser la distance intra-classe et maximiser la distance inter-classe au sein d'un lot d'apprentissage.

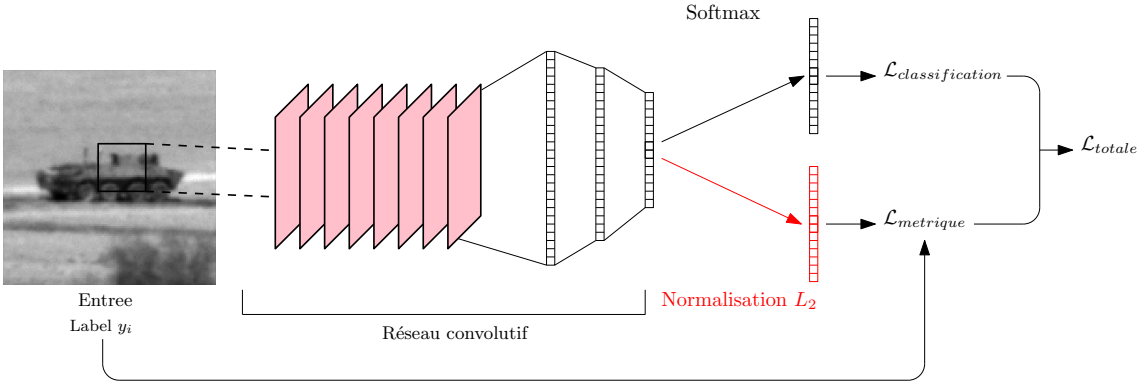


FIGURE 3.6 – Utilisation de l'encapsulation pour améliorer les performances de classification des CNN selon Cheng et al. [37].

Cheng et al. ont aussi observé que, conformément à leur intuition, les images d'une même classe sont mieux regroupées entre elles, et les classes sont mieux séparées. La figure 3.7 présente de façon simplifiée l'effet de l'ajout du terme $J_{metrique}$ à l'erreur totale d'apprentissage. En maximisant la distance inter-classes, en rouge, et en minimisant la distance intra-classe, en bleu, l'encapsulation améliore la séparation de chaque classe en sortie du réseau.

Étant donné la grande variabilité de signatures que nous sommes susceptibles de rencontrer, nous allons dans la suite de ce chapitre tester cette approche d'encapsulation pour essayer d'améliorer les performances de classification d'un réseau de neurones sur des images infrarouges. Elle a été privilégiée par rapport aux approches par paires [100] ou par triplet [123] car elle prends en compte la contribution de l'ensemble des exemples présent dans un lot pour construire le vecteur d'encapsulation. Ces résultats sont présentés dans la section 3.4.4.

3.1.5 Robustesse des CNN aux perturbations

Dans la section 3.1.1, nous avons mentionné que le modèle de réseau de neurones dédié à la RI se trouve après un étage de détection. En conditions réelles, cette détection n'est pas systématiquement parfaite. Les images présentées au système RI peuvent être perturbées avec

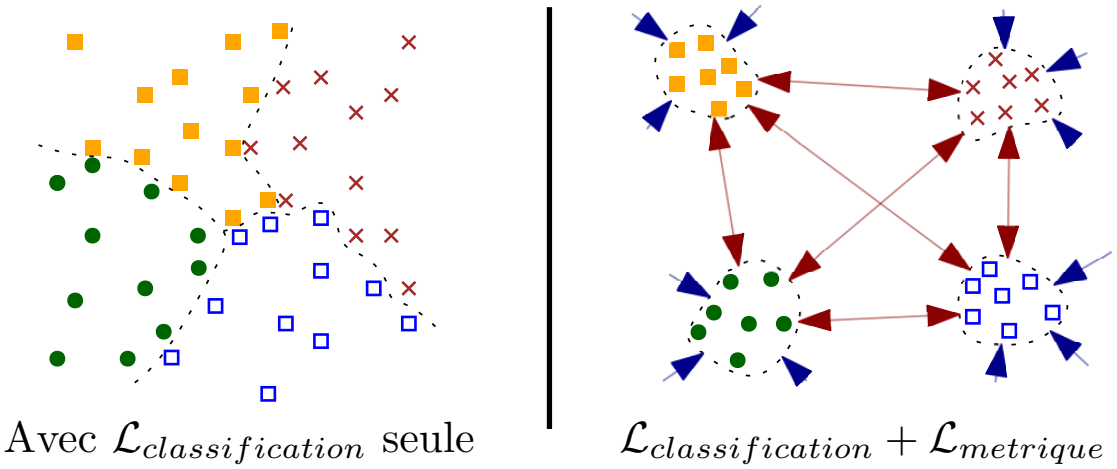


FIGURE 3.7 – Effet de l'encapsulation sur la répartition des classes en sortie d'un CNN selon Cheng et al. [37].

une cible qui peut ne pas être parfaitement centrée. Il faut donc que le système RI, dans notre cas un CNN, soit robuste à ces perturbations.

En visualisant le comportement des couches de convolution d'un CNN, Zeiler et al. [135] ont observé que les dernières couches des CNN contribuaient à la robustesse du modèle aux translations de l'image d'entrée. Ces observations sont confirmées par Bakry et al. [9] qui confirme que les couches denses les plus profondes sont celles qui contribuent le plus à cette invariance. Cette idée peut sembler contre intuitive étant donné que les premières observations sur les CNN indiquaient que l'enchainement de couches de convolution et de *max-pooling* à l'entrée des CNN contribuaient à améliorer la robustesse de l'architecture aux petites perturbation , comme nous l'avons évoqué dans la section 1.3.2. L'ensemble de ces études montrent cependant que les CNNs restent sensibles aux rotations de l'image d'entrée.

Les réseaux convolutifs parviennent donc à apprendre des caractéristiques invariantes par translation. Ils conservent cependant l'information spatiale de l'image d'entrée, qui peut être exploitée pour des tâches de localisation comme dans les deux études [85] [73]. Dans ces travaux, les réseaux utilisés pour localiser des objets ne sont entrainés qu'avec les labels des classes de ces objets. Cela confirme donc que les CNNs finissent par apprendre une représentation invariante aux petites perturbations spatiales.

Il est aussi possible de créer des modèles de CNN robustes à des transformations plus variées, comme les homographies ou les transformations affines. La robustesse est obtenue en ajoutant à un CNN des modules qui corrigent les transformations. Les réseaux qui sont équippés de ces modules sont appelés *Spatial Transformer Network* ou STN [59].

Les *Spatial Transformer Network*, ou STN, sont des architectures conçues pour apprendre simultanément à classer des images et à être invariant aux transformations affines de l'image d'entrée. Le réseau STN est une adaptation du modèle de CNN classique auquel on ajoute un

sous-réseau qui sera dédié à localiser les éléments a classer et à les transformer pour compenser une éventuelle déformation. Un schéma simplifié du modèle est présenté dans la figure 3.8

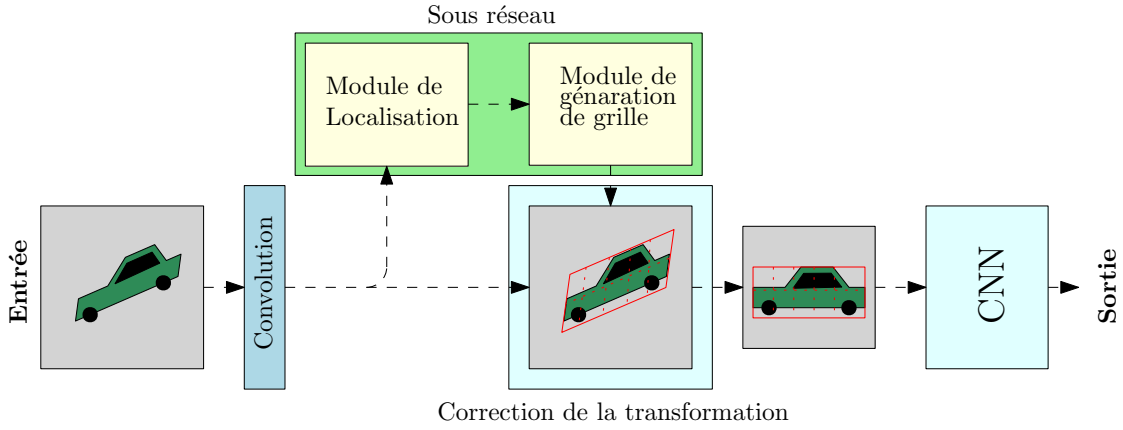


FIGURE 3.8 – Schéma simplifié d'un *Spatial Transformer Network*.

Ce sous-réseau comprends deux élément, un réseau dédié à la localisation de l'objet d'intérêt, et un réseau qui sera chargé de générer une grille correspondant à une estimation de la transformation de l'image d'entrée. Le module de localisation génèrera une matrice de transformation qui sera utilisée par le module de génération de grille pour corriger l'entrée qui pourra ensuite être traitée par un CNN. L'entrainement d'un STN est cependant plus long qu'un CNN simple car il introduit de nouveaux paramètres à optimiser pendant l'apprentissage.

L'utilisation du *Global Average Pooling* [79] peut aussi renforcer la robustesse aux translations et rotations de l'entrée déjà observée pour les CNN sans GAP. Les couches GAP peuvent aussi être utilisés pour la localisation d'objets avec des réseaux de neurones convolutifs [8].

3.1.6 Présentation de l'approche

Compte tenu des éléments présentés dans les sections sur l'utilisation de la simulation en 3.1.3 nous proposons l'approche décrite dans la figure 3.9 pour résoudre notre problème de classification. Celle-ci se base sur un CNN qui sera pré-entrainé sur des données simulées, présentées dans la section 2.3.1 puis testé sur des données réelles, présentées dans la section 2.3.2. Le CNN sera entrainé pour résoudre le problème d'identification. Nous utiliserons ensuite la prédiction du CNN pour déterminer la classe de reconnaissance des cibles. En effet, comme indiqué dans la section 2.3, chaque cible dispose d'un label d'identification et d'un label de reconnaissance. L'idée étant qu'une cible mal identifiée, aura un label de reconnaissance proche de celui le la vrai cible.

Étant donné la popularité de modèles comme VGG19, Inception ou Resnet, nous proposons d'évaluer ces CNN issus de la littérature et dédiés à la classification d'image pour cette tâche. En complément de ces modèles nous ajouterons un CNN dédié à la classification d'images IR [101]

Enfin nous proposons un CNN, présenté dans la section 3.3, construit spécifiquement pour être entrainé sur des données simulées et testé sur des données réelles. L'ensemble de ces réseaux de neurones seront comparés à un algorithme de classification d'image basé sur la classification de descripteurs à l'aide d'une machine a vecteurs de support. Les modèles choisis seront présentés dans la section 3.2.1.

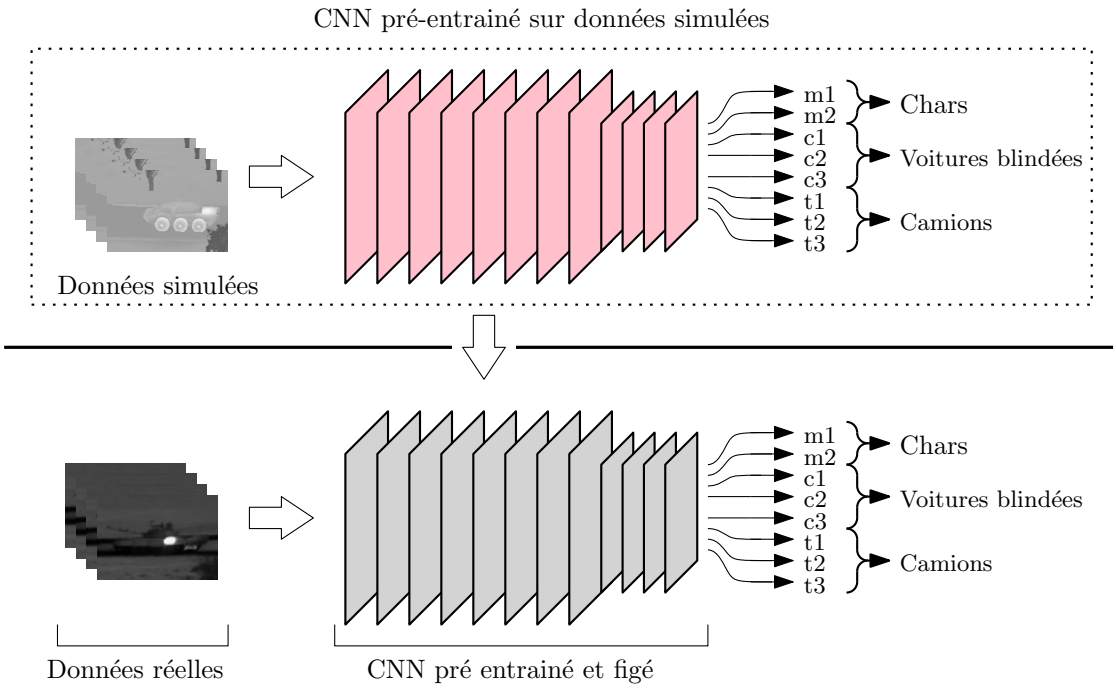


FIGURE 3.9 – Présentation de notre solution.

Les performances de ces modèles de classification pour la RI dépendent de plusieurs paramètres et notamment de la qualité de la simulation. Cependant, les images simulées ne sont jamais parfaitement réalistes. Le niveau de détail des modèles 3D utilisés ainsi que les performances du moteur de rendu ont une influence sur la qualité finale des images. Elles ne seront jamais identiques aux images réelles. Nous souhaitons donc mesurer l'influence de la qualité de la simulation sur les performances des modèles de classifications d'image choisis sur des données réelles. Aussi, en lien avec les éléments présentés dans la section 3.1.4 sur l'encapsulation, nous testerons l'utilisation de l'encapsulation pour essayer de réduire le nombre d'images de simulation nécessaire à l'entrainement du modèle de CNN que nous proposons.

Enfin, la qualité de la prédiction des CNN pour la RI sera aussi dépendante des autres éléments de la chaine complète d'acquisition et de DRI, notamment de l'étage de détection. En condition réelles ce dernier peut extraire de l'image source des entrées pour l'étage RI perturbées. Les cibles à identifier sont généralement non-coopératives et peuvent être translatées, partiellement masquées ou voir leurs échelle modifiée dans l'image qui sera présentée au système RI. Nous

souhaitons donc évaluer les performances des modèles de classification choisis sur des entrées perturbées pour estimer leur robustesse aux erreurs de détection.

3.2 Limites des architectures existantes

3.2.1 Modèles sélectionnés

Pour avoir un aperçu des performances des CNN sur le problème du transfert de la simulation vers le réel, nous avons sélectionné plusieurs modèles de CNN issus de la littérature. Les résultats de ces modèles serviront aussi de référence pour évaluer l'apport d'une architecture dédiée. Les modèles choisis sont les suivants :

- le CNN présenté par Rodger et al. [101] dont nous présenterons les résultats avec le label "R et al.". Ce modèle de CNN simple a déjà montré des performances intéressantes en identification de véhicules dans des images infrarouges.
- VGG19 [108]. Un modèle de CNN simple qui a montré de bonnes performances sur une large variété de problèmes de vision par ordinateur.
- Le CNN bilinéaire ou BCNN tel que présenté dans [115], dont les deux branches sont basées sur VGG16 [108]. Ce modèle représente une alternative légèrement plus performante que le modèle VGG.
- Inception v3 [114] est une architecture récente ayant obtenus de très bons scores sur le challenge Imagenet [32]. Il est souvent utilisé comme modèle pré-entrainé sur différents problèmes.
- Resnet 52 [48] pour compléter Inception v3. Il s'agit aussi d'une architecture récente, performante et populaire pour les problèmes de classification d'images. De plus, ce modèle est entièrement convolutif et utilise une couche de GAP en sortie.

Ces six architectures seront aussi comparées à un modèle d'apprentissage machine pour la classification d'objets de type SVM utilisant des descripteurs HOG, ou *Histogram of Oriented Gradient* pour l'extraction des caractéristiques de l'image. Ce modèle se base sur des travaux effectués en interne chez MBDA pour la classification de cibles en infrarouge.

Afin d'évaluer uniquement l'impact de l'apprentissage sur des données simulées et réalité, les images d'entrainement et de test contiendront une cible est parfaitement centrée. Cela correspond à un fonctionnement parfait de l'étage de détection.

3.2.2 Effets de la représentativité de la synthèse

Dans un premier temps, nous allons mesurer le gain apporté par l'amélioration de la qualité des images de synthèse des bases TRS2 et TRS3 par rapport à TRS1. Chaque architecture a été entrainée à dix reprises au minimum sur les ensembles TRS1, TRS2 et TRS3, puis validée sur les ensembles respectifs TVS1, TVS2 et TVS3. Leur performance est ensuite testée sur un ensemble constitué uniquement d'images infrarouges réelles DR.

Pour évaluer les algorithmes de classification, nous utiliserons deux critères :

- Score d'identification : le pourcentage de prédictions exactement identiques au label attendu sur les données réelles.
- Score de reconnaissance : le pourcentage de prédiction dont la classe, *i.e.* le type de véhicule (e.g. camion, tank, voiture...), est identique à la classe du label attendu.

Ces deux scores sont construits pour se rapprocher du principe de la DRI, pour Détection-Reconaissance-Identification présenté dans la section 2.2.2. Le score de reconnaissance a été privilégié par rapport à un score de type top-N car il est plus intéressant d'un point de vue opérationnel. En effet, en cas d'identification incorrecte, la classe de reconnaissance du véhicule est une information importante pour une meilleure compréhension de la situation tactique.

Données d'entrainement	**TRS2**		**TRS3**	
Architecture	**Ident.**	**Reco.**	**Ident.**	**Reco.**
SVM	3.2%	10.20%	8.8%	16.11%
Rodger et al.	3.34%	8.36%	8.43%	9.27%
VGG19	14,27%	20.68%	23.9%	27.89%
BCNN	3.38%	14%	4.48%	12.64%
Inception v3	0.79%	10.68%	9%	25.77%
ResNet 52	21.51%	26.52%	25.68%	27.49%

TABLE 3.1 – Gains de performance relatifs en test pour les scores d'identification et reconnaissance de chaque architecture sur les images réelles (DR), par rapport à la même architecture de référence entrainée uniquement sur TRS1.

Les résultats regroupés dans le tableau 3.1 montre directement l'amélioration des performances sur les deux critères d'évaluation. Le gain de performance est notamment très important pour les scores reconnaissances ; avec une très nette amélioration de 25.77% pour Inception v3 et de plus de 27% pour VGG19 et Resnet 52. Néanmoins, les gains sont plus limités pour les performances en identification. La variation des performances semble très dépendante de l'architecture.

3.2.3 Comparaison avec un SVM-HOG

Nous présentons dans le tableau 3.2 les performances des modèles de CNN issus de l'état de l'art au SVM-HOG sur les images de DR après avoir été entrainé sur les bases d'images simulées TRS1, TRS2 et TR3. Compte tenu des contraintes de confidentialité Les scores du tableaux correspondent au gain de performance relatif par rapport aux scores obtenus par le SVM-HOG.

Les résultats du tableau 3.2 montrent bien qu'il est difficile de surpasser les performances du SVM pour le cas du transfert de domaine. Nous espérions voir un gain de performances plus important pour les différents CNN par rapport au SVM. Entrainer sur des données simulées à un

Données d'entrainement	**TRS1**		**TRS2**		**TRS3**	
Architecture	**Ident.**	**Reco.**	**Ident.**	**Reco.**	**Ident.**	**Reco.**
R et al.	1.44%	1.78%	2.96%	−2.47%	1.92%	−9%
VGG19	−14.9%	−16.28%	4.84%	0.49%	0.27%	−4.35%
BCNN	1.31%	−7.13%	2.91%	−2.87%	−4.37%	−5.62%
Inception v3	−2.47%	−15.83%	−9.35%	−10.98%	−1.25%	7.82%
Resnet 52	−10.47%	−12.03%	2.02%	−1.09%	2.92%	0.3%

TABLE 3.2 – Gains de performance en identification et reconnaissance des modèles sélectionnés sur les images réelles DR par rapport à un SVM avec descripteurs HOG entrainé sur les mêmes jeux de données.

impact significatif sur les performances des réseaux de neurones. L'utilisation de techniques telles que le *fine-tuning* ou les *gradient-reversal-layers* [130] pourraient servir à améliorer ces résultats, mais ne sont pas évaluées ici.

On peut aussi noter les bons scores obtenus par l'architecture de Rodger et al. [101] qui, malgré sa taille réduite, est la seule qui dépasse les performances en identification du SVM sur les trois jeux de données.

A la lumière des éléments présentés dans les tableaux 3.1 et 3.2, nous proposons un nouveau modèle de réseau de neurones inspiré de R et al., construit spécifiquement pour le problème du transfert d'apprentissage depuis nos données simulées vers les données réelles.

3.3 Présentation du cfCNN

3.3.1 Détails de l'architecture

En se basant sur les apports d'une architecture entièrement convolutive [109], nous proposons une architecture de taille réduite qui comprend environ 130 000 paramètres, tout en conservant des performances adéquates pour notre problème. Nous l'appellerons cfCNN pour *compact and fully convolutional neural network*. Le réseau est présenté en détails dans la figure 3.10 et le tableau 3.3.

Ce réseau convolutif comprend 7 couches de convolution entrecoupées par trois opérations de *Max-Pooling*, et se termine par une couche de *Global Average Pooling* (GAP). La sortie de la couche GAP est directement passée au travers d'une couche Softmax pour produire les scores de classification du réseau. L'ajout de cette couche GAP présente deux avantages majeurs par rapport à une couche entièrement connectée : la conservation de l'information spatiale de chaque cible jusqu'à la dernière couche et le calcul de la moyenne sur chaque filtre rendent la sortie légèrement plus robuste aux petites variations de l'entrée comme les translations. Nous illustrerons cela dans la suite de ce chapitre.

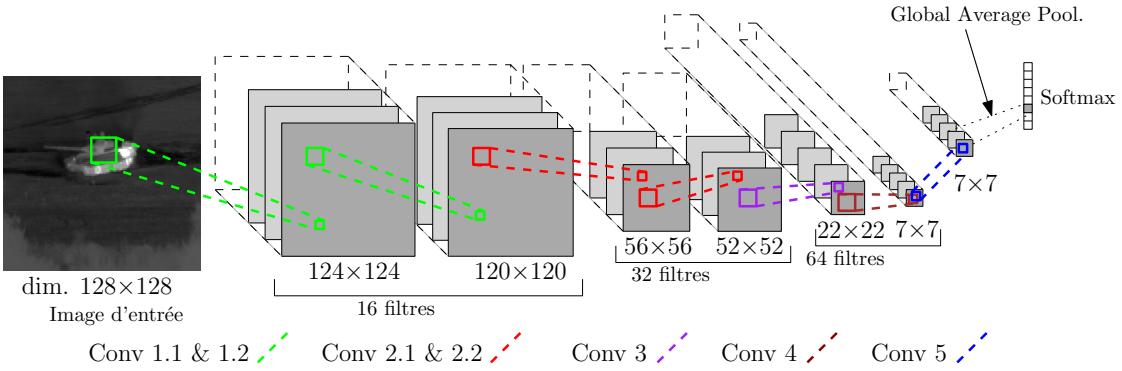


FIGURE 3.10 – Un réseau de neurones compact et entièrement convolutif utilisé pour la classification d'images infrarouges, le cfCNN.

Couche	**Nombre de filtres**	**Taille du noyau**	**Dimensions de la sortie**
Conv 1.1	16	5 × 5	124 × 124
Conv 1.2	16	5 × 5	120 × 120
Max-pool 2 × 2			60 × 60
Conv 2.1	32	5 × 5	56 × 56
Conv 2.2	32	5 × 5	52 × 52
Max-pool 2 × 2			26 × 26
Conv 3	64	5 × 5	22 × 22
Max-pool 2 × 2			11 × 11
Conv 4	64	5 × 5	7 × 7
Conv 5	8	1 × 1	7 × 7
Global average pooling			8 × 1
Softmax			8 × 1

TABLE 3.3 – Détails de l'architecture proposée.

En lieu et place de la fonction d'activation ReLU couramment utilisée dans les architectures de réseaux de neurones convolutifs, l'architecture du cfCNN utilise la fonction *Leaky-ReLU* ou LReLU. En effet, la pente légèrement négative de la fonction LReLU est une façon d'atténuer le problème de disparition des gradients lors de l'apprentissage, et permet d'éviter le phénomène de mort prématurée de neurones dans le réseau. Ce changement va permettre une amélioration des performances de classification lors du cas du transfert de l'apprentissage depuis un ensemble de données simulées vers des données réelles, qui sera décrit dans la suite de ce chapitre.

Par validation croisée, l'initialisation des poids du réseau selon la méthode de He et al. ainsi que l'algorithme Adam ont été retenus pour l'entrainement du cfCNN. Nous avons choisi un

taux d'apprentissage de départ de 10^{-3} pour Adam. À cela vient s'ajouter un critère d'arrêt anticipé de l'apprentissage si l'erreur de validation augmente avant d'atteindre le dernier cycle. Nous utilisons aussi la technique de régularisation *Drop-out* avec une probabilité d'extinction de 0.2. Cette valeur est recommandée par Srivastava et al. [110] dans le cas où le *Drop-out* est utilisé pour des couches de convolution.

3.3.2 Variantes du cfCNN

En complément du cfCNN standard, nous introduisons quatre variantes :

- cfCNN avec des activations ReLU.
- cfCNN(fc), ou *Fully Connected*, avec des couches de neurones denses.
- cfCNN(STN), une architecture combinant un cfCNN et un STN.
- cfCNN(LSE) avec encapsulation LSE.

Les deux premières variantes auront pour but de valider les choix de l'activation ReLU et du GAP pour le cfCNN. La troisième variante sera utilisée pour évaluer l'utilisation d'un STN pour améliorer la robustesse du cfCNN aux perturbations. Nous l'utiliserons aussi pour mesurer l'impact de l'ajout du STN sur les performances de classification. La dernière variante servira a évaluer l'apport du LSE pour la réduction du nombre d'images utilisées pour l'entrainement.

Variante avec l'activation ReLU

Pour montrer l'intérêt de l'activation LReLU pour le transfert de la simulation au réel, nous utiliserons dans la section 3.4.2 une variante du cfCNN utilisant la fonction ReLU. Cette activation sera utilisée pour l'ensemble des unités du réseau excepté ceux de la couche de sortie qui utiliserons l'activation Softmax. Mis à part ce changement, la structure du réseau sera identique à celle du cfCNN standard. Le nombre de couches, de filtres de convolution et leurs dimensions seront exactement les mêmes.

Variante *Fully Connected*

Afin de justifier l'apport du GAP et d'une architecture entièrement convolutive, nous introduisons aussi une variante dite *fully-connected* du cfCNN. Cette variante sera appelée cfCNN(fc).

Ce modèle, illustré dans la figure 3.11, diffère du cfCNN standard dans ses dernière couches. La couche GAP a été remplacée par un MLP à trois couches cachées : deux couches denses, avec respectivement 256 et 8 neurones, et une couche Softmax.

Variante avec STN

Cette version du cfCNN sera simplement complétée par un STN. Le STN sera placé entre les couches Conv 1.1 et Conv 1.2 du cfCNN. Il comprendra un total de 7 couches cachées : Deux couches de convolution, avec respectivement 16 et 32 noyaux de dimension 3×3, deux couches de max-pooling et trois couches de neurones denses, avec respectivement 128, 64 et 6 neurones.

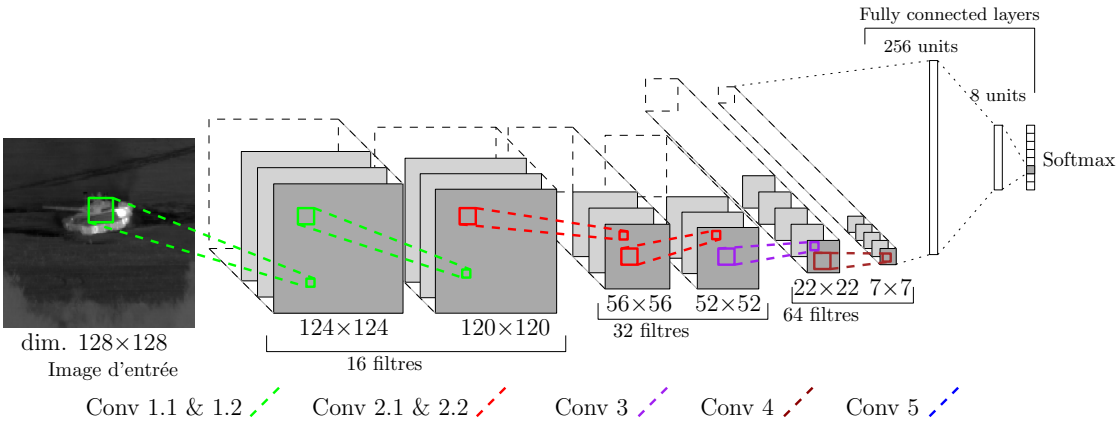


FIGURE 3.11 – Schéma de l'architecture du cfCNN(fc), version *fully-connected* du cfCNN.

L'entrée du STN sera précédée d'une couche de *max-pooling* pour réduire la dimension des entrées à un tenseur de dimension $62 \times 62 \times 16$ et ainsi réduire le nombre de paramètres nécessaires pour le STN. L'architecture complète est présentée dans la figure 3.12.

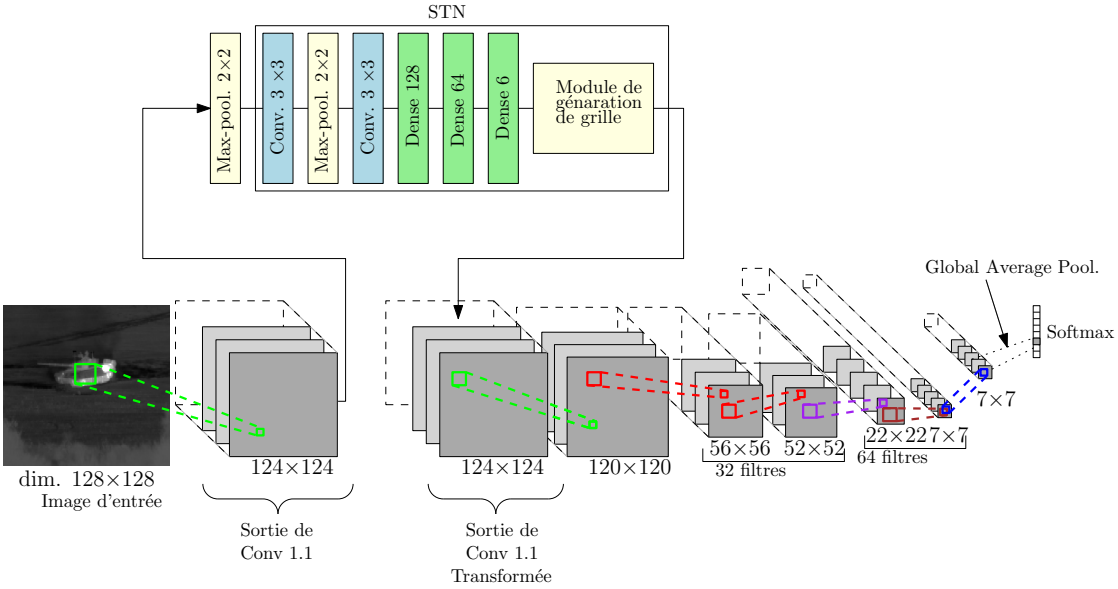


FIGURE 3.12 – Schéma de l'architecture du cfCNN(STN).

Variante avec encapsulation LSE

Pour améliorer la capacité de discrimination de notre cfCNN avec peu d'images, notamment sur les données réelles, nous proposons d'utiliser une approche similaire à celle décrite dans la section 3.1.4. La structure que nous proposons est décrite dans la figure 3.13.

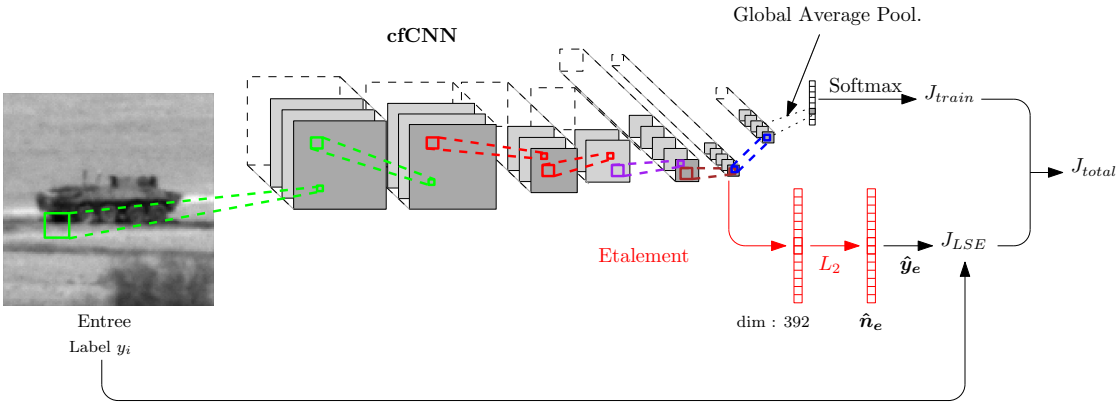


FIGURE 3.13 – Ajout de l'encapsulation au cfCNN.

Cette approche utilise les sorties de la couche de convolution **Conv 5**. Ces dernières sont étalées pour former un vecteur de 392 éléments qui sert d'entrée à une couche de neurones à $\boldsymbol{n_e}$ sorties. La sortie de cette couche est normalisée en utilisant la norme L_2. Ce nouveau vecteur, appelé $\hat{\boldsymbol{y}}_{\boldsymbol{e}}$, sera utilisé pour calculer un terme d'erreur J_{LSE}, qui sera ajouté à l'erreur totale du modèle pendant l'entrainement.

Pour ce nouveau terme nous utiliseront le principe du *Lifted Structured Embedding* [46] ou LSE en lieu et place de l'encapsulation utilisée par Cheng et al. Dans un batch d'images, pour tout couple d'entrée $\{\boldsymbol{x_i}, \boldsymbol{x_j}\}$ nous commencerons par calculer la distance $D_{i,j}$ entre leurs sorties respectives $\hat{\boldsymbol{y}}_{\boldsymbol{e,i}}$ et $\hat{\boldsymbol{y}}_{\boldsymbol{e,j}}$. Puis nous calculerons le terme suivant :

$$\mathcal{J}_{i,j} = \log\left(\sum_{(i,k)\in\mathcal{P}} exp(\alpha - D_{i,k})) + \sum_{(j,l)\in\mathcal{N}} exp(\alpha - D_{j,l})\right) + D_{i,j}. \tag{3.1}$$

avec $\mathcal{P}$ l'ensemble des paires d'images d'entrées appartenant à la même classe, $\mathcal{N}$ l'ensemble des paires d'images d'entrées n'appartenant pas à la même classe. La marge α est choisie empiriquement et permet de pénaliser les distances entre paires qui sont inférieures à ce paramètre.

Enfin, l'équation 3.1 nous permet de calculer le terme d'erreur suivant :

$$J_{LSE} = \frac{1}{2\text{Card}(\mathcal{P})} \sum_{(i,j)\in\mathcal{P}} max(0, \tilde{\mathcal{J}}_{i,j}) \tag{3.2}$$

Et l'erreur totale utilisée pour l'apprentissage sera :

$$J_{total} = J_{train} + J_{LSE} \tag{3.3}$$

Pour nos expériences, après différents essais, nous avons fixé $n_e = 64$ et $\alpha = 0.1$. La mesure de distance utilisée pour $D_{i,j}$ est la distance de Manhattan, qui pour deux vecteur $\boldsymbol{x}$ et $\boldsymbol{y}$ s'écrira

simplement :

$$D_{Manhattan} = |\boldsymbol{x} - \boldsymbol{y}| \tag{3.4}$$

Des détails supplémentaires sur le choix des paramètres pour le LSE et le choix de D sont disponibles dans l'annexe **??**. Nous supposons que l'ajout de cette contrainte supplémentaire aidera le réseau à mieux généraliser sur de nouvelles données. Elle sera un complément aux variations d'éclairement, de position et de bruit présentes dans les images d'entrainement.

3.3.3 Résultats sur SENSIAC

Nous avons commencé par évaluer le cfCNN sur SENSIAC car il s'agit du seul ensemble qui contient des images réelles en bande 2 infrarouge (MWIR) de véhicules militaires comme précisé dans la section 2.3.2. En utilisant uniquement les données prises à courte distance pour 8 véhicules nous obtenons un total de 25600 images que nous divisons en 17920 images d'entrainement, 2560 images de validation et 5120 images de test. Avec cette configuration, nous obtenons un taux de bonne classification supérieur à 95% pour le cfCNN sur l'ensemble des entrainements effectués. Les trois variantes du réseau introduites dans la section 3.3.2 ont obtenus des résultats similaires. Ils sont présentés dans le tableau 3.4. Nous avons notés quelques différences cependant. Le taux de bonne classification du cfCNN(fc) a atteint un minimum de 92.3% et celui du cfCNN avec encapsulation LSE n'est jamais descendu en dessous de 96.7%. Les résultats du cfCNN avec l'activation ReLU étaient très proches de ceux du cfCNN avec LReLU.

Contrairement à des jeux de données comme MNIST ou Imagenet, SENSIAC est relativement moins utilisé. Certaines études dédiées à la classification sont cependant disponibles pour pouvoir comparer les performances initiales de notre cfCNN sur des images infrarouges réalistes. Venkataraman et al. [119] ont par exemple étudié l'utilisation d'une méthode d'interpolation de forme pour la classification des véhicules de la base SENSIAC. Ils ont réparti pour cela les véhicules en 4 catégories pris successivement pour des distances de 1000, 1500 et 2000m. Nous avons essayé de reproduire leur méthode de découpage de la base SENSIAC et nous avons entrainé puis testé le cfCNN, le cfCNN(fc) et le cfCNN avec LSE sur les 4 catégories.

Le tableau 3.4 présente nos résultats par rapport aux performances du modèle de Venkataraman et al. sur les 4 catégories de véhicules.

Les résultats présentés ici montrent de bons résultats pour les trois versions du cfCNN par rapport au SVM avec un léger avantage pour le cfCNN et le cfCNN avec LSE. Cependant, nous considérons que le problème définit par Venkataraman et al. est plus simple que la classification de chaque cible séparément. Il ne permet pas de bien discerner les différences de performances entre le cfCNN et ses variantes.

Classe	Véhicules	[119]	cfCNN	cfCNN(fc)	cfCNN avec LSE
Voitures 1	Pick-up	88.3%	100%	99.2%	100%
Voitures 2	SUV	99.8%	99.5%	100%	100%
Tanks	T-72, ZSU, 2S3	82.2%	99.5%	88.6%	99.7%
APC	BMP-2, BRDM-2, BTR-70	99%	100%	97.5%	99.8%

TABLE 3.4 – Moyenne du taux de classification correcte pour chaque groupe de véhicules selon [119] et comparé au cfCNN.

3.4 Étude du transfert de la simulation vers le réel

Nous évaluons maintenant le cfCNN face à différentes méthodes de classification d'images dans un cas de transfert d'apprentissage direct. Cette étude est comparable à celle présentée dans la section 3.2. Nous comparons ici le cfCNN et ses variantes aux modèles de l'état de l'art et au SVM. Le réseau sera entrainé sur des données de simulation uniquement avant d'être testé sur données réelles.

3.4.1 Résultats du cfCNN

Nous reprenons les tableaux 3.1 et 3.2 y ajoutons les résultats obtenus par le cfCNN et le cfCNN(fc). Le tableau 3.5 présente donc les gains relatifs de chaque modèle sur les données réelles après avoir été entrainé sur bases TRS2 et TRS3, par rapport a un test sur DR après un entrainement sur TRS1. Le tableau 3.6 présente donc les gains relatifs des différents CNN de la littérature et des variantes du cfCNN par rapport au SVM sur les données réelles après avoir été entrainé sur TRS1, TRS2 et TRS3.

Comme pour le tableau 3.1, dans le tableau 3.5, le cfCNN et sa version fully connected bénéficient tous deux de performances améliorées avec l'augmentation de la qualité de la simulation. Les gains du cfCNN(fc) semblent aussi significatifs, avec une amélioration de 23.61% pour l'identification après entrainement sur TRS3 par exemple. Ces valeurs pour ce modèle sont cependant à relativiser étant donné qu'il a obtenu des résultats très faibles sur données réelles après avoir été entrainé sur TRS1.

Dans le tableau 3.6, les gains du cfCNN par rapport aux architectures de l'état de l'art sont mis en évidence. On notera le gain de performance significatif après entrainement sur TRS3 avec une augmentation relative en identification et en reconnaissance par rapport à un SVM-HOG de 12.49% et 5.38% respectivement. Ce gain est le plus élevé de tous les modèles testés ici. Cette progression montre aussi l'intérêt que peut apporter le choix d'une architecture dédiée par rapport à l'utilisation directe d'une architecture issue de l'état de l'art pour notre application.

En comparant les résultats du cfCNN et de sa version "fully-connected", le cfCNN(fc), il est possible de voir l'apport de l'utilisation d'une architecture entièrement convolutive et utilisant le

Données d'entrainement	**TRS2**		**TRS3**	
Architecture	**Ident.**	**Reco.**	**Ident.**	**Reco.**
SVM	3.2%	10.20%	8.8%	16.11%
Rodger et al.	3.34%	8.36%	8.43%	9.27%
VGG19	14,27%	20.68%	23.9%	27.89%
BCNN	3.38%	14%	4.48%	12.64%
Inception v3	0.79%	10.68%	9%	25.77%
ResNet 52	21.51%	26.52%	25.68%	27.49%
cfCNN(STN)	**4.81%**	**9.86%**	**16.01%**	**13.89%**
cfCNN(fc)	**14.17%**	**12.71%**	**23.61%**	**17.71%**
cfCNN	**6.20%**	**10.62%**	**14.67%**	**17.98%**

TABLE 3.5 – Gains de performance relatifs en test pour les scores d'identification et reconnaissance de chaque architecture dont le cfCNN et sa version "fully-connected" sur les images réelles (DR) par rapport à la même architecture de référence entrainée uniquement sur TRS1.

Données d'entrainement	**TRS1**		**TRS2**		**TRS3**	
Architecture	**Ident.**	**Reco.**	**Ident.**	**Reco.**	**Ident.**	**Recog.**
R et al.	1.44%	1.78%	2.96%	−2.47%	1.92%	−9%
VGG19	−14.9%	−16.28%	4.84%	0.49%	0.27%	−4.35%
BCNN	1.31%	−7.13%	2.91%	−2.87%	−4.37%	−5.62%
Inception v3	−2.47%	−15.83%	−9.35%	−10.98%	−1.25%	7.82%
Resnet 52	−10.47%	−12.03%	2.02%	−1.09%	2.92%	0.3%
cfCNN(STN)	**−10.22%**	**−8.38%**	**−7.19%**	**−13.37%**	**−4.73%**	**−10.05%**
cfCNN(fc)	**−16.58%**	**−3.57%**	**4.38%**	**−1.49%**	**10.02%**	**−2.65%**
cfCNN	**12.69%**	**5.09%**	**18.44%**	**3.49%**	**12.49%**	**5.38%**

TABLE 3.6 – Gains de performance en identification et reconnaissance des modèles sélectionnés dont le cfCNN et sa version *fully-connected* sur les images réelles (DR) par rapport à un SVM avec descripteurs HOG entrainé sur les mêmes jeux de données.

GAP pour le transfert d'apprentissage. Ces éléments semblent aider le cfCNN a mieux généraliser sur les données d'entrainement comment le montrent l'écart des résultats dans le tableau 3.6 sur l'ensemble TRS1, le plus simple, entre le cfCNN et le cfCNN(fc).

Nous avons aussi regroupé dans le tableau 3.7 le nombre de paramètres de chaque CNN, la taille de lot choisie et le temps d'entrainement par lot. Ces résultats on été obtenus sur une Nvidia Tesla P4 avec 8 giga-octets de mémoire vidéo.

Modèle	Nb. paramètres	Taille de lot	Délai d'entrainement
R et al.	310×10^3	256	59s
VGG19	143×10^6	128	108s
BCNN	285×10^6	96	251s
Inception v3	24×10^6	128	85s
ResNet 52	23×10^6	128	67s
cfCNN(STN)	8.1×10^5	128	86s
cfCNN(fc)	1×10^6	256	58s
cfCNN	$\mathbf{200 \times 10^3}$	**256**	**55s**

TABLE 3.7 – Nombre de paramètres approximatif et temps d'entrainement par lot des différents modèles de CNN sur un GPU Nvidia Tesla P4.

Nous pouvons voir que le cfCNN est le modèle le plus léger, en terme de nombre de paramètres, et le plus rapide à entrainer sur une itération de Adam. Il y a donc un réel intérêt à privilégier des modèles compacts et dédiés à un problème de classification particulier par rapport à la ré-utilisation de modèles issus de la littérature.

3.4.2 Gains apportés par la fonction LeakyRELU

Pour justifier le choix de la non linéarité LReLU par rapport à la fonction ReLU simple, et compléter les résultats présentés précédemment, nous avons comparé deux versions du cfCNN entrainées à dix reprises sur les images de meilleure qualité, TRS3, puis testés sur DR. La première version est le cfCNN standard avec la fonction LReLU et la deuxième utilise ReLU. En comparant les résultats, nous avons observé que l'utilisation de LReLU permet d'améliorer de score d'identification de 4.75% et le score d'identification de 2.85% en moyenne par rapport au modèle utilisant la fonction ReLU. La fonction LReLU a aussi permis de réduire sensiblement la déviation standard de la distribution des résultats en identification et reconnaissance des différents modèles en passant de 8.24% et 5.23% respectivement pour ReLU à 2.82% et 1.93% respectivement pour LReLU.

Pour observer qualitativement l'impact du changement entre ReLU et LReLU nous avons utilisé l'outil *guided-Grad-CAM*. Il nous permet de visualiser les zones de l'image ayant le plus contribué au résultat de la classification. Le principe de fonctionnement est détaillé en annexe **??**. Compte tenu des limitations de partage de la base de donnée MBDA nous montrerons des résultats visuellement similaires obtenus avec les images de la base de donnée SENSIAC dans la figure 3.14.

En comparant les images des Grad-CAM dans la figures 3.14b et la figure 3.14c obtenues à partir de l'image source 3.14a, nous pouvons voir que le réseau utilisant l'activation LReLU

(a) Image d'origine (b) Guided-Grad-CAM ReLU (c) Guided-Grad-CAM LReLU

FIGURE 3.14 – Visualisation des Guided-Grad-CAM pour le cfCNN avec activations ReLU et LReLU sur une image de BMP2 issue de SENSIAC.

semble baser sa décision de classification sur un nombre de détails plus important que le réseau utilisant la fonction ReLU. Cela est notamment visible au niveau des roues du véhicule ou de sa tourelle qui sont plus visibles dans la. figure 3.14c par rapport à la figure 3.14b. Cette différence peut expliquer le gain de performance observé pour le cfCNN équipé de la fonction d'activation LReLU.

3.4.3 Effets de la réduction du nombre d'images d'entrainement

Pour compléter la comparaison de l'ensemble des modèles choisis dans la section 3.2.1, nous avons ré-entrainé chaque réseau de neurones sur des fractions décroissantes du jeu de données TRS3 puis testé sur DR pour déterminer un seuil critique en dessous duquel la dégradation des performances est trop importante. Les résultats présentés sont les performances relatives de chaque modèle par rapport à ses performances dans le tableau 3.6.

Le tableau 3.8 présente l'évolution des performances de chaque réseau de neurones par rapport au cfCNN entrainé sur l'ensemble du jeu de données simulées. Quelle que soit la fraction choisie, le cfCNN obtient les meilleurs résultats de classification. De plus, il est encore capable de maintenir un taux de classification sur les données réelles en moyenne supérieur à 95% du résultat obtenu en utilisant la totalité de l'ensemble TRS3 avec seulement la moitié des images de TRS3 pour l'entrainement.

Compte tenu des performances du cfCNN après avoir été entrainé sur seulement un quart des images d'entrainement, nous avons encore réduit la taille du jeu de données d'entrainement. La figure 3.15 présente les résultats de classification sur données réelles pour un nombre d'image d'entrainement allant de 4096 à seulement 64.

Les valeurs indiquées dans la figure 3.15 correspondent aux performances relatives de ces nouveaux entrainements du cfCNN par rapport aux cas idéal utilisant la totalité du jeu de

Architecture	Nb. d'images d'entrainement	
	9472	4736
SVM	89.42%	78.66%
R et al.	79.25%	76.19%
VGG19	81.01%	74.97%
BCNN	86.07%	79.05%
Inception	85.82%	71.92%
ResNet	88.67%	80.4%
cfCNN(STN)	71.96%	57.74%
cfCNN(fc)	87.26%	78.06%
cfCNN	97.59%	87.88%

TABLE 3.8 – Performances relatives en identification par rapport aux résultats du cfCNN sur DR après un entrainement sur TRS3.

donnée TRS3. Sur cette figure, nous pouvons voir que les performances pour le test sur le jeu de données réelles DR chutent rapidement. Celles sur les données simulées restent supérieures à 90% des performances originales du cfCNN jusqu'à 1024 images d'entrainement. Nous pouvons aussi voir de légères oscillations sur la courbe 3.15b. Celle-ci peuvent être dues à un nombre de réalisation insuffisant. Il est possible que cette courbe soit plus lisse en répétant l'expérience.

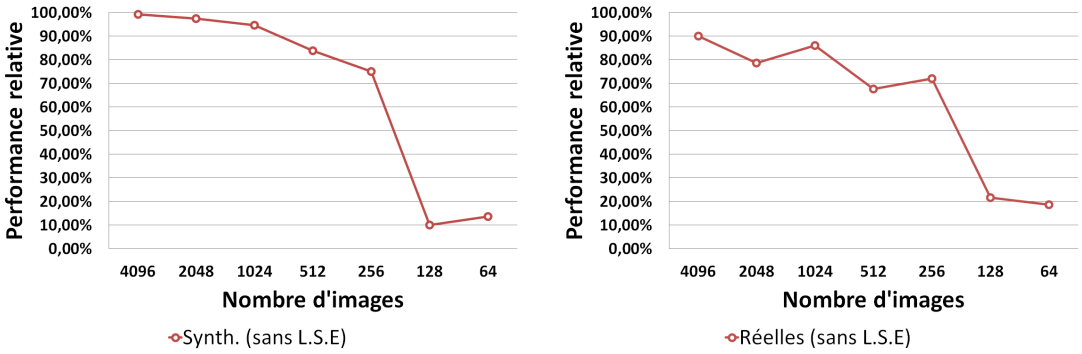
3.4.4 Résultats avec encapsulation

Dans cette section, nous présentons les résultats obtenus par le cfCNN augmenté par le LSE et entrainé sur le jeu de données TRS3. Ces résultats sont comparés à ceux obtenus dans la section 3.4.3.

La figure 3.16 compare les résultats de classification sur données simulées pour un nombre d'image d'entrainement allant de 4096 à 64 avec LSE en bleu comparé aux résultats sans LSE de la figure 3.15 en rouge.

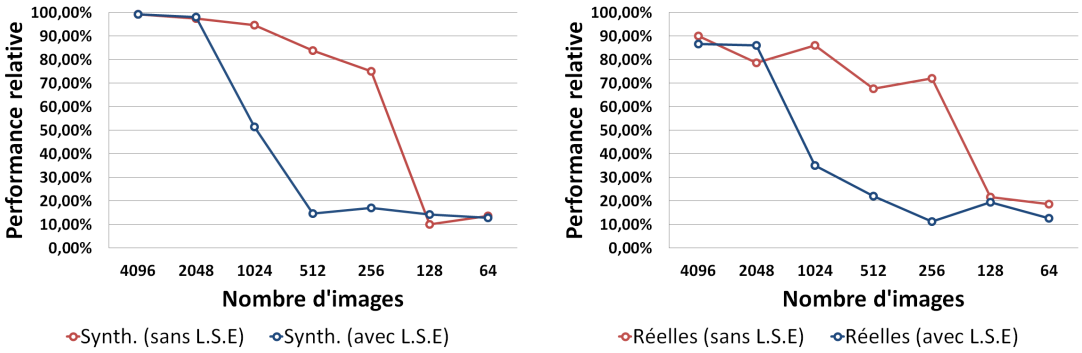
L'utilisation de l'encapsulation n'a permis d'augmenter le score d'identification sur images réelles que pour 2048 images d'entrainement de 7.5% en moyenne. Cependant, pour tout nombre d'images inférieur à 2048, la contrainte supplémentaire imposée par le LSE rend difficile l'apprentissage. Le taux de bonne classification décroit alors fortement, aussi bien pour le test sur des images simulées que sur les images réelles.

De plus, en utilisant la totalité des 18944 images du jeu de données TRS3 pour entrainer le cfCNN nous observons une chute de performance de 1,15% en moyenne sur les images réelles par rapport au cfCNN sans LSE. Nous aurions espéré observer au minimum un léger gain de performances. En effet, nous avions supposé que l'ajout de cette contrainte combiné à la variabilité



(a) Performances sur images synthétiques (b) Performances sur images réelles

FIGURE 3.15 – Détail des variations de performances pour le cfCNN en fonction du nombre d'images d'entrainement utilisées.



(a) Performances sur images synthétiques (b) Performances sur images réelles

FIGURE 3.16 – Détail des variations de performances pour le cfCNN en fonction du nombre d'images d'entrainement utilisées.

des signatures puisse aider le réseau à converger vers un état où il serait moins affecté par les irrégularités et les perturbations présentes dans les images réelles.

L'approche par encapsulation ne permet donc pas d'améliorer les performances du cfCNN, que ce soit pour un entrainement avec un faible nombre de données, ou avec la totalité des images d'entrainement. L'encapsulation n'est peut-être pas adapté pour des cas où il y a une différence trop importante entre les données d'entrainement et les données de test. Ici, l'ajout du LSE pendant l'entrainement à entrainé un sur-apprentissage sur les données simulées.

3.5 Robustesse aux perturbations

La partie 3.4 a illustré l'intérêt d'utiliser notre cfCNN pour la classification d'images dans un cas de transfert de la simulation vers le réel par rapport à un modèle issu de l'état de l'art ou un SVM. Cette classification étant effectuée après un étage de détection, il est possible d'obtenir des erreurs de localisation de la cible et donc des boîtes englobantes mal dimensionnées qui impactent la position des cibles dans les images en entrée du cfCNN. La cible peut donc paraître translatée, à la mauvaise échelle ou partiellement masquée par les bords de l'image. De plus, comme tout système de prise de vue optique utilisant un capteur numérique, les images infrarouges sont susceptibles d'être fortement bruitées lors des prises de vue.

Dans la suite de cette partie, nous nous concentrerons donc sur trois formes de perturbations fréquemment rencontrées : translations, erreurs d'échelle et bruit de mesure. Compte tenu des résultats obtenus avec le LSE présenté précédemment, nous utiliserons ici un cfCNN sans encapsulation et similaire à celui présenté dans la section 3.3.

3.5.1 Construction d'une base d'images perturbées

Pour tester la robustesse des différents modèles présentés en 3.4 trois jeux de données supplémentaires seront utilisés. Nous les appellerons TTS, TSS et TBS. Ils seront utilisés pour simuler les erreurs de détection qui se traduisent par l'introduction d'erreurs de localisation de la boite englobante. À savoir quand les images d'entrées du cfCNN ne sont pas parfaitement centrées sur la cible, les problèmes d'échelle ainsi que la présence de bruit dans l'image.

Préparation de TTS

Le premier jeu, TTS, servira à évaluer la robustesse des modèles de classification lorsque la boite englobante après détection est translatée par rapport à une boite englobante parfaite. Pour le constituer, nous utilisons les images du jeu TV3 dont nous translatons les boites englobantes selon l'axe horizontal et vertical. L'amplitude de la translation correspond à une fraction de la dimension de la boite englobante le long de l'axe sélectionné. Cette nouvelle boite est utilisée pour extraire une nouvelle image de 128×128 en se servant de la boite englobante translatée comme référence. La boite sera translatée selon quatre directions, "HAUT" et "BAS" pour l'axe vertical et "GAUCHE" et "DROITE" pour l'axe horizontal. À partir de TV3 on crée ainsi un ensemble d'images comprenant des exemples de translation dans les quatre directions pour des amplitudes relatives comprises entre 0.1 et 0.5. Ces amplitudes correspondent au ratio entre l'amplitude absolue de la translation et la dimension de la boite d'origine dans la direction du déplacement. La figure 3.17 illustre le processus pour une translation vers le haut d'une amplitude de 0.3. Nous disposons de 4096 images par amplitude de translation.

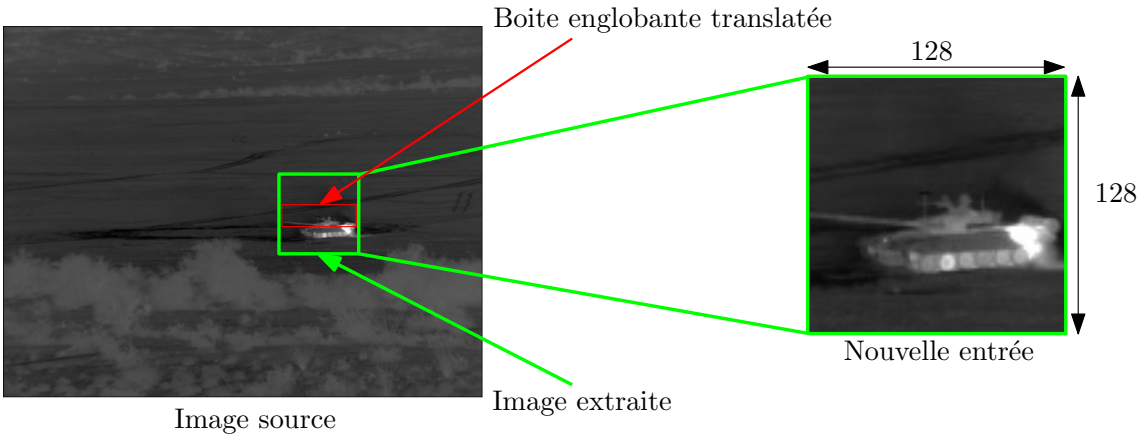


FIGURE 3.17 – Méthode de création des images d'entrée translatée du cfCNN à partir de la source. La boite ici a été décalée d'une amplitude de 0.3 vers le haut pour simuler une erreur de détection.

Préparation de TSS

Le second jeu, TSS, sera lui utilisé pour évaluer la robustesse des modèles face aux erreurs d'échelle *i.e.* lorsque la boite englobante est trop petite ou trop grande par rapport au véhicule cible. Une nouvelle fois, nous utiliserons les images du jeu de données TV3 pour lesquelles les boites englobantes resteront centrées mais dont l'échelle variera d'un facteur compris entre 0.5 et 1.5. Ces nouvelles boites, serviront une nouvelle fois à extraire des patchs de 128×128 pixels. Une illustration de la méthode est présentée dans la figure 3.22 .

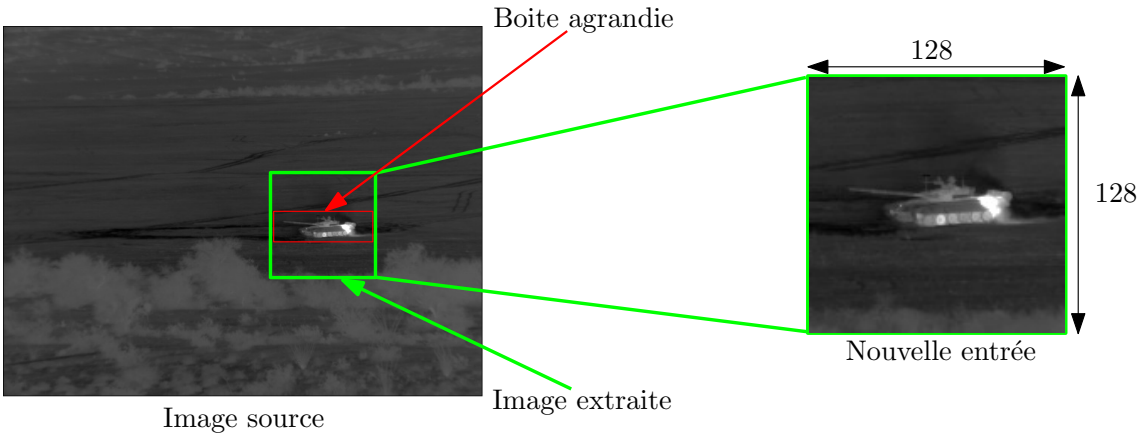


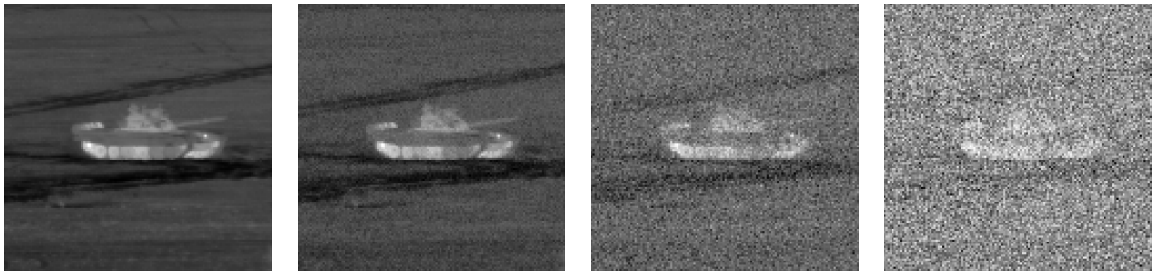
FIGURE 3.18 – Méthode de création des images d'entrée translatée du cfCNN à partir de la source. La boite d'origine ici a été mise à l'échelle d'un facteur 1.4 simuler une erreur de détection.

Comme pour les images centrées des ensembles d'images TRS1, TRS2, TRS3 et DR les images utilisées pour TTS et TSS ont simplement été normalisées entre 0 et 1. Nous disposons de 4096

images par facteur d'échelle.

Préparation de TBS

En complément de ces deux jeux de données nous évaluerons aussi la robustesse des performances de classification en fonction du bruit dans l'image. Pour cela nous utiliserons les architectures testées sur TRS3 puis nous les testerons sur les images de TVS3 normalisées en y ajoutant un bruit gaussien centré sur zéro et dont la déviation standard σ_{bruit} variera entre 0.01 et 0.2 à l'inférence par rapport au domaine $[0, 1]$ des images normalisées. Ces images sont ensuite re-normalisées dans l'intervalle $[0, 1]$. Comme pour TTS et TSS nous disposons de 4096 images par niveau de bruit testé. Des exemples d'images bruitées sont présentées dans la figure 3.19



(a) $\sigma = 0$ (b) $\sigma = 0.02$ (c) $\sigma = 0.08$ (d) $\sigma = 0.2$

FIGURE 3.19 – Exemples d'images bruitées du jeu de données TBS pour différentes valeurs de σ.

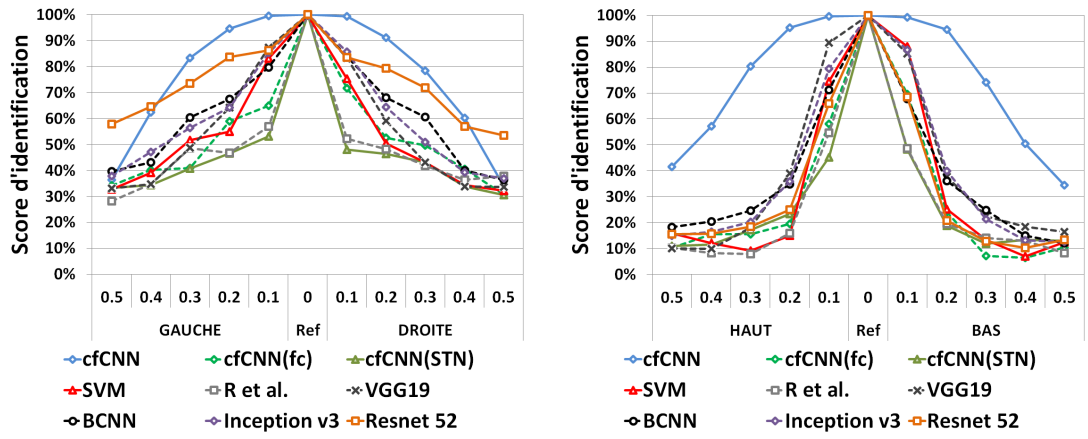
3.5.2 Mise en évidence de la robustesse du cfCNN aux perturbations

Translation de la boîte englobante

Les premiers essais sont effectués sur les boites translatées et les modèles utilisés dans la section 3.2 et 3.3 sans la variante du cfCNN avec encapsulation. Chaque modèle sera entrainé sur le jeu de données TRS3 sans augmentation de donnée supplémentaire. Les boites englobantes de ce jeu de données étant idéales, les cibles seront parfaitement centrées dans les images d'entrainement.

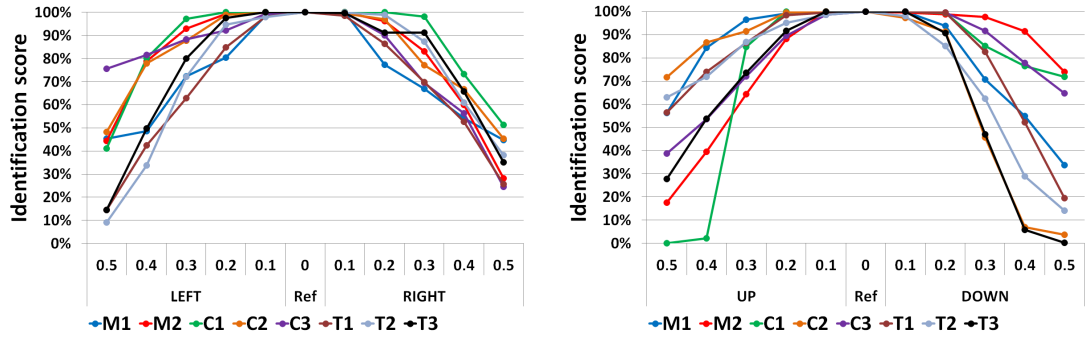
Après avoir été entrainé, chaque modèle est ensuite testé sur l'ensemble d'images TTS pour obtenir les performances de robustesse. Entrainer les architectures sur des données non translatées nous permet de mesurer directement l'impact du décalage de la cible dans l'image sur les performances. De plus les images de TTS étant issues du jeu de données TV3 les résultats ne seront pas impactés par le transfert de domaine. Les résultats obtenus par chaque modèles sont présentés dans la figure 3.20.

Ces deux ensembles courbes montrent que le cfCNN surpasse les modèles de la littérature aussi bien en translation horizontale, figure 3.20a, que verticale, figure 3.20b. Il s'agit de l'unique



(a) Axe horizontal (b) Axe vertical

FIGURE 3.20 – Variation des performances relatives de classification pour des boites englobantes translatées verticalement ou horizontalement.



(a) Axe horizontal (b) Axe vertical

FIGURE 3.21 – Variation des performances de classification pour chaque classe de véhicule pour le cfCNN.

modèle dont les performances en identification se maintiennent au-dessus de 80% pour un décalage de la boite englobante d'une amplitude de 0.3. Parmi les autres modèles, aucun ne maintient des performances au-delà de 80% pour des décalages de la boite englobante d'une amplitude supérieure à 0.1 à l'exception de Resnet 52. Ce dernier maintient un niveau de performance supérieur à 54% pour les translations horizontales de toute amplitude. Il est cependant surpassé par le cfCNN pour les translations horizontales d'une amplitude inférieure à 0.3. Le BCNN ainsi que Inception v3 surpassent légèrement le SVM. Cela montre que sans augmentation de données pour ce type de perturbation, la robustesse d'un CNN n'est pas acquise. En ce qui concerne le SVM, la robustesse des descripteurs HOG dépends de la résolution de la grille utilisée pour les calculer. Si l'amplitude de la translation est supérieure à cette résolution, le descripteur utilisé change. Les descripteurs HOG sont surtout robuste vis à vis des rotations et non aux translations, ce qui peut expliquer en partie les faibles résultats.

En comparant les résultats du cfCNN avec ceux du cfCNN(fc) nous pouvons voir que l'ajout de la couche de GAP permet bien d'améliorer la robustesse du modèle pour des petites perturbations comme suggéré dans la section 3.1.5. Cependant, pour le cfCNN(STN), l'ajout du STN au cfCNN n'a pas permis d'améliorer la robustesse du réseau. Cela est probablement du au choix d'entrainer les différents réseaux choisis sur des images non-perturbées. Ainsi, le STN n'a pas été en mesure d'apprendre une façon de transformer les sorties de la couche Conv 1.1 du cfCNN pour compenser les erreurs de détection. Pour mieux visualiser l'apport du cfCNN(STN) il faudrait ajouter des exemples d'images perturbés dans la base d'entrainement.

La symétrie des courbes de la figure 3.20 masque le fait que chaque cible se comporte différemment en fonction de la translation. Les courbes par cibles pour le cfCNN dans la figure 3.21 illustre en partie ce phénomène. En détaillant les résultats véhicule par véhicule. La dissymétrie de la réponse du modèle pour chaque cible est bien visible, notamment dans la figure 3.21b entre les courbes de la cible M1 et C1 par exemple. Cela peut s'expliquer par le fait que le décalage de la boite englobante peut masquer partiellement la cible et notamment des points chauds parfois essentiels à l'identification du véhicule.

Mise à l'échelle de la boite englobante

Après avoir évalué la réponse du réseau sur la translation des cibles. Nous allons étudier le cas où l'échelle de la boite est faussée. Les résultats de l'étude sont présentés dans la figure 3.22.

Comme pour une translation de la boîte englobante, le cfCNN montre de très bonnes performances malgré l'ajout de la perturbation. Il dépasse tous les autres modèles de réseaux de neurones testés ici. Dans le cas où le facteur d'échelle est inférieur à 1, il maintient de bonne performances malgré la présence potentielle de masquages du véhicule et de certains points chauds. Encore une fois, le cfCNN(STN) n'est pas en mesure de compenser les variation d'échelles si ce dernier n'a pas été entrainé au préalable sur des données qui contiennent ce type de perturbations.

Cependant, pour les cas où la boite est plus grande que la cible d'origine, *i.e.* pour un facteur d'échelle supérieur à 1, le SVM obtient de meilleurs performances que le cfCNN. Cela peut

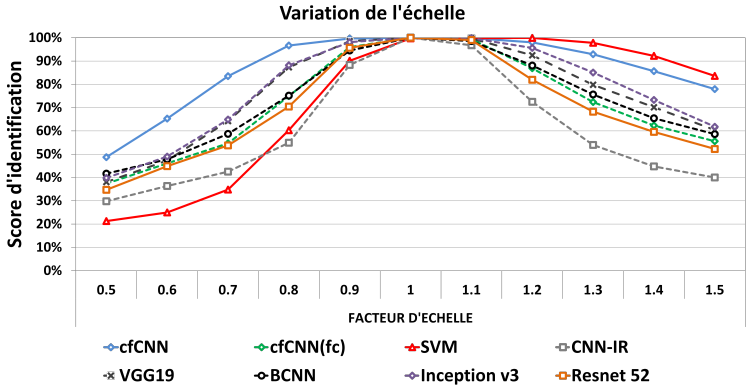


FIGURE 3.22 – Résultats d'identification après modification de l'échelle de la boite englobante.

suggérer une meilleure robustesse du SVM face aux petites cibles.

Robustesse de la classification face au bruit

Comme dernier test de robustesse nous évaluons ici la réponse des modèles sélectionnés face au bruit présent dans les images normalisées. Dans la figure 3.23 nous présentons l'évolution du score d'identification en fonction de la déviation standard du bruit σ_{bruit}. Il faut aussi noter que les images simulées de l'ensemble TRS3 contiennent des exemples avec des niveaux de bruits de mesure variées.

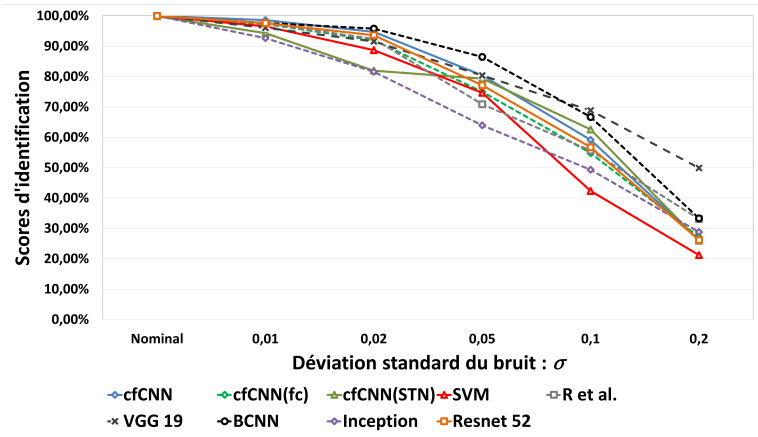


FIGURE 3.23 – Évolution des performances relatives en fonction de la déviation standard du niveau de bruit dans les images d'entrées.

Parmi les architectures testées, le BCNN est le plus robuste pour un σ_{bruit} inférieur à 0.1. En second vient le cfCNN, qui obtient des taux de classification similaires au BCNN et supérieurs à 94% pour des valeurs de σ_{bruit} inférieures à 0.02. Pour un σ_{bruit} supérieur à 0.1 cependant, ses

performances chutent fortement avec seulement 26% des cibles correctement identifiées. Pour ces valeurs, VGG 19 obtient les meilleures performances d'identification.

Les bonnes performances du BCNN et de VGG 19 sont cependant à relativiser compte tenu de ses résultats lors du transfert d'apprentissage présenté dans le tableau 3.6.

3.5.3 Conclusions de l'étude de robustesse

Les sections 3.5.2, 3.5.2 et 3.5.2 ont illustré les performances du cfCNN sur des perturbations régulièrement rencontrées par des systèmes de reconnaissance et d'identification. Ces perturbation comprenaient des translations et des changements d'échelle des images d'entrées pour simuler des erreurs de détection, ainsi que des exemples bruités.

Les résultats du cfCNN ont été comparés à ceux d'un SVM utilisant descripteurs HOG qui sont relativement robustes aux translation et d'une sélection d'architectures issues de la littérature. Par rapport à l'ensemble de ces modèles, le cfCNN est le plus robuste face aux translations de la cible dans l'image d'entrée.

Cependant, il n'est pas le plus performant les problèmes de changement d'échelle de la boite englobante et en cas de présence de bruit dans l'image. Néanmoins, ils se place au minimum en deuxième position. L'écart minimum observé par rapport à la meilleure solution était systématiquement inférieure à 5%.

Aussi l'ajout d'un STN au cfCNN n'a pas permis d'améliorer la robustesse du réseaux aux translations et aux changement d'échelles. Comme nous avons fait le choix d'entrainer les modèles sélectionnées sans augmentation de données, le STN n'a pas été en mesure d'apprendre les transformations adéquates pour compenser les perturbations introduites par un étage de détection.

3.6 Conclusion du chapitre

Notre objectif était d'identifier des véhicules dans des images infrarouges à l'aide de réseaux de neurones. La particularité de notre approche réside dans les bases de données utilisées et les limites imposées par le contexte militaire.

Compte tenu des vulnérabilités que peuvent introduire les méthodes de *fine-tuning*, nous avons choisi d'entrainer de zéro un modèle sur des données simulées avant de le tester sur des images réelles. Après avoir montrés les limites des architectures issues de l'état de l'art pour le problème du transfert d'une architecture de réseau de neurones depuis des données simulées vers des données réelles, nous avons présenté une architecture dédiée, le cfCNN.

Ce réseau de neurones a obtenu les meilleures performances parmi tous les modèles que nous avons sélectionnés pour le transfert de la simulation vers le réel. Il a aussi notamment surpassé les performances d'un SVM avec descripteur HOG, qui avait obtenu de bons résultats de classification sur les bases de données que nous avons utilisé. De plus, nous avons aussi montré

que le cfCNN est, grâce à sa structure, robuste à un ensemble de perturbations que nous sommes susceptibles de rencontrer dans un contexte opérationnel.

Chapitre 4

Détection des anomalies de classification

Contents

Ce chapitre concernera le problème de la détection d'anomalies de classification pour les réseaux de neurones et plus particulièrement les CNN. La section 4.1 présentera le contexte ainsi que certaines méthodes pour la détection d'anomalies en général et avec des réseaux de neurones. Puis, dans la section 4.2 nous présenterons notre principale contribution : un détecteur d'anomalies pour les réseaux de neurones basé sur l'algorithme du Local Outlier Factor. Ce détecteur nécessitant des données d'entrainement, nous l'évaluerons dans la section 4.3 dans un contexte ou les données d'entrainement et de test appartiennent au même domaine i.e. de l'infrarouge réel à l'infrarouge réel et de l'infrarouge simulé à l'infrarouge simulé. Enfin, dans la section 4.4 nous testerons ce détecteur sur des données réelles après l'avoir entrainé sur des données simulées. Nous présenterons aussi dans cette section une version de notre détecteur adaptée à ce changement de domaine.

4.1 Contexte et méthodes existantes

4.1.1 Classification erronée avec une confiance élevée

Malgré leurs bonnes performances sur les tâches de classification, les réseaux de neurones sont susceptibles de produire des sorties erronées. Pour les CNN, ces erreurs sont notamment plus fréquentes quand les images d'entrée présentent des perturbations comme nous avons pu les voir dans la section 3.5. Ces erreurs de classification peuvent aussi être liées à des perturbations non-étudiées ici comme la qualité de la compression pour des images, comme le niveau de compression JPEG, ou le niveau de contraste global de l'image [103].

Il arrive que, pour ces sorties erronées, les sorties de la couche Softmax montrent une forme d'hésitation du réseau de neurone entre une ou plusieurs classes. On observe alors pour la classe privilégiée une valeur de score Softmax au mieux légèrement supérieure à 0,5. Dans ces cas de figure, un simple seuillage des valeurs en sortie de la couche Softmax permettrait alors de filtrer ces erreurs de classification [51].

Cependant, les réseaux de neurones sont capables de produire des sorties erronées avec un taux de confiance élevé sur des entrées perturbées même si celles-ci sont visuellement très différentes de la classe prédite par le réseau [2]. Ce comportement a été aussi observé pour la classification d'images infrarouges par Rodgers et al. [101]. De plus, nous avons évoqués dans la section 1.5.3 qu'il était possible de créer des attaques adverses pouvant provoquer des réponses erronées avec un score de Softmax élevé. Une des caractéristiques de ces attaques est qu'elles sont construites de façon à être les plus discrètes possibles, et donc le plus souvent indétectables à l'œil nu.

Ce risque est aussi confirmé par l'existence d'erreurs de classification sur les images perturbées mais aussi des images réelles de la partie précédente, dans laquelle la cible est à priori entièrement

visible. Nous présentons dans la figure 4.1 un histogramme du maximum du score Softmax obtenu pour les erreurs d'identification d'un cfCNN sur les données de l'ensemble DR. Comme nous pouvons le voir, il existe une très forte proportion de cas d'erreurs ou la prédiction du réseau était proche de 1. Le cfCNN est donc susceptible de se tromper avec une confiance élevée.

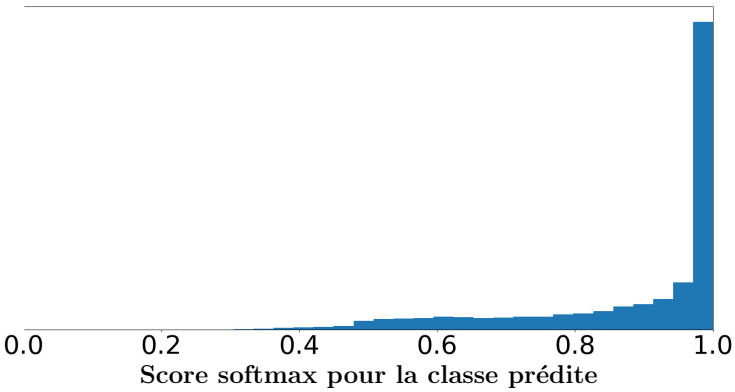


FIGURE 4.1 – Histogramme des scores Softmax obtenus pour les prédictions $\hat{\boldsymbol{y}}$ d'un cfCNN dans les cas d'erreurs de classifications sur le jeu de données DR.

De plus, nous avons pu voir dans la section 3.5 que la présence de bruit ou les images avec des cibles non centrées pouvaient conduire à des erreurs de classification. En effet, dans cette partie, nous nous sommes placé dans un contexte ou nous nous intéressons uniquement à l'identification. Nous sommes donc tributaire des performances du module de détection tel que présenté dans la figure 3.1. Ce dernier peut parfois présenter au CNN des zones du fond de l'image, sans cibles. Aussi, pendant son utilisation, le système peut avoir à gérer la présence de cibles inconnues dans les images. Si ces cibles sont prise en compte par le module de détection, le CNN pour l'identification et la reconnaissance se verra présenter des images de classes inconnues. Avoir la possibilité de détecter ces deux formes d'anomalies, i.e. les nouvelles classes et les fonds, permettrait d'améliorer la fiabilité d'un système de DRI complet.

Pour limiter l'impact de ces perturbations sur les performances de classification, il est d'abord possible de modifier les architectures de réseaux de neurones pour améliorer leur robustesse. Nous présentons dans la section suivantes certaines de ces techniques.

4.1.2 Robustification des architectures

Nous avons évoqué dans la section 3.1.5 différentes approches pour améliorer la robustesse d'un réseau de neurones aux translations et aux changements d'échelle de la cible dans l'image d'entrée. Nous avons notamment inclus dans l'architecture du cfCNN une couche de *Global Average Pooling* qui a sensiblement amélioré la robustesse du modèle aux exemples translatés. Cependant, nous n'avons pas évoqué dans la section 3.1.5 le cas des exemples adverses. Nous avons

simplement évoqués leur existence dans le chapitre 1.5.3 sans présenter les stratégies existante pour se protéger contre ces attaques.

Comme pour les autres formes de perturbations, il est possible de modifier certaines couches pour accroitre la robustesse d'un modèle face aux exemples adverses. Une première forme simple de modification porte sur l'introduction de couches ajoutant un bruit aléatoire à l'entrée du réseau pour atténuer l'effet perturbateur des exemples adverses [75] [80].

L'amélioration de la robustesse peut aussi être abordée dès l'entrainement du modèle que l'on cherche à consolider. L'utilisation du *Stability Training* [105] permet d'introduire à l'entrainement une forme d'augmentation de données, vue dans la section1.4.5, guidée. En procédant de la sorte on est en mesure de renforcer le réseau face aux exemples perturbés en ciblant directement ses faiblesses.

Sur le même sujet, Bastani et al. [10] proposent un ensemble de métriques pour évaluer la robustesse d'un modèle a des exemples adverses. Ces métriques peuvent être alors utilisées pour diriger le processus d'apprentissage et améliorer la robustesse des exemples adverses. Ils suggèrent également d'ajouter à la base d'entrainement des exemples adverses comme nous le ferions pour de l'augmentation de données. Ce procédé, qualifié d'*Adversarial Training* est une des stratégies couramment employée comme technique de défense contre les exemples adverses[43] [72] [68]. Elle n'est cependant pas suffisante pour protéger totallement un modèle [118] [107]. En effet, malgré l'utilisation de l'*Adversarial Training* pour renforcer la réponse d'un modèle face à une famille d'attaques adverses, un réseau de neurone peut rester vulnérable à de nouveaux exemples adverses générés par d'autres stratégies d'attaques.

Toujours concernant les exemples adverses, d'autres méthodes de défense ont été proposées comme la distillation [97] ou le masquage des gradients [99]. Cependant ces deux méthodes présenteraient toujours des vulnérabilitées exploitables pour la génération d'exemples adverses [19][6].

Malgré l'adoption de stratégies de défense contre les perturbations et les exemples adverses, un réseau de neurones peut donc toujours générer des erreurs de classification, comme toute autre méthode de classification. Il semble donc intéressant d'essayer de compléter ou remplacer les techniques présentées dans cette section par un système capable de détecter les erreurs de classification des réseaux de neurones. Nous proposons d'étudier l'utilisation de méthodes de détection d'anomalies pour identifier les cas où la réponse d'un modèle n'est pas fiable.

4.1.3 Approches existantes pour la détection d'anomalies

Contexte général

Le problème de la détection d'anomalie est abordé dans de nombreux domaines, de la sécurité informatique à l'analyse de données médicales. Cette variété de sujets et de données a favorisé le développement de techniques de détection d'anomalies. Les solutions existantes peuvent faire usage d'algorithmes de regroupement ou de densité comme l'algorithme des k-plus proches voisins

[39], DBSCAN [84], les *Isolation Forests* [63], ou le *Local Outlier Factor* [15] et ses variantes.

Il est aussi possible d'utiliser des méthodes d'apprentissage pour la détection d'anomalies. Parmi les méthodes populaires, les machines à vecteurs de support à une classe ont montré de bonnes performances [106] [70]. Plus récemment, des réseaux de neurones particuliers, les auto-encodeurs [42], ont été utilisés avec succès pour la détection d'anomalies [127].

Il existe donc une large variété de solutions pour détecter des anomalies dans un jeu de données. Le choix de la solution à employer dépend essentiellement de la nature des données à traiter. Dans la section suivante, nous nous concentrerons sur les méthodes de détection d'anomalies construites spécifiquement pour les réseaux de neurones.

Cas des réseaux de neurones

Pour la détection des erreurs de classification d'un réseau de neurones, il existe plusieurs approches dans la littérature. Une première façon simple de détecter les anomalies exploite les sorties de la couche Softmax des réseaux de classification. La détection des anomalies peut être effectuée simplement via l'utilisation de mesures de distances entre les sorties probabilistes du réseau et une valeur de référence calculée à partir des données d'apprentissage. Cette approche, proposée par Mandelbaum et al. [89], permet de créer un score de confiance pour la prédiction, basé sur cette distance, qui peut ensuite servir à isoler les anomalies avec un simple seuillage. Pour améliorer les performances de détection, , Ils utilisent une fonction Softmax modifiée pour améliorer la détection des anomalies. Il s'agit d'utiliser un terme de "température" pour abaisser les scores élevés et relever les scores faibles dans le vecteur en sortie de la couche Softmax.

Toujours pour détecter des erreurs de classification, Hendrycks et al. [50] ont choisi d'utiliser un "module d'anomalie" qui est entrainé conjointement avec le réseau de neurones qu'il vient compléter. Ce module d'anomalie est un petit perceptron multi-couches. Il sera chargé de faire la distinction entre les exemples réguliers et les anomalies causées par des perturbations.

Nous avons évoqué dans la section 4.1.1 la possibilité d'observer des erreurs de classification avec des scores de Softmax élevés pour des exemples perturbés. La fonction Softmax écrase la valeur des scores qui ne correspondent pas à la classe dominante. Pour le problème de la détection d'anomalies, ces scores secondaires peuvent cependant aider à discerner les exemples entre eux, et faciliter l'isolation des exemples anormaux.

Une méthode de détection d'anomalie appelée ODIN, pour *Out-of-DIstribution detector for Neural networks* [78] utilise aussi un score Softmax modifié avec un terme de température. Comme pour Mandelbaum et al., ce nouveau score Softmax est comparé a un seuil empirique pour déterminer le caractère anormal de l'exemple. Plutôt que d'utiliser ou modifier la couche sofmax du réseau, Lee et al. [76] utilisent les sorties de la couche qui la précède. En comparant les différentes sorties à l'aide de la distance de Mahalanobis, leur méthode surpasse ODIN.

Le principal défaut de ODIN et de la méthode proposée par Lee et al. vient du fait qu'ils utilisent une forme de pré-traitement sur l'entrée pour améliorer la détection. Le principe de ce pré-traitement est décrit dans la figure 4.2. Celui-ci consiste à ajouter une perturbation,

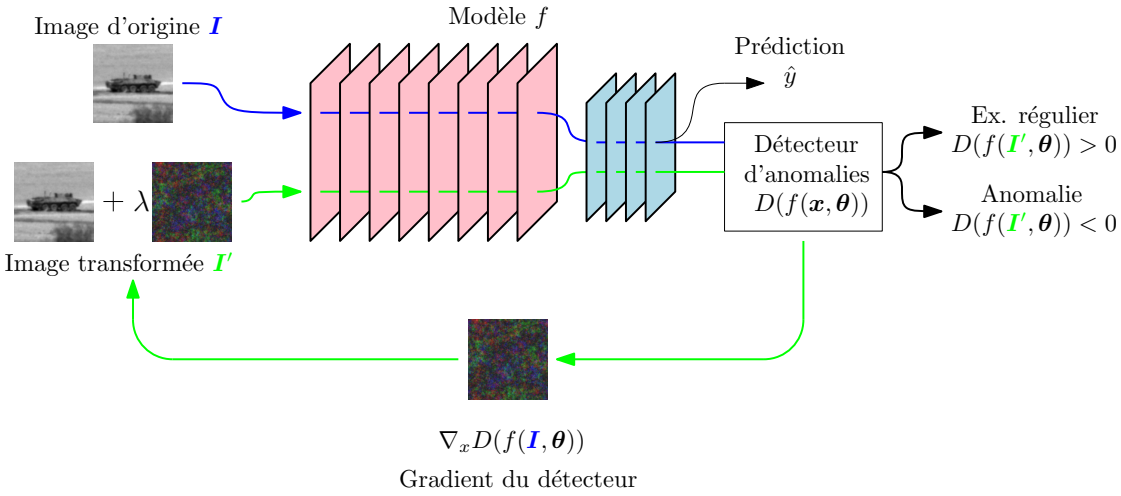


FIGURE 4.2 – Principe du pré-traitement de l'entrée tel qu'utilisé dans [78] et [76].

ressemblant à une image de bruit, calculée à partit du gradient du détecteur $\nabla_x D(f(\boldsymbol{I}, \boldsymbol{\theta}))$ pour améliorer la détection des anomalies. En effet, on suppose que pour un exemple anormal, la réponse du réseau est instable. L'ajout de cette perturbation permet d'écarter cette anomalie de la distribution des exemples réguliers. Il faut donc passer l'entrée, ici l'image $\boldsymbol{I}$, une première fois dans le réseau pour avoir accès aux gradients du modèle, puis une deuxième fois après l'ajout de la perturbation pour déterminer si $\boldsymbol{I}$ est anormale. Le coefficient λ permet de moduler l'amplitude de la perturbation ajoutée. L'image perturbée sera ensuite utilisée pour calculer la valeur $D(f(\boldsymbol{I}', \boldsymbol{\theta}))$, et la comparer avec un seuil empirique.

L'ensemble des techniques décrites précédemment peuvent être utilisées sur des modèles de réseaux de neurones déjà entrainés. Il est aussi possible d'utiliser un score de confiance appris conjointement avec la classe des exemples d'entrainement pour identifier les anomalies [33] [25]. Corbière et al. [25] proposent par exemple d'augmenter un réseau en lui ajoutant un réseau appelé *ConfidNet* qui prends en entrée les sortie d'une couche cachée profonde du réseau augmenté. Ce réseau sera entrainé à prédire ce qu'ils appellent la *True-Class-Probability*, ou TCP. La TCP, pour un couple $\{x^{(t)}, y^{(t)} = c\}$, correspond simplement à la sortie probabiliste $p(c|x^{(t)})$. La motivation de cette approche est basée sur l'observation que, en cas d'erreur de classification, *i.e.* quand $\hat{\boldsymbol{y}} \neq c$, la valeur de la TCP sera faible alors que $p(\hat{\boldsymbol{y}}|x^{(t)})$ sera élevé.

Parmi les méthodes présentées ici, les plus performantes utilisent une technique de pré-traitement similaire à celle de la figure 4.2. Avec le cfCNN, nous avons proposé un CNN compact et léger, ce qui facilite sa préparation et son déploiement sur différentes matériels, *i.e.* CPU, GPU ou *System-on-chip*. Nous souhaitons rester dans cette démarche pour la détection d'anomalies et donc mettre en place un système de détection d'anomalies pour le cfCNN qui ne nécessite pas de pré-traitement et utilise un nombre limité d'hyper-paramètre tout en garantissant un niveau de performances proches de celui d'une méthode comme ODIN. Nous proposons pour cela une

nouvelle méthode de détection d'anomalies de classification. Plus simple, cette méthode utilisera les sorties de l'avant dernière couche du réseau pour séparer les anomalies des exemples réguliers et sans pré-traitement.

4.2 Proposition de méthodes de détection d'anomalies

4.2.1 Approche générale

Notre approche part du principe que nous disposons uniquement d'un réseau de neurones. Nous considérons aussi que nous pouvons l'instrumenter à différents endroits de son architecture. Pour simplifier la visualisation dans un premier temps, nous considérons un réseau simple composé de couches entièrement connectées dédié à la classification d'objet dans c classes distinctes. La dernière couche du réseau sera une couche Sofmax avec c sorties.

Parmi les entrées existantes pour ce réseau, nous désignerons par $\mathbf{T}$ l'ensemble des éléments utilisés pour l'entrainement, et par $\mathbf{V}$ l'ensemble des éléments utilisés pour valider l'apprentissage. Le réseau est supposé avoir été pré-entrainé sur $\mathbf{T}$, et ses poids figés.

Pendant l'inférence, il est possible que ce modèle génère des prédictions erronées, des anomalies de classification, avec une probabilité a postériori qui peut être très élevée. Ces erreurs sont causées par les phénomènes que nous avons évoqués dans la section 4.1.1

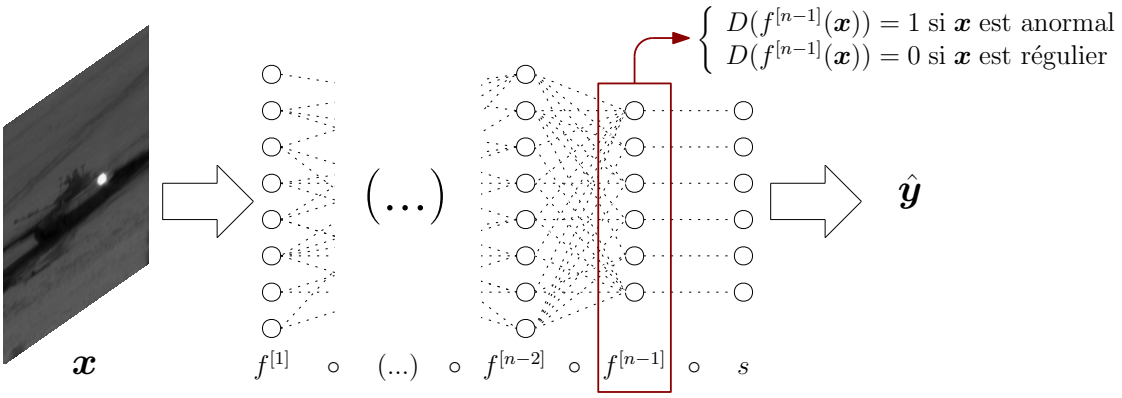
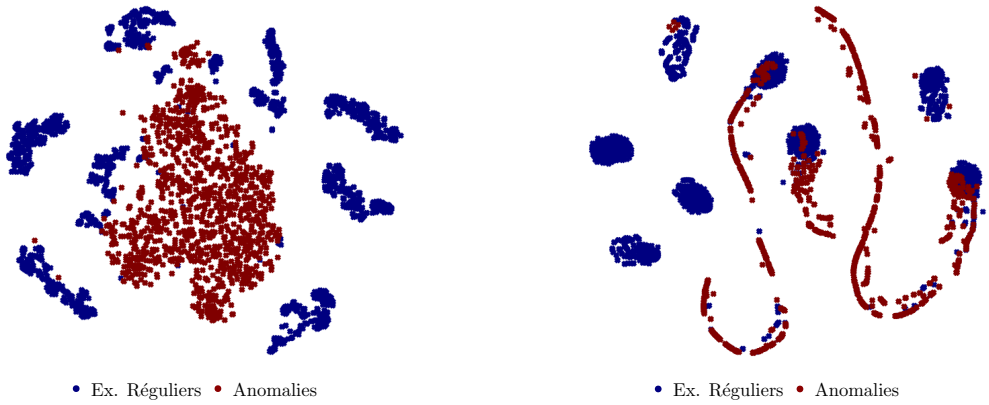


FIGURE 4.3 – Description simple du processus d'identification d'exemples anormaux.

Dans un premier temps, nous instrumentons le réseau à l'avant dernière couche, soit juste avant la couche Softmax. À cet endroit se trouve une couche entièrement connectée, $f^{[n-1]}$, qui possède c sorties comme indiqué dans la figure 4.3. Nous expérimenterons aussi avec les sorties des autres couches intermédiaires du réseau. Ces sorties serviront d'entrée pour une fonction D, qui pour une entrée x sera chargée de déterminer à partir de $f^{[n-1]}(x)$ si x est une anomalie, ou "outlier", ou non.

Nous supposons que les différences entre des exemples réguliers et des anomalies seront plus visibles dans les sorties de la couche $f^{[n-1]}$, que dans les sorties de la couche Softmax. Les

observations de la répartition entre des exemples réguliers et des anomalies dans les sorties de la couches Softmax et $f^{[n-1]}$ via l'algorithme TSNE, pour *T-Distributed Stochastic Neighbor Embedding* [88], nous ont confortés dans notre choix. Un exemple de visualisation de ces sorties est illustré dans la figure 4.4. Ces représentations ont été obtenues à l'aide d'un cfCNN pré-entrainé sur TRS3. Nous avons ensuite utilisé les sorties correspondant aux exemples issus de TVS3 pour les exemples réguliers et un ensemble de fonds et d'images de classes inconnues pour générer les sorties correspondant aux anomalies.



(a) Sorties de la couche $f^{[n-1]}$

(b) Sorties de la couche Softmax

FIGURE 4.4 – Représentation visuelle du changement de distribution des sorties la couche $f^{[n-1]}$ et de la couche Softmax d'un cfCNN pour des exemples réguliers et des anomalies.

En comparant les distributions des sorties de la couche $f^{[n-1]}$ dans la figure 4.4a avec celles de la couche Softmax dans la figure 4.4b, nous pouvons voir que les anomalies, en rouge, semblent être plus facilement regroupées ensembles dans la figure 4.4a. A l'inverse, pour les sorties Softmax, nous pouvons observer à trois endroits différents un mélange entre exemples réguliers et anomalies. Dans les deux cas, nous pouvons aisément observer huit regroupements de marqueurs bleus qui correspondent aux huit classes de véhicules présents dans TVS3.

Dans la mesure où cette fonction doit venir faciliter l'interprétation des prédictions du réseau, elle doit dépendre de peu d'hyper-paramètres. Avoir la possibilité d'utiliser cette fonction sans devoir ré-entrainer le réseau de neurones serait aussi un avantage. Enfin nous souhaitons que cette fonction ne nécessite pas de pré-traiter l'entrée comme c'est le cas pour ODIN par exemple.

En suivant ces critères, deux algorithmes sont proposés : un basé sur le Local Outlier Factor [15] et un autre basé sur un SVM à une classe ou *1-class SVM*.

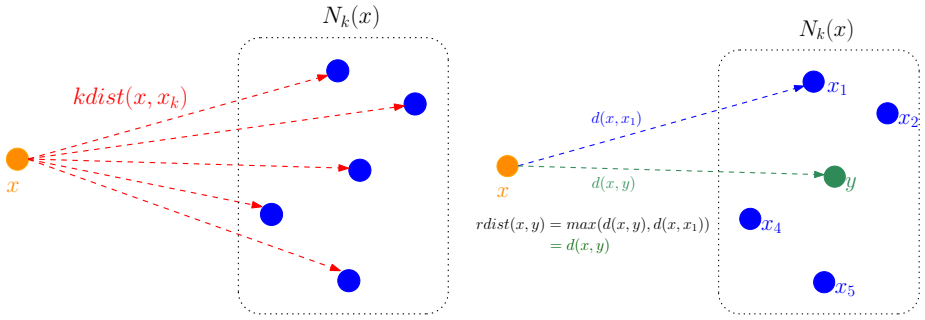
4.2.2 Détection avec le Local Outlier Factor (LOF)

Fonctionnement de l'algorithme

Le Local Outlier Factor, ou LOF, est un algorithme de détection d'anomalies proposé par Breunig et al. [15]. Pour déterminer si une donnée est anormale, sa distance par rapport à ses plus proches voisins est comparée à la distance de ces derniers entre eux. Le LOF exploite donc la densité locale dans l'espace des données à tester pour détecter les anomalies.

Le principe de l'algorithme pour calculer le LOF d'un point x à partir de son voisinage $N_k(x)$ peut être résumé en quatre étapes :

- Dans un premier temps on calcule $d(x, x_k)$, illustré dans la figure 4.5a, qui correspond à la distance entre x et son $k$ième plus proche voisin x_k. La métrique pour le calcul de d est choisie en fonction du problème. Elle peut tout à fait être sélectionnée par les métrique classiques comme que la distance Euclidienne.
- Ensuite, pour chaque plus proche voisin x_l de x, on peut calculer pour tous les k plus proches voisins restants $rdist_k = max(d(x, x_k), d(x_k, x_l))$, présenté dans la figure 4.5b.



(a) k-distance (b) r-distance

FIGURE 4.5 – Illustrations de *kdist* et *rdist* utilisées dans le calcul du LOF.

- A partir *rdist*, on est en mesure de calculer la *local reachability distance*, ou *lrd*, de x :

$$lrd_k(x) = \frac{\text{Card}(N_k(x))}{\sum_{y \in N_k(x)} rdist_k(x, y)}. \tag{4.1}$$

  Cette grandeur représente pour un point la densité locale de ses plus proches voisins dans l'espace.
- Le LOF de x avec ses plus proches voisins est alors défini par :

$$LOF_k(x) = \frac{1}{\text{Card}(N_k(x))} \sum_{y \in N_k(x)} \frac{lrd_k(y)}{lrd_k(x)}. \tag{4.2}$$

Le résultat de l'équation 4.2 permet l'identification des anomalies. En effet, en comparant la

densité d'un point x avec ses plus proches voisins via le ratio $lrd_k(y)/lrd_k(x)$, on détermine si celui-ci est régulier ou non. Une autre conséquence de ce ratio, calculé sur l'ensemble des plus proches voisins de x, est que pour un exemple régulier $LOF(x) \in]0,1]$ et que pour une anomalie $LOF(x) \in]1,+\infty[$. En pratique on introduit une marge ϵ pour adapter la réponse du LOF aux données tel que : $LOF(x_{regulier}) \in]0,1+\epsilon]$ et $LOF(x_{anomalie}) \in]1+\epsilon,+\infty[$. Le paramètre ϵ sera choisi empiriquement.

Utilisation du LOF en complément d'un réseau de neurones

En reprenant la situation décrite dans la partie 4.2.1 nous supposons avoir accès aux sorties du modèle, et plus particulièrement : $\mathbf{Y_T} = f^{[n-1]}(\mathbf{T})$ et $\mathbf{Y_V} = f^{[n-1]}(\mathbf{V})$. Cela correspond à la sortie de la couche précédant l'opération Softmax en sortie du modèle pour les données d'entrainement et de validation. Pour notre étude, sachant que nous fournirons les données anormales, les anomalies seront regroupées dans un ensemble $\mathbf{O}$. Nous aurons donc aussi accès aux sorties $\mathbf{Y_O} = f^{[n-1]}(\mathbf{O})$. Ces valeurs serviront d'entrée pour le LOF qui sera utilisé dans deux configurations qui seront décrites dans ce qui suit : une approche globale et une approche par classe.

Le caractère local du LOF devrait normalement entrainer des résultats similaires pour l'approche globale ou par classe dans le cas où on utilise la distance euclidienne. Cette dernière n'est pas la plus adaptée, les sorties pouvant être de grande dimension selon les réseaux testés. En effet, les distances basées sur des p-normes ou $p > 2$ perdent en contraste sur des données de grande dimension [3].

La distance que nous utiliserons sera la distance de Mahalanobis, présentée dans l'équation 4.3. Pour un ensemble de vecteurs $X = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$, avec K_X la matrice de covariance associée à X, la distance de Mahalanobis D_M entre deux vecteurs $\boldsymbol{x}_i$ et $\boldsymbol{x}_j$ de X sera défini par l'équation :

$$D_{Mahalanobis} = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top K_X (\boldsymbol{x}_i - \boldsymbol{x}_j)} \tag{4.3}$$

En effet, selon Lee et al. [76], celle-ci sera plus adaptée pour la séparation des anomalies multivariées que nous sommes susceptibles de rencontrer. Elle nécessite cependant de calculer une matrice de covariance. Nous décrivons dans les paragraphes suivants la façon dont nous utiliserons les éléments $\mathbf{Y_T}$ pour cela.

Approche globale

Notre première approche sera qualifiée de globale. Nous la désignerons par la suite par LOF_g. Elle est représentée de façon schématique dans la figure 4.6.

La totalité des éléments de $\mathbf{Y_T}$ sera utilisée pour calculer la matrice de covariance $\mathbf{K_T} = cov(\mathbf{Y_T})$. En procédant de cette façon, les anomalies détectées seront des anomalies globales par rapport aux données d'entrainement. Une fois la matrice de covariance calculée, le LOF_g peut être utilisé pendant l'inférence. Pour chaque prédiction $\hat{y}$ faite par le réseau sur une entrée

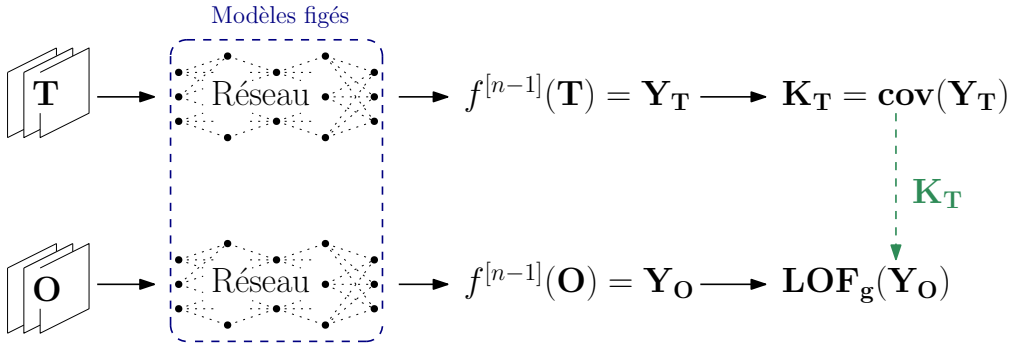


FIGURE 4.6 – Principe de l'approche globale pour l'utilisation du LOF.

$\boldsymbol{x}$, nous calculons son score $LOF(f^{[n-1]}(\boldsymbol{x}))$. Si $LOF(f^{[n-1]}(\boldsymbol{x})) > 1 + \epsilon$, alors $\boldsymbol{x}$ est considéré comme une anomalie. Si $LOF(f^{[n-1]}(\boldsymbol{x})) \leq 1 + \epsilon$, $\boldsymbol{x}$ est un exemple régulier.

Par l'utilisation d'une fonction d'erreur comme l'erreur quadratique moyenne ou l'entropie croisée, la classe de chaque entrée influence fortement sa position dans l'espace des sorties de $f^{[n-1]}$ par rapport aux exemples d'une même classe. Une anomalie globale sera en théorie placée à l'écart des groupes formés par les exemples réguliers. Le LOF utilisant la densité locale de l'espace pour identifier les anomalies, celles-ci seront théoriquement identifiée.

Approche par classe

En s'inspirant de Lee et al. [76], une approche par classe est proposée. Elle sera désignée par LOF_c. Cette approche a été introduite pour déterminer si l'efficacité du LOF peut être améliorée en comparant une anomalie, à laquelle le réseau a attribué une classe c, aux exemples correspondant à cette classe c uniquement. Son principe général est présenté dans la figure 4.7.

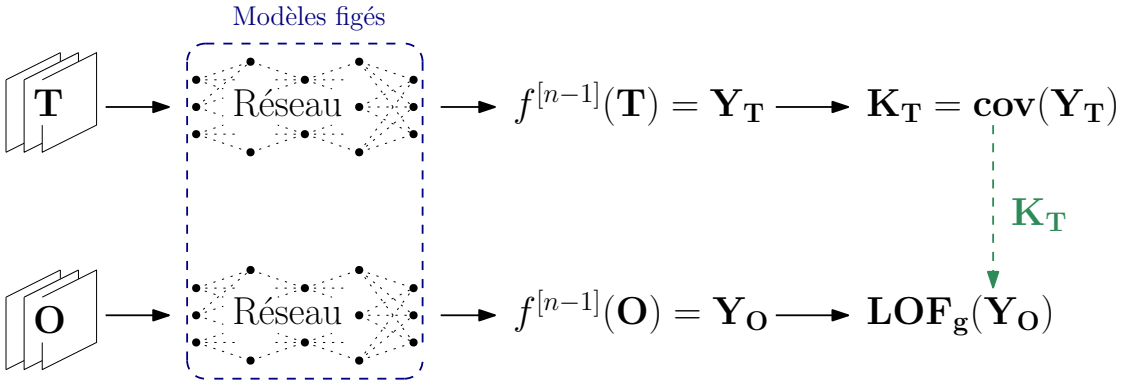


FIGURE 4.7 – Principe de l'approche par classe pour l'utilisation du LOF.

Ainsi, plutôt que d'utiliser une unique matrice de covariance, nous en calculons autant qu'il y a de classes présentes dans les données d'entrainement. Nous définissons ainsi c matrices, une

pour chaque classe i : $\mathbf{K_{T,i}} = cov(\mathbf{Y_{T,i}})$.

À la différence de l'approche globale, nous utilisons le résultat de $\hat{y}$ pour déterminer quelle matrice de covariance sera utilisée pour calculer le LOF. Comme indiqué dans la figure 4.7, le LOF est ensuite appliqué en fonction de $\hat{y}$ de la même manière que pour l'approche globale. Pour simplifier le LOF$_c$, le choix de la grandeur ϵ qui sert à adapter le seuil de détection sera le même quel que soit la prédiction $\hat{y}$. Mais il peut être adapté de façon empirique si nécessaire pour adapter la réponse du LOF.

4.2.3 Détection d'anomalies avec un 1-class SVM

En parallèle du LOF, une machine à vecteur de supports (SVM) mono-classe, 1-SVM, a été évaluée pour la détection d'anomalies. Le 1-SVM est une adaptation du SVM. Là où le SVM standard repose sur la découverte d'un hyperplan pour séparer deux classes, l'algorithme du 1-SVM est fondé sur la recherche d'un hyperplan entre l'origine de l'espace des caractéristiques que l'on cherche à classer et l'ensemble des points utilisés pour son entrainement. Ensuite, nous cherchons à maximiser la distance entre l'origine et cet hyperplan, afin que le caractère anormal de toute nouvelle donnée puisse être identifié en fonction de sa position par rapport à l'hyperplan. Ce principe est illustré dans la figure 4.8.

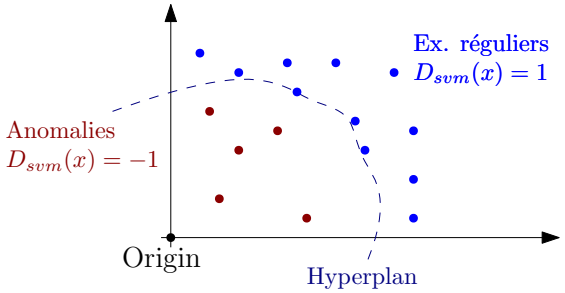


FIGURE 4.8 – Principe simplifié du *one-class*-SVM.

L'entrainement du 1-SVM sur n éléments x_i sert à déterminer la fonction de décision suivante, pour tout nouveau point x à tester :

$$D_{svm}(x) = \text{Sign}\Big(\sum_{i=1}^{n} \alpha_i K(x, x_i) - \rho\Big) \tag{4.4}$$

Où les α_i sont les coefficients de Lagrange et K le noyau. Plusieurs options sont envisageables pour K. Dans notre cas nous utiliserons en guise de noyau, un noyau à fonction à base radiale dite RBF pour *radial basis function* : $K(x, x_i) = \exp(-\gamma \|x - x_i\|_2^2)$. Le coefficient γ permet de régler l'influence des données d'entrainement sur la forme de la fonction de décision. Pour un γ élevé, seul les exemples proches de la frontière de décision seront utilisée pour déterminer sa forme. Un γ faible inclut des exemples plus lointains.

Comme entrée du 1-SVM, nous utiliserons la sortie de l'avant dernière couche, i.e $f^{[n-1]}$, comme pour l'approche présenté en 4.2.2. Le signe de D_{svm} servira à séparer les exemples réguliers des anomalies comme illustré dans la figure 4.8. Pour l'approche SVM, seule une approche globale sera présentée ici. En effet, une approche par classe a été envisagée mais fournissait des résultats trop variables et souvent très faibles comparés à une approche globale avec les données dont nous disposions.

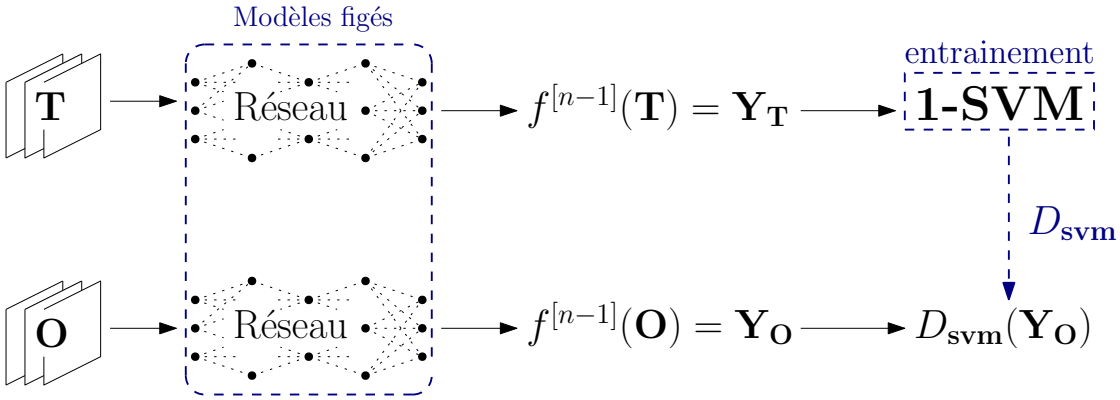


FIGURE 4.9 – Utilisation du 1-SVM pour la détection d'anomalies.

4.3 Résultats sans transfert de domaine

Dans un premier temps, nous testerons les méthodes présentées sans changement de domaine. Nous entrainerons et testerons un CNN avec son détecteur d'anomalies (LOF_g, LOF_c et SVM) sur des données issues du même domaine. Nous évaluerons leurs performances pour la détection de nouvelles classes, de fonds ainsi que la détection d'images dites "canoniques". En premier lieu nous évaluerons les performances des trois méthodes, i.e. LOF_g, LOF_c et SVM, sur SENSIAC pour déterminer laquelle sera utilisée pour la suite des expérimentations. Une fois cette méthode choisie nous l'évaluerons sur les données simulées MBDA.

4.3.1 Évaluation sur SENSIAC

Préparation du jeu de données

Le jeu de données utilisé dans cette section sera SENSIAC [90]. Nous avons sélectionné 8 cibles parmi celles disponibles : le SUV, le Pick-up, le BRDM2, le BTR70, le BMP1, le T72, le ZSU-23 et le 2S3. Pour chacune de ces cibles, nous avons uniquement utilisé les séquences vidéo où le véhicule était le plus proche, i.e. 1500m. Nous disposons donc de deux séquences par véhicule, une de jour et une de nuit, représentant 3600 images par véhicule. Le découpage de cet ensemble est décrit dans le tableau 4.1. Pour éviter tout biais lié au choix de ces cibles, la correspondance

entre les numéros du tableau 4.1 et les cibles du jeu de données SENSIAC changera de façon aléatoire sur les différents entrainements du réseau de neurones et de son détecteur.

Nom	Type d'exemple	Cibles	nbr images	Utilisation
$\mathbf{T}_S$	Réguliers	1,2,3,4,5,6	15120	Entrainement
$\mathbf{V}_S$		1,2,3,4,5,6	6480	Validation
$\mathbf{O}_S$	Nvelles classes	7,8	7200	Test
$\mathbf{F}_S$	Fonds	N/A	4096	Test
$\mathbf{C}$	Ex. Canoniques	N/A	4096	Test

TABLE 4.1 – Détails sur le découpage du jeu de données SENSIAC.

Les exemples dits canoniques de l'ensemble $\mathbf{C}$ sont constitués de formes géométriques simples de taille et de niveau de gris aléatoires combinées à du flou gaussien et un bruit additif gaussien centré sur zéro et de variance aléatoire. Ils sont illustrés dans la figure 4.10. Nous présentons aussi dans la figure 4.11 quelques exemples de fonds de le l'ensemble $\mathbf{F}_S$ prélevés dans les images de SENSIAC. Ces images étant très différentes de $\mathbf{T}_S$ et $\mathbf{V}_S$, nous nous attendons à ce que le LOF_g puisse facilement les détecter. Dans le cas contraire, cela indiquerait que la méthode de détection d'anomalies ou les hyper-paramètres choisis ne sont pas adaptés au problème.

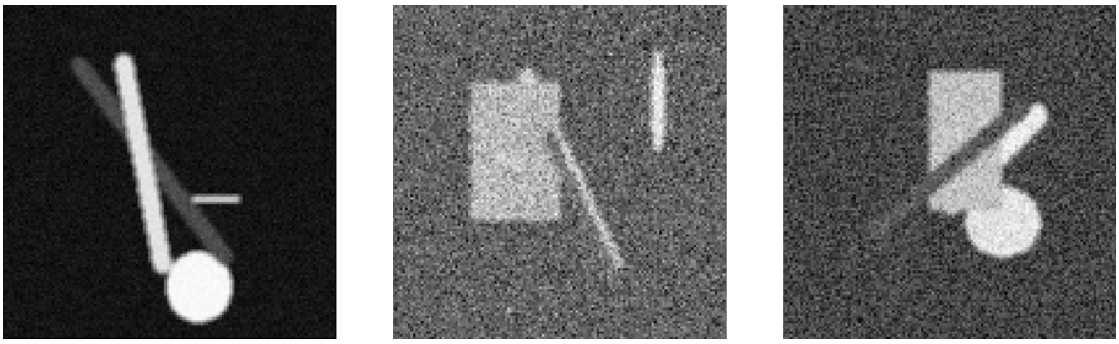


FIGURE 4.10 – Exemples cannoniques utilisés dans $\mathbf{C}$.

Protocole expérimental

Nous cherchons ici à évaluer la capacité du LOF ainsi que celle du 1-SVM en tant que détecteurs d'anomalies de classification tel que décrit dans la section 4.2.2 et 4.2.3 respectivement. Nous utilisons comme réseau de neurones le cfCNN présenté dans le chapitre 3. Ce dernier sera entrainé à plusieurs reprises sur le jeu de données $\mathbf{T}_S$ pour 100 époques avec l'algorithme ADAM avec un taux d'apprentissage de départ de 10^{-3} sur des lots de 64 images. Les six cibles présentes dans $\mathbf{T}_S$ seront différentes à chaque entrainement. L'ensemble $\mathbf{O}_S$ regroupera les deux dernières

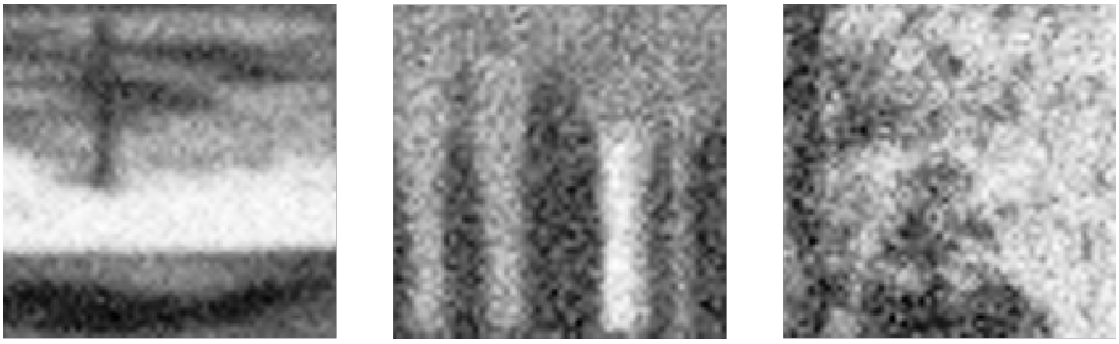


FIGURE 4.11 – Exemples de fonds utilisés dans $\mathbf{F}_S$.

cibles, absentes de $\mathbf{T}_S$, qui serviront de nouvelle classes.

Une fois entrainé, nous avons extrait les sorties de la couche $f^{[n-1]}$ qui précède la sortie Softmax pour l'ensemble des éléments de $\mathbf{T}_S$ et $\mathbf{V}_S$, qui seront stockées respectivement dans $\mathbf{Y_T}$ et $\mathbf{Y_V}$. Nous procédons de façon similaire pour les éléments de $\mathbf{O}_S$, $\mathbf{F_S}$ et $\mathbf{C}$ que nous concaténons pour former $\mathbf{Y_O}$.

Les élements de $\mathbf{Y_T}$ serviront à préparer une matrice de covariance globale $\mathbf{K}_T$ et les six matrices de covariances par classe $\mathbf{K}_{T,y=i}$. À partir de ces matrices, nous pouvons alors préparer deux détecteurs d'anomalies, LOF_g et LOF_c. Le nombre de plus proches voisins utilisés sera de 10. Nous utiliserons une valeur de ϵ de 0.19. Selon Breunig et al. [15] et après différents tests empiriques de notre part, une valeur supérieure n'apporte pas de gain significatif en terme de performance de détection. Par contre, une augmentation de ϵ augmente sensiblement les temps de calcul. Pour le 1-SVM, nous utiliserons une valeur de γ pour le noyau RBF de 5.10^{-3}. Ces quatres détecteurs seront ensuite testés sur les éléments de $\mathbf{Y_V}$ et $\mathbf{Y_O}$. Nous définissons pour chaque détecteur :

- TP : L'ensemble des anomalies correctement détectées.
- FP : L'ensemble des exemples régulier détectés par erreur.
- TN : L'ensemble des exemples régulier non détectés.
- FN : L'ensemble des anomalies non détectées.

Ces grandeurs servent à définir les trois métriques utilisées pour comparer ces approches : la précision $\text{PRE} = \frac{TP}{TP+FP}$, le taux de faux positifs $\text{FPR} = \frac{FP}{FP+TN}$, et le F-score $\text{F1} = \frac{2\times\text{PRE}\times\text{REC}}{\text{PRE}+\text{REC}}$ avec $\text{REC} = \frac{TP}{TP+FN}$. Nous complèterons ces métriques en traçant les courbes ROC, pour *Receiver Operating Characteristic*. Enfin, nos trois propositions seront comparées au détecteur ODIN.

Résultats

Dans un premier temps nous nous intéressons aux courbes ROC du LOF_g , du LOF_c et 1-SVM pour 5 entrainements différents du cfCNN dans la figure 4.12. Les figures 4.12a, 4.12c et 4.12e présentent les courbes entières. Les figures 4.12b, 4.12d et 4.12f, montrent un détail de chaque courbe.. Les courbes entières, dans la partie droite de la figure 4.12, ne permettent pas

facilement de déterminer avec précision lequel des trois détecteurs que nous proposons fonctionne le mieux. Bien que les figures 4.12b et 4.12f montrent des variations entre les différentes courbes ROC, seule la figure 4.12c montre variance plus importante entre les différentes courbes tracées pour le LOF_c. Cette variance est visible plus clairement dans la figure 4.12d.

Pour compléter ces observations un peu limitées, nous avons regroupé dans le tableau 4.2 les métriques des quatre détecteurs testés. Pour chacune d'entre-elles, nous avons indiqué la valeur moyenne et les bornes maximum et minimum obtenues par chaque détecteur pour plus de 10 cycles d'entrainement et de test.

Détecteur	**PRE** **Min/Mean/Max**	**FPR** **Min/Mean/Max**	**F1** **Min/Mean/Max**
ODIN	0.9821/0.9860/1.0000	0.0090/0.0157/0.0233	0.9811/0.9898/0.9902
LOF_g	0.9850/0.9874/0.9919	0.0135/0.0212/0.0256	0.9797/0.9838/0.9889
LOF_c	0.9650/0.9818/0.9919	0.0081/0.0182/0.0350	0.9767/0.9802/0.9884
1-SVM	0.8764/0.9340/0.9704	0.0509/0.1216/0.2394	0.9341/0.9627/0.9776

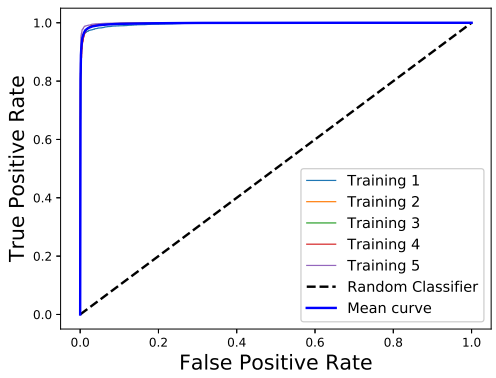
TABLE 4.2 – Performance du LOF avec la distance Mahalanobis pour les deux approches et du 1-SVM comparées a ODIN sur les ensembles combinées $\mathbf{V}_S$, $\mathbf{O}_S$, $\mathbf{F}_S$ et $\mathbf{C}$.

Dans ces résultats, les bonnes performances du LOF_g par rapport aux méthodes que nous avons introduites sont mieux visibles. En effet, il surpasse le LOF_c et le 1-SVM en précision et en score F1, avec une valeur moyenne de 0.9838 tout en restant équivalent au LOF_c pour le FPR, avec une moyenne de 0.0212. Nous pouvons aussi voir que ODIN a obtenu les meilleurs résultats sur les quatre critères. Cependant, il faut noter que les trois méthodes, et notamment le LOF_g, obtiennent des résultats très proches de ODIN. Cela montre l'intérêt des méthodes proposées qui n'utilisent pas de pré-traitement des données comme ODIN.

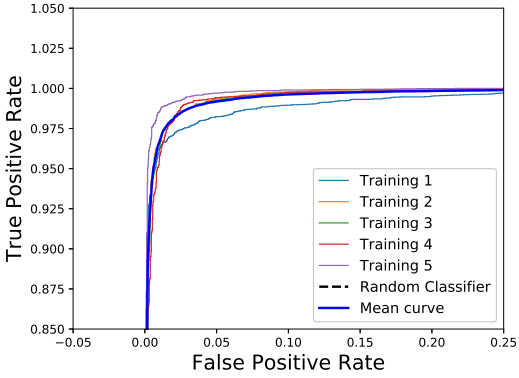
Parmi les autres méthodes que nous avons proposées, l'approche par classe n'a pas permis d'obtenir de meilleurs résultats que l'approche globale. Nous aurions pu supposer qu'en réduisant la complexité du problème de détection d'anomalie en séparant le processus pour chaque classe, les résultats seraient au moins similaires à ceux du LOF_g. Cependant seul le FPR est en moyenne très légèrement inférieur au FPR du LOF_g avec une valeur moyenne de 0.0182.

Malgré les observations faites sur les courbes de la figure 4.12 pour le LOF_c, le 1-SVM montre une variance de performance plus importante que les autres méthodes. Sur les trois critères PRE, FPR et F1, l'écart entre les valeurs minimales et maximales est le plus important de toutes les méthodes. On peut aussi noter une valeur de FPR élevée de 0.1216 en moyenne pour le SVM, par rapport au LOF ou ODIN. Cet écart est répercuté sur la valeur du score F1 qui est légèrement inférieur à tous les autres avec une moyenne de 0.9627 là où les trois autres méthodes ont un score F1 supérieur ou égal à 0.9802.

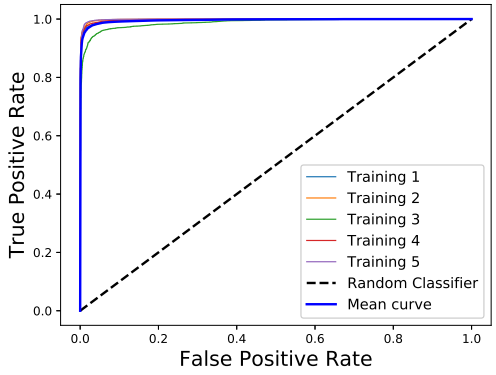
Avec ces données les performances des quatre détecteurs sont très proches et très bonnes. Nous consolidons ces observations avec d'autre expériences dans les sections qui suivent.



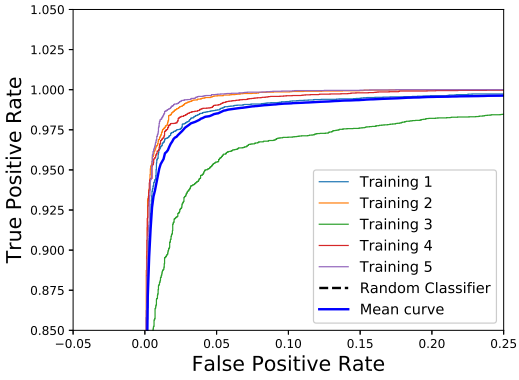
(a) LOF_g



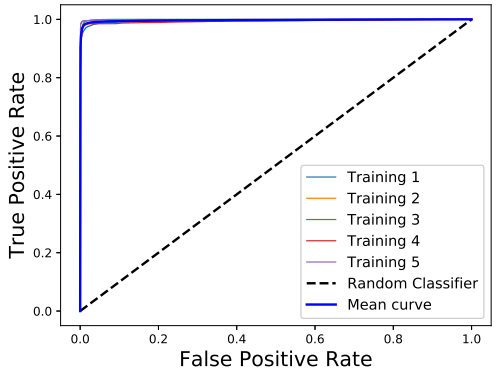
(b) détail de la courbe



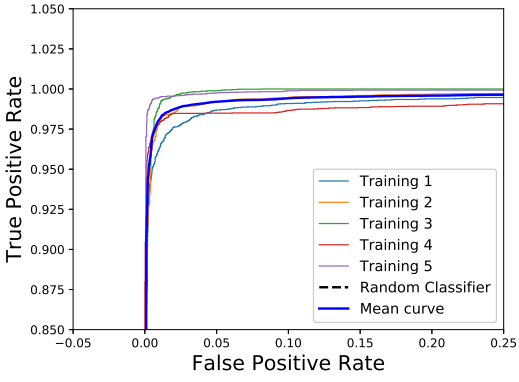
(c) LOF_c



(d) détail de la courbe



(e) 1-SVM



(f) détail de la courbe

FIGURE 4.12 – Courbes ROC du LOF pour les différentes approches sur les ensembles combinées $\mathbf{V}_S$, $\mathbf{O}_S$, $\mathbf{F}_S$ et $\mathbf{C}$.

4.3.2 Détection d'exemples adverses

Dans la section 4.3.1 les anomalies choisies étaient relativement grossières ou visuellement très différentes des exemples réguliers. Même si dans certains cas le réseau peut se tromper avec un score Softmax élevé, il n'est pas exclu que certains de ces exemples puissent être éliminées avec un simple seuillage. Il existe cependant des méthodes pour générer des anomalies difficiles à éliminer par seuillage de façon répétée, comme les exemples adverses.

Dans cette section, nous allons évaluer la réponse de nos différents détecteurs contre des exemples adverses. Présentés dans la section 1.5.3 ces exemples sont construits à partir d'attaques qui visent à modifier la réponse d'un réseau de neurones sur un exemple particulier, en essayant de préserver au maximum l'aspect visuel de l'exemple d'origine.

Protocole et description de l'attaque

Nous réutilisons les cfCNNs et les détecteurs de la section 4.3.1. À partir des images de $\mathbf{V}_S$ nous générons des exemples adverses à l'aide de l'attaque ZOO, ou *Zeroth-Order-Optimization* [21]. Cette attaque a été choisie car il s'agit d'une attaque de type *black-box*, qui ne nécessite pas de modèle de substitution, contrairement à des attaques comme celles présentés par Papernot et al. [98], [96]. L'absence de modèle de substitution permet de simplifier la mise en place de l'attaque et facilite son utilisation. De plus, cette attaque présente des performances proches de l'attaque *white-box* de Carlini et al. [20] qui représente une des méthodes les plus efficaces pour générer des exemples adverses performants.

ZOO a été utilisée en configuration non ciblée. Cela signifie que, pour un couple exemple-label $\{\boldsymbol{x}, \boldsymbol{y}\}$, l'objectif est d'obtenir un score Softmax $p(\hat{y} = c|\boldsymbol{x})$ supérieur à 0.99 pour une classe différente de y. Avec un score de confiance aussi élevé, il est possible que l'attaque soit détectable à l'œil nu. En contrepartie nous nous assurons qu'une détection sur le seuillage des scores Sofmax ne peut être suffisant pour détecter l'exemple adverse. Nous sommes alors en mesure de comparer les performances de nos détecteurs d'anomalies sur des exemples qui, d'après leurs scores Softmax, sont proches d'exemples réguliers vis à vis d'u cfCNN. Étant donné le coût calculatoire pour la génération d'exemples averses, et sachant que nous devons en générer pour chaque entrainement, nous avons limité leur nombre a 500.

Résultats

Le tableau 4.3 regroupe les résultats obtenus sur la détection d'exemples adverses générés via l'attaque ZOO. Là encore, ODIN montre qu'une solution avec pré-traitement obtient de meilleurs performances. Cependant, en moyenne les trois approches (LOF_g, LOF_c et SVM) et ODIN obtiennent des performances équivalentes. Le LOF_g parvient par exemple à détecter 98.89% des exemples adverses en moyenne et le 1-SVM 99.03% en moyenne. Les performances minimum de chaque algorithme sont cependant différentes. Le LOF_c, le SVM et ODIN détectent un minimum de 98% des exemples adverses environ, contre 95.92% pour le LOF_g. Ces bons résultats montrent

qu'ils peuvent tout à fait être utilisés pour cette tâche. En effet, ces approches sont en mesure de détecter des exemples dont la valeur du score Softmax est difficile à discerner de celle d'un exemple régulier.

Détecteur	**Taux de détection (%)**
ODIN	97.55/99.08/100
LOF_g	95.92/98.89/100
LOF_c	98.17/98.74/100
SVM	97.62/99.03/100

TABLE 4.3 – Taux de détection des exemples adverses généré par ZOO.

4.3.3 Performance du détecteur avec peu de données

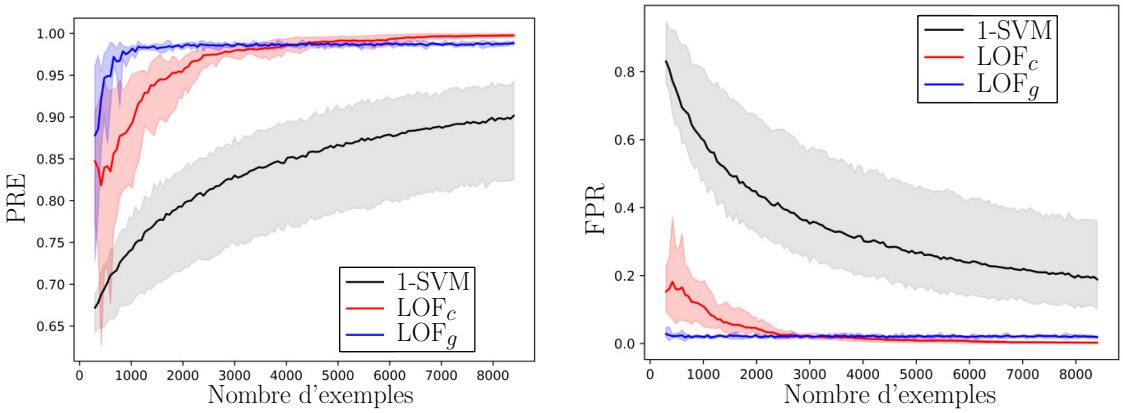
En se basant sur les résultats des sections 4.3.1 e t4.3.2, il est encore difficile de déterminer quelle est la meilleure des méthodes de détection d'anomalies proposées. Comme nous l'avons vu dans la section 2.4, le fait de travailler dans un contexte militaire peut impliquer que la quantité de données dont nous disposons pour entrainer un CNN et son détecteur d'anomalies soit bien plus limité par rapport à la quantité d'images dans SENSIAC.

Nous avons donc voulu évaluer les performances du 1-SVM, du LOF_g et LOF_c lorsqu'ils sont entrainés avec seulement une fraction de l'ensemble $\mathbf{Y_T}$. Pour conserver la cohérence de nos résultats, nous avons réutilisé les cfCNN de la section 4.3.1, et ré-entrainé une série de 1-SVM, du LOF_g et LOF_c avec entre 50 et 8000 exemples tirés aléatoirement de $\mathbf{Y_T}$. Nous les avons ensuite évalué sur l'ensemble combiné $\mathbf{Y_V} \cup \mathbf{Y_O}$.

La figure 4.13 présente les résultats de cette étude pour les mesures de précision et le taux de faux positif pour le LOF_g, le LOF_c et le 1-SVM en bleu, rouge et gris respectivement. Sont aussi visibles sur ces courbes les variances des réponses sur ces critères, symbolisées par les zones colorées semi-transparentes.

Pour les deux critères PRE, dans la figure 4.13a, et FPR, dans la figure 4.13b, le LOF_g et LOF_c atteignent de bonnes performances rapidement à environ 1000 exemples. Le LOF_c accuse un léger retard de performances pour PRE et FPR. Cela peut notamment s'expliquer par le fait que les matrices de covariances par classe $\mathbf{K}_{T,y=i}$ n'utilisent pour cet essai qu'un sixième des données disponibles soit environ 166 exemples d'entrainement. Néanmoins les résultats s'équilibrent vite, et les deux méthodes montrent des performances similaires au-delà de 2000 exemples. On peut observer un léger avantage pour le LOF_c mais les résultats que nous avons vu dans le tableau 4.2 laissent à penser que cette tendance s'inverse avec un nombre d'exemples d'entrainement supérieur.

Dans ces deux figures, Le 1-SVM montre ses faiblesses avec une précision et un taux de



(a) PRE (b) FPR

FIGURE 4.13 – Évolution de PRE et FPR en fonction du nombre d'éléments utilisés pour l'apprentissage .

faux positifs bien supérieur au deux formes du LOF. L'augmentation du nombre d'exemple d'entrainement favorise un gain de précision et une réduction de faux positif mais augmente aussi la variance de la réponse, visible en gris clair sur les figures 4.13a et 4.13b.

4.3.4 Évaluation sur la base MBDA

Dans les sections 4.3.1, 4.3.2 et 4.3.3, nous avons évalué nos trois détecteurs, le LOF$_g$, le LOF$_c$ et un 1-SVM sur des données infrarouges réelles issues de SENSIAC. Le LOF$_g$ a montré de bonnes performances de détections même avec peu d'images d'entrainement. Compte tenu de ces résultats nous l'avons sélectionné pour poursuivre notre étude de performances sur les données de simulation MBDA. Nous étudions ici un cas où les données d'entrainement et de test appartiennent à des distributions proches.

Préparation des données et protocole

Nous utiliserons ici les images issues du jeu de données simulées DS3 présenté dans la section 2.3.1. Cet ensemble de données comprend lui aussi 8 cibles et nous reprendrons un découpage proche de celui proposé dans la section 4.3.1 et détaillé dans le tableau 4.1. Les jeux de données prendront cette fois les noms $\mathbf{T}_M$, $\mathbf{V}_M$, $\mathbf{O}_M$, $\mathbf{F}_M$ et $\mathbf{C}$. Ils auront respectivement 13184, 4096, 5760, 4096 et 4096 images. Nous avons aussi généré 500 exemples adverses à l'aide de ZOO pour chaque cycle d'entrainement et de test. Ce jeu de données prendra le nom $\mathbf{A}_M$. Le détail du découpage sera présenté dans le tableau 4.4.

Une nouvelle fois, nous entrainons plusieurs cfCNNs sur le jeu de données $\mathbf{T}_M$ avec les mêmes paramètres que ceux utilisés dans la section 4.3.1. Avec ces réglages, nous obtenons un pourcen-

Nom	Nature	Type d'exemple	nb. cible	nbr images	Utilisation
$\mathbf{T}_M$	Simulation	Réguliers	6	13184	Entrainement
$\mathbf{V}_M$	Simulation			4096	Validation
$\mathbf{O}_M$	Simulation	Nvelles classes	2	5760	Test
$\mathbf{F}_M$	Simulation	Fonds	N/A	4096	Test
$\mathbf{C}$	Simulation	Ex. Canoniques	N/A	4096	Test

TABLE 4.4 – Détails sur le découpage du jeu de données MBDA.

tage de cibles correctement identifiées supérieur à 98% sur le jeu de données $\mathbf{V}_M$. Une fois les modèles entrainés, nous utiliserons les sorties de l'avant dernière couche $f^{[n-1]}$ de chaque réseau pour former les ensemble $\mathbf{Y_T}$, $\mathbf{Y_V}$ et $\mathbf{Y_O}$ pour chaque cfCNN entrainé. Enfin, nous utiliserons $\mathbf{Y_T}$ pour préparer un LOF$_g$ qui sera testé ensuite sur l'union $\mathbf{Y_V} \cup \mathbf{Y_O}$. Pour l'évaluation nous reprendrons les métriques utilisées dans la section 4.3.1.

Résultats

Dans cette sous-section, nous présentons les performances du LOF$_g$ sur les données MBDA simulées uniquement. Le tableau 4.5 présentera les performances globales et le tableau 4.6 présentera le détail des performances du LOF$_g$ sur chaque type d'anomalies et d'exemples réguliers.

Détecteur	PRE Min/Mean/Max	FPR Min/Mean/Max	F1 Min/Mean/Max
LOF$_g$	0.9621/0.9651/0.9664	0.1001/0.1052/0.1152	0.9386/0.9473/0.9584

TABLE 4.5 – Résultats de détection d'anomalies sur les données MBDA du LOF$_g$ sur les ensembles combinées $\mathbf{V}_M$, $\mathbf{O}_M$, $\mathbf{F}_M$ et $\mathbf{C}$.

Le tableau 4.5 présente les résultats obtenus par le LOF$_g$ sur les données MBDA. En comparant avec les résultats du tableau 4.2, nous pouvons voir que les performances du LOF sont légèrement en retrait sur le jeu de données MBDA. Cependant, nous pouvons remarquer un score de précision de 0.9651 en moyenne et un taux de faux positifs qui ne dépasse pas 0.1152. Cela montre que le détecteur LOF$_g$ reste pertinent pour ce jeu de données. L'écart de performances sur les résultats entre SENSIAC et les données MBDA pourrait être lié à la diversité des images présentes dans l'ensemble $\mathbf{T}_M$ extrait de DS3 et utilisé pour entrainer le cfCNN et le LOF. Il y a dans ce jeu de données plus de variation en termes de météo et d'état thermique par rapport à SENSIAC. En effet, ce dernier ne contient que des véhicules en roulement et pour seulement deux météos. Ces variations peuvent affecter la capacité du détecteur d'anomalies à bien fonctionner.

Les résultats de détection sur chaque ensemble de données sont présentés dans le tableau

Données	**Non détectés (réguliers)** Min/Mean/Max%	**Détecté comme anomalie** Min/Mean/Max%
Exemples réguliers		
$\mathbf{V}_M$	88.5/89.5/90%	10/10.5/11.5%
Anomalies et nouvelles classes		
$\mathbf{F}_M$	5.6/12.5/16.2%	83.8/87.5/94.4%
$\mathbf{O}_M$	2.6/5.64/8.9%	91.1/94.36/97.4%
$\mathbf{C}$	1.6/2.7/4.63%	95.37/97.234/98.4%
$\mathbf{A}_M$	6.4/7.82/9.7%	90.3/92.18/93.6%

TABLE 4.6 – Détails par type d'anomalie des taux de détection du LOF_g.

4.6. Dans ce tableau, nous considérons que, pour un bon détecteur d'anomalies, le pourcentage d'exemples réguliers, ici $\mathbf{V}_M$, non détectés doit être élevé et le pourcentage d'exemple réguliers détectés comme anomalies doit être faible. Inversement, pour un bon détecteur, le pourcentage d'anomalies non détectées doit être faible.

Sous ce format, nous pouvons voir que le FPR élevé du tableau 4.5 est confirmé par le nombre d'exemple de $\mathbf{V}_M$ détectés comme anomalies. Idéalement, l'ensemble du jeu de données $\mathbf{V}_M$ devrait être considéré comme régulier. En effet, cet ensemble ne contient que des exemples sur lesquels le réseau a été entrainé.

En ce qui concerne les anomalies, nous pourrions nous attendre à voir un grand nombre d'exemples adverses ne pas être détectés par le LOF_g. Cependant, nous pouvons voir que les fonds du jeu de données $\mathbf{F}_M$ sont les plus difficiles à gérer pour le LOF_g. En moyenne, 12.5% d'entre eux ne seront pas détectés par le LOF_g, contre 7.82% des éléments de $\mathbf{A}_M$.

En ce qui concerne les autres détecteurs présentés dans la section 4.3.1, nous tenons à apporter quelques précisions pour justifier leur absence dans cette section. Pour ODIN, des problèmes d'utilisation de certaines librairies sur les machines respectant les conditions de sécurité imposées par MBDA nous ont empêché d'obtenir des résultats. Pour le 1-SVM, il est impossible d'obtenir un FPR faible avec un score PRE élevé comme pour les résultats sur 4.3.1.

4.4 Résultats avec transfert de domaine

Les résultats des sections précédentes ont montré que le LOF_g permettait de détecter un grand nombre d'anomalies de classification sur deux jeux de données d'images infrarouges. Dans le chapitre 3, nous avons entrainé un CNN compact sur des données simulées puis testé ce réseau sur des données réelles. Dans cette section nous évaluons le LOF_g sur des données réelles après avoir été préparé dur des données simulées. Pour cela, nous entrainons simultanément un cfCNN et un détecteur d'anomalies basé sur le LOF_g sur des données simulées et nous testons ensuite

l'ensemble sur des données réelles.

4.4.1 Essais avec le LOF_g

Préparation des données

Nous réutilisons les données de l'ensemble DS3, avec un découpage légèrement différent de celui introduit dans la section 4.3.4, en utilisant l'ensemble des cibles présentes dans DS3 pour former nos ensembles d'entrainement et de test. Nous les appellerons respectivement $\mathbf{T}_R$ et $\mathbf{V}_R$. Aussi, les nouvelles classes seront cette fois ci des images réelles de cibles inconnues qui seront regroupées dans l'ensemble appelé $\mathbf{O}_R$. Les cibles en question sont une camionnette et un pick-up. Enfin, les fonds ne seront plus tirés d'images synthétiques mais d'images réelles. Le détail du découpage de ces ensembles est présenté dans le tableau 4.7.

Nom	Nature	Type d'exemple	nb. cible	nbr images	Utilisation
$\mathbf{T}_R$	Simulation	Réguliers	8	18944	Entrainement
$\mathbf{V}_R$	Simulation			4096	Validation
$\mathbf{R}$	Réel			43810	Test
$\mathbf{O}_R$	Réel	Nvelles classes	2	1862	Test
$\mathbf{F}_R$	Réel	Fonds	N/A	4096	Test
$\mathbf{C}$	Simulation	Ex. Canoniques	N/A	4096	Test

TABLE 4.7 – Détails sur le découpage du jeu de données MBDA.

Protocole expérimental

Comme pour les sections 4.3.1 et 4.3.4, nous entrainerons plusieurs itérations du cfCNN sur les données du jeu $\mathbf{T}_R$ avec les mêmes paramètres. Une fois les modèles entrainés, nous regroupons les sorties de la couche $f^{[n-1]}$ associées à $\mathbf{V}_R$ et $\mathbf{R}$ dans l'ensemble $\mathbf{Y}_V$, et celles des ensembles $\mathbf{O}_R$, $\mathbf{F}_R$ et $\mathbf{C}$ dans $\mathbf{Y}_O$. Enfin, nous préparons une nouvelle fois un LOF_g pour chaque cfCNN entrainé sur les sorties associées à $\mathbf{T}_R$. Ce LOF_g est ensuite testé sur $\mathbf{Y}_V \cup \mathbf{Y}_O$. Nous réutiliserons les critères PRE, FPR et F1 pour évaluer les performances de ce détecteur.

Résultats

Nous présentons ici les résultats des test de détection d'anomalies avec le LOF_g sur les données MBDA réelles. Les performances globales du détecteur sont référencées dans le tableau 4.8 et le détail des taux de détection par type d'anomalie sera présenté dans la section 4.9.

A partir des données présentées dans le tableau 4.8, nous pouvons voir que le LOF_g ne fonctionne pas correctement sur ce jeu de données. En comparant avec les résultats des tableaux

Détecteur	**PRE** **Min/Mean/Max**	**FPR** **Min/Mean/Max**	**F1** **Min/Mean/Max**
LOF_g	0.1437/0.2368/0.4794	0.5205/0.5661/0.6074	0.2324/0.3118/0.5140

TABLE 4.8 – Résultats de détection d'anomalies sur les données MBDA réelles, combinées aux ensembles $\mathbf{V}_M$, $\mathbf{O}_M$, $\mathbf{F}_M$ et $\mathbf{C}$.

4.2 et 4.5 nous pouvons voir que la PRE et le score F1 sont bien en dessous de ce que nous avions obtenu sans transfert de domaine. Le FPR est aussi très élevé, avec une valeur moyenne de 0.5661, qui indique qu'au moins un exemple régulier sur deux est identifié par erreur comme une anomalie. Un élément important est aussi la variance des résultats présentés, notamment pour PRE et le score F1. Une telle instabilité pour un module de détection d'anomalies, notamment dans un contexte militaire, n'est pas acceptable.

Données	**Non détectés (réguliers)** Min/Mean/Max%	**Détecté comme anomalie** Min/Mean/Max%
Exemples réguliers		
$\mathbf{V}_R$	99.1/99.4/99.7%	0.3/0.5/0.9%
$\mathbf{R}_R$	17.1/43.2/86.3%	13.7/56.8/82.9%
Anomalies et nouvelles classes		
$\mathbf{F}_R$	64.8/74.5/84%	16/25.5/35.2%
$\mathbf{O}_R$	28.4/37.8/48.9%	51.1/62.2/71.6%
$\mathbf{C}$	11.5/14.8/18.6%	81.4/85.2/88.5%

TABLE 4.9 – Détails par type d'anomalie des taux de détection du LOF_g.

Le détail des performances du LOF_g sur chaque ensemble d'images est présenté dans le tableau 4.9. Nous pouvons voir que sur $\mathbf{V}_R$ et $\mathbf{C}$, les performances se rapprochent des résultats de la section 4.3.4. Le taux de fausses alarmes sur $\mathbf{V}_R$ reste faible, à 0.5% en moyenne. Cependant, le nombre d'exemples cannoniques non détectés est en augmentation à 11.5% au minimum.

Les mauvais résultats du LOF_g sont principalement dus aux performances de détection sur $\mathbf{R}_R$, $\mathbf{F}_R$ et $\mathbf{O}_R$. Nous pouvons voir que jusqu'à 82.9% des exemples réguliers réels de $\mathbf{R}_R$ son détectés par erreur comme des anomalies. Inversement, seulement 25.5% des fonds de $\mathbf{F}_R$, et 62.2% des nouvelles classes de $\mathbf{O}_R$ sont correctement détectées par le LOF_g.

4.4.2 Proposition d'un détecteur en cascade

Description du nouveau détecteur

Les résultats présentés dans la section 4.4.1 montrent que le LOF_g n'est pas adapté pour la détection d'anomalies avec passage de la simulation au réel. Pour améliorer les résultats de détection en utilisant toujours la base du LOF tel que décrit dans la section 4.2.2, nous proposons de combiner l'information issue de plusieurs couches du réseau utilisé pour la classification. Nous allons donc utiliser plusieurs modules de détection basé sur le LOF_g connectés aux premières couches cachées du réseau, en complément d'un LOF_g connecté à l'avant dernière couche i.e. $f^{[n-1]}$. En procédant de la sorte, nous espérons pouvoir exploiter le changement de la représentation des données d'entrée entre chacune des couches du réseau.

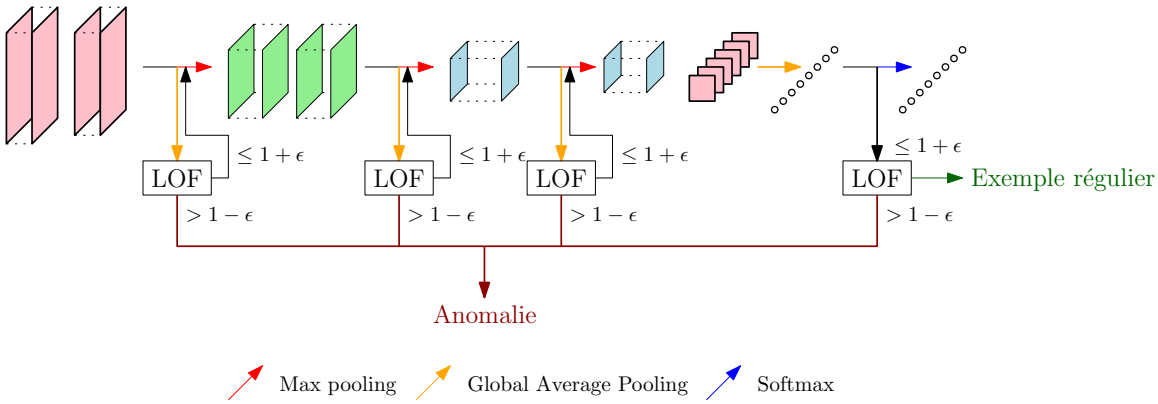


FIGURE 4.14 – Présentation du détecteur en cascade ou CLOF.

Ce nouveau module, présenté dans la figure 4.14, sera appelé LOF en cascade ou CLOF. Dans cet exemple, le réseau utilisé est un cfCNN. Celui-ci étant entièrement convolutif, nous avons ajouté entre chaque couche de convolution et chaque LOF intermédiaire une opération de GAP pour réduire la taille des entrées du LOF. En effet, sans cette étape, la dimension des sorties des couches de convolution risque d'être très importante et de limiter fortement la vitesse de traitement pour la détection d'anomalie.

Comme illustré dans la figure 4.14, pour qu'une entrée $\boldsymbol{x}$ soit classée comme une anomalie, seul l'avis d'un détecteur intermédiaire est suffisant. A l'inverse, l'entrée doit avoir été déclarée comme régulière par la totalité des détecteurs pour pouvoir être qualifiée de régulière en sortie du réseau. C'est le passage successif dans les différents détecteurs qui donne son nom de cascade au CLOF.

Résultats

Nous avons répété les expériences de la section 4.4.1 en utilisant le CLOF pour mesurer les gains obtenus par rapport au LOF_g simple sur les mêmes ensembles d'images. Les résultats

globaux sont présentés dans le tableau 4.10 et détaillés dans le tableau 4.11.

Détecteur	**PRE** Min/Mean/Max	**FPR** Min/Mean/Max	**F1** Min/Mean/Max
CLOF	0.6525/0.6673/0.6995	0.1954/0.2506/0.2780	0.6696/0.6835/0.6938

TABLE 4.10 – Résultats de détection d'anomalies sur les données MBDA réelles, combinées aux ensembles $\mathbf{V}_M$, $\mathbf{O}_M$, $\mathbf{F}_M$ et $\mathbf{C}$.

Nous pouvons voir que les valeurs minimum pour PRE et F1 du CLOF dépassent le maximum obtenu pour le LOF$_g$ dans le tableau 4.8 . Nous pouvons aussi noter que le FPR est fortement descendu à 0.2506 en moyenne. Les résultats sur les trois critères semblent aussi bien plus stables que pour le LOF$_g$.

Données	**Non détectés (réguliers)** Min/Mean/Max%	**Détecté comme anomalie** Min/Mean/Max%
Exemples réguliers		
$\mathbf{V}_R$	99.4/99.5/99.6%	0.4/0.5/0.6%
$\mathbf{R}_R$	60.9/64.8/72.6%	27.4/35.2/39.1%
Anomalies et nouvelles classes		
$\mathbf{F}_R$	35.2/42.9/48.8%	51.2/57.1/64.8%
$\mathbf{O}_R$	30.8/34.8/59.9%	40.1/65.2/69.2%
$\mathbf{C}$	9.4/10.9/12.3%	87.7/89.1/90.6%

TABLE 4.11 – Détails par type d'anomalie des taux de détection du LOF$_g$.

En regardant les résultats présentés dans le tableau 4.11, nous pouvons confirmer l'amélioration de la performance du CLOF. L'utilisation du CLOF a notamment permis de réduire fortement le taux d'exemples détectés par erreur sur l'ensemble $\mathbf{R}_R$, qui est maintenant de 35.2% en moyenne, contre 56.8% pour le LOF$_g$. De plus le pourcentage de fonds réels correctement détectés comme anomalies à lui aussi augmenté. Il passe de 25.5% en moyenne dans le tableau 4.9 à 57.1% pour le CLOF. Enfin, nous pouvons aussi voir que le gain de stabilité observée sur les données du tableau 4.10 est confirmée dans le tableau 4.11. Le détecteur en cascade semble mieux adapté au problème de la détection d'anomalie avec changement de domaine.

4.5 Conclusion sur la détection d'anomalies

Dans ce chapitre, nous avons introduit plusieurs façons de détecter différentes formes d'anomalies de classification pour les réseaux de neurones. Parmi ces méthodes nous avons montrés

que celle basée sur le LOF, et que nous avons appelé LOF_g, nous permettait d'obtenir des résultats équivalents à une méthode de la littérature appelée ODIN. Cette équivalence ne concerne cependant que des résultats ou les données d'entrainement du LOF_g et les données de test appartenaient au même domaine.

Dans un contexte ou le domaine change entre l'entrainement et le test, les performances LOF_g ne sont pas satisfaisants. Pour traiter ce problème, nous avons introduit une nouvelle façon d'utiliser le LOF baptisée CLOF. Ce CLOF est une cascade de LOF_g connectés à plusieurs couches du réseau de neurones qu'il supporte. Grâce à cette évolution nous sommes en mesure de détecter un plus grand nombre d'anomalies tout en réduisant sensiblement le taux de faux positifs.

Conclusion

Ce chapitre clôture ce manuscrit en présentant un bilan des travaux réalisés au cours de cette thèse. Nous allons donc aborder successivement l'utilisation de données simulées pour l'entrainement d'un CNN dédié à la classification d'images infrarouges réelles et la détection d'anomalies de classification. Pour chacune de ces contributions, nous proposerons aussi des axes d'amélioration.

Classification d'images infrarouges réelles à partir de données simulées

Nous avons dans un premier temps traité le problème de la reconnaissance et de l'identification, ou RI, de véhicules dans des images infrarouges à l'aide d'un réseau de neurones. Le premier problème à résoudre concerne la quantité de données disponible pour entrainer un CNN. En effet, nous avons indiqué qu'il était difficile de construire des jeux de données exhaustifs et annotés pour les tâches de reconnaissance et d'identification. Cela peut rendre difficile l'utilisation de CNN pour la RI en infrarouge car leurs performances sont en partie dépendantes de la quantité d'images utilisées pour les entrainer.

Pour compenser le manque d'images d'entrainement, une première approche est de pré-entrainer un CNN. Cela suppose de disposer de deux jeux de données : un jeu de petite taille qui correspond au problème à résoudre, ou jeu ciblé, et un jeu plus important pour un problème de classification différent, ou jeu source. Nous commençons alors par entrainer le CNN sur le jeu source avant d'affiner l'apprentissage avec le jeu de données ciblé. Cette approche permet d'atteindre rapidement de bonnes performances pour des tâches de classification sans utiliser d'ensembles de données d'entrainement de grande taille pour le domaine visé. Elle permet aussi de ré-utiliser rapidement des modèles de CNN issus de la littérature qui auraient été pré-entrainés sur le jeu source. Néanmoins, nous avons mentionné des travaux de recherche sur cette pratique et les risques qu'elle présente pour des applications militaires. Elle autorise une attaque indirecte du modèle pré-entrainé en empoisonnant le jeu source.

Nous avons donc privilégié l'utilisation de jeux de données simulées pour entrainer les CNN destinés à la classification d'images. Dans un premier temps, nous avons évalué l'influence de la qualité de la simulation sur les résultats de la classification en utilisant des modèles de CNN

issus de la littérature ainsi qu'un algorithme de classification basé sur un SVM et des descripteurs HOG. Nous disposions pour cela de trois ensembles de données simulées avec différents niveaux de qualité et un ensemble de données réelles. Nous avons comparé les différents modèles de classification en fonction de leurs scores de reconnaissance et d'identification. Nous les avons tous entrainés successivement sur les trois ensembles de données simulées avant de les tester sur les données réelles.

Les résultats obtenus ont d'abord montré que, comme nous pouvions le supposer, améliorer la qualité de la simulation entraine un gain de performance significatif pour la classification de véhicules dans les images réelles. Cependant, nous avons aussi observé que parmi les architectures de CNN choisies, seule un CNN simple et un Resnet 52 parvenaient à égaler ou surpasser le SVM en termes de performances en identification. Nous avons alors proposé un nouveau modèle de CNN compact pour améliorer les performances de classification. Baptisé cfCNN, ce réseau est un CNN entièrement convolutif à 7 couches cachées, terminé par une opération de *global average pooling*, ou GAP, et dont les neurones utilisent l'activation Leaky-ReLU. Ce réseau a obtenu les meilleurs résultats de reconnaissance et d'identification quelle que soit la qualité des données simulées utilisées pour l'entrainement par rapport aux autres modèles testés. L'utilisation de l'activation Leaky-ReLU a notamment permis de réduire la variance des résultats de classification. De plus, en comparant le cfCNN avec une version modifiée utilisant deux couches de neurones entièrement connectées à la place du GAP, nous avons montré que le GAP améliorait les performances du cfCNN sur des données réelles après un entrainement sur des données simulées.

Dans un deuxième temps, nous avons évalué la robustesse du cfCNN aux erreurs de détection. En effet, les systèmes de RI sont précédés d'un module de détection qui découpe une image source en zone d'intérêt. Ces zones sont ensuite traitées par le système RI. Ainsi, une erreur de l'étage de détection pourra impacter la qualité de la réponse du module RI. Pour évaluer cet impact, nous avons repris certains des modèles précédents et nous avons testé la réponse de chacun sur des données simulées perturbées. En choisissant de réaliser ces évaluations sur les données simulées nous nous sommes assurés que les variations de performances observées étaient uniquement liées aux perturbations que nous avons ajouté. Ces perturbations comprenaient des translations, des changements d'échelle et l'ajout de bruit. Là encore le cfCNN a obtenu les meilleurs résultats notamment grâce à l'utilisation du GAP. Ces résultats montrent le potentiel du cfCNN en vue de développer un système de RI automatisé. Sa compacité accélère la vitesse de l'apprentissage et de l'inférenc et pourra faciliter son implémentation sur des cibles matérielles avec peu de puissance de calcul. De plus, des travaux complémentaires ont montré que le cfCNN pouvait être utilisé pour des tâches de détection de petites cibles dans les images infrarouges [4].

Après avoir travaillé sur le cfCNN, nous avons identifié plusieurs axes de développement pour améliorer les résultats présentés dans ce manuscrit :

- Dans un premier temps, nous pourrions travailler à améliorer les résultats de classification sur données réelles après avoir appris sur des données simulées. Nous pouvons suggérer deux axes différents. Le premier axe serait d'utiliser dans l'architecture du cfCNN des

Gradient Reversal Layers [130]. Ces couches permettent de faire du transfert de domaine non-supervisé et d'entrainer conjointement un CNN sur des données simulées annotées et des données réelles non annotées. Nous pourrions ainsi utiliser pendant l'entrainement des images simulées de véhicules militaires et des images réelles infrarouges de véhicule civils qui sont plus simples à obtenir. En deuxième axe, nous proposons d'évaluer l'apport des réseaux génératif antagonistes, ou GAN, pour améliorer la qualité de la synthèse et réduire l'écart visuel entre des images réelles et des images simulées.

- Dans un deuxième temps, nous pourrions aussi nous intéresser à la robustesse du cfCNN sur d'autres critères. Nous pourrions ainsi commencer par étudier un problème d'identification ou de reconnaissance avec un nombre de cibles plus important, à condition de disposer de suffisamment de données annotées. Un autre point non abordé dans le manuscrit concerne les effets du masquage partiel des cibles par des éléments de l'environnement. Il serait aussi intéressant d'évaluer la robustesse de la classification en fonction de l'orientation du véhicule, ou même de la position de certains éléments du véhicule comme la tourelle d'un char de combat.
- Enfin, il serait aussi utile d'étudier la possibilité de quantifier le réseau, *i.e.* de travailler avec un cfCNN dont les paramètres ne sont pas codés sur 32 bits mais 16 ou 8 bits. Dans le même thème, nous pourrions aussi évaluer l'impact de la dynamique de l'image sur les performances du réseau. En effet, dans le cadre de ces travaux, nous avons simplement déterminé que de travailler avec la dynamique complète de l'image, *i.e.* 14 bits, nous permettait d'obtenir les meilleurs scores d'identification et de reconnaissance. Il serait intéressant de quantifier l'écart de performance avec un cfCNN qui utilise des entrées quantifiées sur moins de 14 bits.

Détection d'anomalies de classification

Nous avons ensuite traité le problème de la détection des erreurs de classification qui a déjà été abordé dans la littérature. Cependant, les méthodes les plus efficaces nécessitent de pré-traiter l'image en entrée et de calculer les gradients du modèle par rapport à cette entrée pendant l'inférence. Nous avons donc proposé trois approches basées sur la détection des anomalies à partir des sorties de la couche $f^{[n-1]}$ d'un CNN, qui précède la sortie softmax : une utilisant un SVM à une classe, ou 1-SVM, et les deux autres basées sur le *Local Outlier Factor* ou LOF. Le LOF est un algorithme de détection d'anomalies utilisant des calculs de distance, ici la distance entre un point et ses plus proches voisins. Nous l'avons utilisé selon deux approches : une approche par-classe LOF_c et une approche globale LOF_g. Dans les deux cas, la distance utilisée était la distance de Mahalanobis. Pendant l'entrainement du CNN, les sorties de $f^{[n-1]}$ sont utilisée pour entrainer le 1-SVM et calculer les matrices de covariances des approches basées sur le LOF. Une fois préparés, chaque détecteur peut être utilisé pendant l'inférence pour déterminer si une entrée est anormale ou régulière.

Ces trois méthodes de détection d'anomalies ont été couplées au cfCNN et comparées à la méthode ODIN pour détecter plusieurs anomalies basées sur le jeu de données SENSIAC : des nouvelles classes de véhicules, des fonds d'images infrarouges sans cibles, des exemples canoniques et des exemples adverses générés à l'aide de la méthode ZOO. Les trois approches proposées, et notamment le LOF_g, ont montré des performances proches de ODIN. À la lumière de ces résultats, nous avons donc utilisé un cfCNN avec le LOF_g sur les données MBDA. Comme pour le problème de la classification d'images à partir de données simulées, nous avons entrainé le cfCNN et le LOF_g sur des données simulées, puis nous l'avons testé sur des données réelles. Sur ce test, le taux de faux positifs sur les données réelles a augmenté significativement par rapport aux résultats obtenus sur SENSIAC.

Pour améliorer ces résultats, nous avons proposé un détecteur en cascade, ou CLOF, utilisant une combinaison de LOF_g sur plusieurs couches du cfCNN. Cela permet d'exploiter le changement de distribution des variables des différentes couches du réseau. Cette nouvelle approche nous a permis d'améliorer fortement les performances de détection. Il reste cependant une marge de progression importante avant d'atteindre les résultats obtenus sur le problème de la détection d'anomalie sans transfert de domaine.

Nous avons identifié plusieurs axes d'amélioration ou de recherches pour la détection d'anomalie :

- Le CLOF et le LOF_g utilisent tous les deux des scores obtenus en sortie d'une couche GAP. Nous avons choisi cette approche pour sa simplicité car elle permet de réduire fortement la taille des vecteurs utilisés en entrées des blocs LOF et donc de réduire le temps de traitement. Néanmoins en appliquant cette opération, il est possible que des informations utiles pour caractériser les anomalies soient perdues. Il serait alors intéressant d'étudier d'autres métriques pour extraire les informations en sortie des couches de convolution du réseau.
- Un autre axe d'amélioration serait d'essayer de séparer les différents types d'anomalies en fonction des scores obtenus avec le CLOF ou le LOF_g. Cela permettrait d'améliorer l'interprétation des résultats et de faciliter l'utilisation du système pour un opérateur.

Bibliographie

[1] A. Abdelkader et al. “Headless Horseman : Adversarial Attacks on Transfer Learning Models”. In : *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Mai 2020, p. 3087-3091 (cf. p. 57).

[2] A. Nguyen, J. Yosinski et J. Clune. “Deep neural networks are easily fooled : High confidence predictions for unrecognizable images”. In : *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Juin 2015, p. 427-436 (cf. p. 90).

[3] C. C. Aggarwal, A. Hinneburg et D. A. Keim. “On the Surprising Behavior of Distance Metrics in High Dimensional Spaces”. In : *Proceedings of the 8th International Conference on Database Theory*. ICDT ’01. Berlin, Heidelberg : Springer-Verlag, 2001, p. 420-434 (cf. p. 98).

[4] Alexandre Baussard et al. “Faster-RCNN with a compact CNN backbone for target detection in infrared images”. In : t. 11543. Sept. 2020 (cf. p. 118).

[5] A. Alwosheel, S. van Cranenburgh et C. G. Chorus. “Is your dataset big enough ? Sample size requirements when using artificial neural networks for discrete choice analysis”. In : *Journal of Choice Modelling* 28 (sept. 2018), p. 167-182 (cf. p. 31).

[6] A. Athalye, N. Carlini et D. Wagner. “Obfuscated Gradients Give a False Sense of Security : Circumventing Defenses to Adversarial Examples”. In : *Proceedings of the 35th International Conference on Machine Learning*. Sous la dir. de J. Dy et A. Krause. T. 80. Proceedings of Machine Learning Research. Stockholm Sweden : PMLR, juill. 2018, p. 274-283 (cf. p. 92).

[7] B. L. Kalman et S. C. Kwasny. “Why tanh : choosing a sigmoidal function”. In : *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*. T. 4. Juin 1992, 578-581 vol.4 (cf. p. 25).

[8] B. Zhou et al. “Learning Deep Features for Discriminative Localization”. In : *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Juin 2016, p. 2921-2929 (cf. p. 64).

[9] A. Bakry et al. “Digging Deep into the layers of CNNs : In Search of How CNNs Achieve View Invariance”. In : *arXiv e-prints* (août 2015), arXiv :1508.01983 (cf. p. 63).

[10] O. Bastani et al. “Measuring Neural Net Robustness with Constraints”. In : *Advances in Neural Information Processing Systems 29*. Sous la dir. de D. D. Lee et al. Curran Associates, Inc., 2016, p. 2613-2621 (cf. p. 92).

[11] H. Bay et al. “Speeded-Up Robust Features (SURF)”. In : *Computer Vision and Image Understanding* 110.3 (juin 2008), p. 346-359 (cf. p. 14).

[12] Y. Bengio. “Practical Recommendations for Gradient-Based Training of Deep Architectures”. In : *Neural Networks : Tricks of the Trade : Second Edition*. Sous la dir. de G. Montavon, G. B. Orr et K.-R. Müller. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, p. 437-478 (cf. p. 7, 28).

[13] P. Bhattacharya, J. Riechen et U. Zölzer. “Infrared Image Enhancement in Maritime Environment with Convolutional Neural Networks”. In : *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2018)*. 2018 (cf. p. 46).

[14] C. M. Bishop. *Pattern Recogniton and Machine Learning*. Springer, 2006 (cf. p. 22, 30).

[15] M. M. Breunig et al. “LOF : Identifying Density-based Local Outliers”. In : *SIGMOD Rec.* 29.2 (mai 2000), p. 93-104 (cf. p. 93, 96, 97, 103).

[16] J. S. Bridle. “Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition”. In : *Neurocomputing*. Sous la dir. de F. F. Soulié et J. Hérault. Springer Berlin Heidelberg, 1990, p. 227-236 (cf. p. 5).

[17] C. Chen et al. “DeepDriving : Learning Affordance for Direct Perception in Autonomous Driving”. In : *2015 IEEE International Conference on Computer Vision (ICCV)*. Déc. 2015, p. 2722-2730 (cf. p. 60).

[18] Z. Cao, X. Zhang et W. Wang. “Forward-Looking Infrared Target Recognition Based On Histograms of Oriented Gradients”. In : *MIPPR 2011 : Automatic Target Recognition and Image Analysis*. T. 8003. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. Nov. 2011, 80030S (cf. p. 46).

[19] N. Carlini et D. Wagner. “Defensive Distillation is Not Robust to Adversarial Examples”. In : *arXiv e-prints* (juill. 2016), arXiv :1607.04311 (cf. p. 92).

[20] N. Carlini et D. Wagner. “Towards Evaluating the Robustness of Neural Networks”. In : *arXiv e-prints* (août 2016), arXiv :1608.04644 (cf. p. 34, 106).

[21] P.-Y. Chen et al. “ZOO : Zeroth Order Optimization Based Black-Box Attacks to Deep Neural Networks without Training Substitute Models”. In : *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. AISec ’17. event-place : Dallas, Texas, USA. New York, NY, USA : Association for Computing Machinery, 2017, p. 15-26 (cf. p. 34, 106).

[22] J. Cho et al. “How much data is needed to train a medical image deep learning system to achieve necessary high accuracy ?” In : *arXiv e-prints* (nov. 2015), arXiv :1511.06348 (cf. p. 31).

[23] Choromanska, A. et al. “The Loss Surfaces of Multilayer Networks”. In : *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. Sous la dir. de Guy Lebanon et S. V. N. Vishwanathan. PMLR, fév. 2015, p. 192-204 (cf. p. 10).

[24] D.-A. Clevert, T. Unterthiner et S. Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. San Juan, Puerto Rico, 2016 (cf. p. 27).

[25] C. Corbière et al. “Addressing Failure Prediction by Learning Model Confidence”. In : *Advances in Neural Information Processing Systems 32*. Sous la dir. de H. Wallach et al. Curran Associates, Inc., 2019, p. 2902-2913 (cf. p. 94).

[26] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In : *Mathematics of Control, Signals and Systems* 2.4 (déc. 1989), p. 303-314 (cf. p. 5, 25).

[27] D. C. Cireşan, U. Meier et J. Schmidhuber. “Transfer learning for Latin and Chinese characters with Deep Neural Networks”. In : *The 2012 International Joint Conference on Neural Networks (IJCNN)*. Juin 2012, p. 1-6 (cf. p. 31).

[28] D. C. Cireşan et al. “Deep, Big, Simple Neural Nets for Handwritten Digit Recognition”. In : *Neural Computation* 22.12 (déc. 2010), p. 3207-3220 (cf. p. 15).

[29] D. Malmgren-Hansen et al. “Improving SAR Automatic Target Recognition Models With Transfer Learning From Simulated Data”. In : *IEEE Geoscience and Remote Sensing Letters* 14.9 (sept. 2017), p. 1484-1488 (cf. p. 61).

[30] D. P. Williams. “Underwater Target Classification in Synthetic Aperture Sonar Imagery Using Deep Convolutional Neural networks”. In : *2016 23rd International Conference on Pattern Recognition (ICPR)*. Déc. 2016, p. 2497-2502 (cf. p. 46).

[31] N. Dalal, B. Triggs et C. Schmid. “Human detection using oriented histograms of flow and appearance”. In : *European conference on computer vision*. Springer, 2006, p. 428-441 (cf. p. 14).

[32] J. Deng et al. “ImageNet : A Large-Scale Hierarchical Image Database”. In : *CVPR09*. 2009 (cf. p. xx, 16, 31, 66).

[33] T. DeVries et G. W. Taylor. “Learning Confidence for Out-of-Distribution Detection in Neural Networks”. In : *arXiv e-prints* (fév. 2018), arXiv :1802.04865 (cf. p. 94).

[34] J. Duchi, E. Hazan et Y. Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In : *J. Mach. Learn. Res.* 12.null (juill. 2011), p. 2121-2159 (cf. p. 28).

[35] Farzana Khan, J. et Alam, M. S. “Target detection in cluttered forward-looking infrared imagery”. In : *Optical Engineering* 44.7 (juill. 2005), p. 1-8 (cf. p. 46).

[36] *FLIR Thermal Dataset for Algorithm Training*. 2018 (cf. p. 42).

[37] G. Cheng et al. “When Deep Learning Meets Metric Learning : Remote Sensing Image Scene Classification via Learning Discriminative CNNs”. In : *IEEE Transactions on Geoscience and Remote Sensing* 56.5 (mai 2018), p. 2811-2821 (cf. p. 62, 63).

[38] G. Huang et al. “Densely Connected Convolutional Networks”. In : *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Juill. 2017, p. 2261-2269 (cf. p. 17).

[39] G. Kollios et al. “Efficient biased sampling for approximate clustering and outlier detection in large data sets”. In : *IEEE Transactions on Knowledge and Data Engineering* 15.5 (oct. 2003), p. 1170-1187 (cf. p. 93).

[40] G. Wang et al. “Interactive Medical Image Segmentation Using Deep Learning With Image-Specific Fine Tuning”. In : *IEEE Transactions on Medical Imaging* 37.7 (juill. 2018), p. 1562-1573 (cf. p. 31).

[41] L. Garrido et al. “A Regularization Term to Avoid The Saturation of The Sigmoids in Multilayer Neural Networks”. In : *Int. J. Neur. Syst.* 07.03 (juill. 1996), p. 257-262 (cf. p. 25).

[42] I. Goodfellow, Y. Bengio et A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016 (cf. p. 6, 12, 19, 21, 28-30, 93).

[43] I. J. Goodfellow, J. Shlens et C. Szegedy. “Explaining and Harnessing Adversarial Examples”. In : *3rd International Conference on Learning Representations, ICLR*. 2015, arXiv :1412.6572 (cf. p. 34, 92).

[44] T. Gu, B. Dolan-Gavitt et S. Garg. “BadNets : Identifying Vulnerabilities in the Machine Learning Model Supply Chain”. In : *arXiv e-prints* (août 2017), arXiv :1708.06733 (cf. p. 58).

[45] H. Deng et al. “Infrared Small-Target Detection Using Multiscale Gray Difference Weighted Image Entropy”. In : *IEEE Transactions on Aerospace and Electronic Systems* 52.1 (fév. 2016), p. 60-72 (cf. p. 46).

[46] H. O. Song et al. “Deep Metric Learning via Lifted Structured Feature Embedding”. In : *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Juin 2016, p. 4004-4012 (cf. p. 72).

[47] B. Hanin. “Which Neural Net Architectures Give Rise to Exploding and Vanishing Gradients ?” In : *Advances in Neural Information Processing Systems 31*. Sous la dir. de S. Bengio et al. Curran Associates, Inc., 2018, p. 582-591 (cf. p. 26).

[48] K. He et al. “Deep Residual Learning for Image Recognition”. In : *CoRR* abs/1512.03385 (2015) (cf. p. 17, 66).

[49] K. He et al. “Delving Deep into Rectifiers : Surpassing Human-Level Performance on ImageNet Classification”. In : *CoRR* abs/1502.01852 (2015). arXiv : 1502.01852 (cf. p. 27).

[50] D. Hendrycks et K. Gimpel. “A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks”. In : *arXiv e-prints* (oct. 2016), arXiv :1610.02136 (cf. p. 93).

[51] D. Hendrycks et K. Gimpel. “A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks.” In : *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017 (cf. p. 90).

[52] S. Hochreiter. “The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions”. In : *Int. J. Unc. Fuzz. Knowl. Based Syst.* 06.02 (avr. 1998), p. 107-116 (cf. p. 26).

[53] K. Hornik. “Approximation capabilities of multilayer feedforward networks”. In : *Neural Networks* 4.2 (jan. 1991), p. 251-257 (cf. p. 5).

[54] Z. Huang, Z. Pan et B. Lei. “Transfer Learning with Deep Convolutional Neural Network for SAR Target Classification with Limited Labeled Data”. In : *Remote. Sens.* 9 (2017), p. 907 (cf. p. 61).

[55] A. Ilyas et al. “Adversarial Examples Are Not Bugs, They Are Features”. In : *Advances in Neural Information Processing Systems 32*. Sous la dir. de H. Wallach et al. Curran Associates, Inc., 2019, p. 125-136 (cf. p. 34).

[56] S. Ioffe et C. Szegedy. “Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In : *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. ICML’15. Lille, France : JMLR.org, 2015, p. 448-456 (cf. p. 17).

[57] J. Li, C. Qu et J. Shao. “Ship Detection In SAR Images Based on an Improved Faster R-CNN”. In : *2017 SAR in Big Data Era : Models, Methods and Applications (BIGSARDATA)*. Nov. 2017, p. 1-6 (cf. p. 31).

[58] J. Tobin et al. “Domain randomization for transferring deep neural networks from simulation to the real world”. In : *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, p. 23-30 (cf. p. 60, 61).

[59] M. Jaderberg et al. “Spatial Transformer Networks”. In : *Advances in Neural Information Processing Systems 28*. Sous la dir. de C. Cortes et al. Curran Associates, Inc., 2015, p. 2017-2025 (cf. p. 63).

[60] James A. Ratches. “Review of Current Aided/Automatic Target Acquisition Technology for Military Target Acquisition Tasks”. In : *Optical Engineering* 50.7 (mars 2011), p. 1-8 (cf. p. 53).

[61] K. Eykholt et al. “Robust Physical-World Attacks on Deep Learning Visual Classification”. In : *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Juin 2018, p. 1625-1634 (cf. p. 59).

[62] K. He et al. “Delving Deep into Rectifiers : Surpassing Human-Level Performance on ImageNet Classification”. In : *2015 IEEE International Conference on Computer Vision (ICCV)*. Déc. 2015, p. 1026-1034 (cf. p. 26, 28).

[63] K. Zhang, X. Kang et S. Li. “Isolation Forest for Anomaly Detection in Hyperspectral Images”. In : *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*. Août 2019, p. 437-440 (cf. p. 93).

[64] M. Kang et al. “Synthetic Aperture Radar Target Recognition with Feature Fusion Based on a Stacked Autoencoder”. eng. In : *Sensors (Basel)* 17.1 (jan. 2017), p. 192 (cf. p. 61).

[65] J. F. Khan, M. S. Alam et S. M. A. Bhuiyan. “Automatic target detection in forward-looking infrared imagery via probabilistic neural networks”. In : *Appl. Opt.* 48.3 (jan. 2009), p. 464-476 (cf. p. 46).

[66] S. Kim, W. Song et S. Kim. “Infrared Variation Optimized Deep Convolutional Neural Network for Robust Automatic Ground Target Recognition”. In : *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2017, p. 195-202 (cf. p. 46).

[67] D. P. Kingma et J. Ba. “Adam : A Method for Stochastic Optimization”. In : *arXiv e-prints* (déc. 2014), arXiv :1412.6980 (cf. p. 29).

[68] Z. Kolter et A. Madry. “Adversarial Robustness : Theory and Practice”. In : *Tutorial at NeurIPS 2018*. Montreal, déc. 2018 (cf. p. 92).

[69] A. Krizhevsky, I. Sutskever et G. E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In : *Advances in Neural Information Processing Systems 25*. Sous la dir. de F. Pereira et al. Curran Associates, Inc., 2012, p. 1097-1105 (cf. p. 16, 26).

[70] Kun-Lun Li et al. “Improving one-class SVM for anomaly detection”. In : *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693)*. T. 5. Nov. 2003, 3077-3081 Vol.5 (cf. p. 93).

[71] A. Kurakin, I. Goodfellow et S. Bengio. “Adversarial examples in the physical world”. In : *arXiv e-prints* (juill. 2016), arXiv :1607.02533 (cf. p. 59).

[72] A. Kurakin, I. J. Goodfellow et S. Bengio. “Adversarial Machine Learning at Scale.” In : *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017 (cf. p. 92).

[73] L. Bazzani et al. “Self-taught object localization with deep networks”. In : *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. Mars 2016, p. 1-9 (cf. p. 63).

[74] Y. LeCun et al. “Gradient-Based Learning Applied to Document Recognition”. In : *Proceedings of the IEEE* 86.11 (nov. 1998), p. 2278-2324 (cf. p. 15, 16).

[75] M. Lecuyer et al. “Certified Robustness to Adversarial Examples with Differential Privacy”. In : *arXiv e-prints* (fév. 2018), arXiv :1802.03471 (cf. p. 92).

[76] K. Lee et al. “A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks”. In : *Advances in Neural Information Processing Systems 31*. Sous la dir. de S. Bengio et al. Curran Associates, Inc., 2018, p. 7167-7177 (cf. p. 93, 94, 98, 99).

[77] M. Leshno et al. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In : *Neural Networks* 6.6 (jan. 1993), p. 861-867 (cf. p. 6).

[78] S. Liang, Y. Li et R. Srikant. “Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks”. In : *arXiv e-prints* (juin 2017), arXiv :1706.02690 (cf. p. 93, 94).

[79] M. Lin, Q. Chen et S. Yan. “Network In Network”. In : *arXiv e-prints* (déc. 2013), arXiv :1312.4400 (cf. p. 64).

[80] X. Liu et al. “Towards Robust Neural Networks via Random Self-ensemble”. In : *The European Conference on Computer Vision (ECCV)*. 2018 (cf. p. 92).

[81] Y. Liu et al. “Delving into Transferable Adversarial Examples and Black-box Attacks”. In : *arXiv e-prints* (nov. 2016), arXiv :1611.02770 (cf. p. 34).

[82] D. G. Lowe. “Object Recognition from Local Scale-Invariant Features”. In : *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*. ICCV ’99. USA : IEEE Computer Society, 1999, p. 1150 (cf. p. 14).

[83] L. Lu et al. *Dying ReLU and Initialization : Theory and Numerical Examples*. 2019. arXiv : 1903.06733 [stat.ML] (cf. p. 26).

[84] M. Çelik, F. Dadaşer-Çelik et A. Ş. Dokuz. “Anomaly Detection in Temperature Data Using DBSCAN algorithm”. In : *2011 International Symposium on Innovations in Intelligent Systems and Applications*. Juin 2011, p. 91-95 (cf. p. 93).

[85] M. Oquab et al. “Is object localization for free ? - Weakly-supervised learning with convolutional neural networks”. In : *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Juin 2015, p. 685-694 (cf. p. 63).

[86] M. Teutsch et W. Krüger. “Classification of Small Boats in Infrared Images for Maritime Surveillance”. In : *2010 International WaterSide Security Conference*. Nov. 2010, p. 1-7 (cf. p. 46).

[87] A. L. Maas, A. Y. Hannun et A. Y. Ng. “Rectifier nonlinearities improve neural network acoustic models”. In : *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013 (cf. p. 26).

[88] L. v. d. Maaten et G. Hinton. “Visualizing data using t-SNE”. In : *Journal of machine learning research* 9.Nov (2008), p. 2579-2605 (cf. p. 96).

[89] A. Mandelbaum et D. Weinshall. “Distance-based Confidence Score for Neural Network Classifiers”. In : *arXiv e-prints* (sept. 2017), arXiv :1709.09844 (cf. p. 93).

[90] *Military Sensing Information Analysis Center (SENSIAC) : Dataset for Automatic Target Recognition in Infrared Imagery*. 2008 (cf. p. 42, 48, 101).

[91] M. Minsky et S. Papert. *Perceptrons*. MIT Press, 1969 (cf. p. 2).

[92] N. Tajbakhsh et al. “Convolutional Neural Networks for Medical Image Analysis : Full Training or Fine Tuning ?” In : *IEEE Transactions on Medical Imaging* 35.5 (mai 2016), p. 1299-1312 (cf. p. 31).

[93] V. Nair et G. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In : *ICML Conference 2010* (2010) (cf. p. 25).

[94] H.-W. Ng et al. “Deep Learning for Emotion Recognition on Small Datasets Using Transfer Learning”. In : *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*. ICMI ’15. event-place : Seattle, Washington, USA. New York, NY, USA : Association for Computing Machinery, 2015, p. 443-449 (cf. p. 31).

[95] P. Tzannes, A. et Brooks, D. H. “Point target detection in IR image sequences based on target and clutter temporal profile modeling”. In : *Infrared Technology and Applications XXV*. T. 3698. Juill. 1999 (cf. p. 46).

[96] N. Papernot, P. McDaniel et I. Goodfellow. “Transferability in Machine Learning : from Phenomena to Black-Box Attacks using Adversarial Samples”. In : *arXiv e-prints* (mai 2016), arXiv :1605.07277 (cf. p. 106).

[97] N. Papernot et al. “Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks”. In : *arXiv e-prints* (nov. 2015), arXiv :1511.04508 (cf. p. 92).

[98] N. Papernot et al. “Practical Black-Box Attacks against Machine Learning”. In : *arXiv e-prints* (fév. 2016), arXiv :1602.02697 (cf. p. 106).

[99] N. Papernot et al. “Practical black-box attacks against machine learning”. In : *2017 ACM Asia Conference on Computer and Communications Security, ASIA CCS 2017*. ASIA CCS 2017 - Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security (avr. 2017), p. 506-519 (cf. p. 34, 92).

[100] R. Hadsell, S. Chopra et Y. LeCun. “Dimensionality Reduction by Learning an Invariant Mapping”. In : *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’06)*. T. 2. Juin 2006, p. 1735-1742 (cf. p. 62).

[101] I. Rodger, B. Connor et N. M. Robertson. “Classifying objects in LWIR imagery via CNNs”. In : *Electro-Optical and Infrared Systems : Technology and Applications XIII*. T. 9987. Oct. 2016 (cf. p. 46, 64, 66, 68, 90).

[102] S. Chen et al. “Target Classification Using the Deep Convolutional Networks for SAR Images”. In : *IEEE Transactions on Geoscience and Remote Sensing* 54.8 (août 2016), p. 4806-4817 (cf. p. 46).

[103] S. Dodge et L. Karam. “Understanding how image quality affects deep neural networks”. In : *2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX)*. Juin 2016, p. 1-6 (cf. p. 90).

[104] S. Moosavi-Dezfooli et al. “Universal Adversarial Perturbations”. In : *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Juill. 2017, p. 86-94 (cf. p. 34).

[105] S. Zheng et al. “Improving the Robustness of Deep Neural Networks via Stability Training”. In : *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Juin 2016, p. 4480-4488 (cf. p. 92).

[106] B. Schölkopf et al. “Support Vector Method for Novelty Detection”. In : *Proceedings of the 12th International Conference on Neural Information Processing Systems*. NIPS 99. event-place : Denver, CO. Cambridge, MA, USA : MIT Press, 1999, p. 582-588 (cf. p. 93).

[107] A. Shafahi et al. “Adversarial training for free !” In : *Advances in Neural Information Processing Systems 32*. Sous la dir. de H. Wallach et al. Curran Associates, Inc., 2019, p. 3358-3369 (cf. p. 92).

[108] K. Simonyan et A. Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In : *arXiv e-prints* (sept. 2014), arXiv :1409.1556 (cf. p. 17, 66).

[109] J. T. Springenberg et al. “Striving for Simplicity : The All Convolutional Net”. In : *CoRR* arXiv :1412.6806 (2014) (cf. p. 20, 21, 68).

[110] N. Srivastava et al. “Dropout : A Simple Way to Prevent Neural Networks from Overfitting”. In : *Journal of Machine Learning Research* 15 (2014), p. 1929-1958 (cf. p. 23, 24, 70).

[111] C. Szegedy, S. Ioffe et V. Vanhoucke. “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning”. In : *CoRR* abs/1602.07261 (2016) (cf. p. 17).

[112] C. Szegedy et al. “Going Deeper with Convolutions”. In : *CoRR* arXiv :1409.4842 (2014) (cf. p. 17).

[113] C. Szegedy et al. “Intriguing properties of neural networks”. In : *arXiv e-prints* (déc. 2013), arXiv :1312.6199 (cf. p. 34).

[114] C. Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In : *arXiv e-prints* (déc. 2015), arXiv :1512.00567 (cf. p. 17, 66).

[115] T. Lin, A. RoyChowdhury et S. Maji. “Bilinear CNN Models for Fine-Grained Visual Recognition”. In : *2015 IEEE International Conference on Computer Vision (ICCV)*. Déc. 2015, p. 1449-1457 (cf. p. 66).

[116] R. Tibshirani. “Regression Shrinkage and Selection via the Lasso”. In : *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1 (1996), p. 267-288 (cf. p. 22).

[117] T. Tieleman et G. Hinton. “Lecture 6.5-rmsprop : Divide the gradient by a running average of its recent magnitude”. In : *COURSERA : Neural networks for machine learning* 4.2 (2012), p. 26-31 (cf. p. 28).

[118] F. Tramer et D. Boneh. “Adversarial Training and Robustness for Multiple Perturbations”. In : *Advances in Neural Information Processing Systems 32*. Sous la dir. de H. Wallach et al. Curran Associates, Inc., 2019, p. 5866-5876 (cf. p. 92).

[119] V. Venkataraman et al. “Automated Target Tracking and Recognition Using Coupled View and Identity Manifolds for Shape Representation”. In : *EURASIP Journal on Advances in Signal Processing* 2011.1 (déc. 2011), p. 124 (cf. p. 46, 73, 74).

[120] J. E. Vitela et J. Reifman. “Premature Saturation in Backpropagation Networks : Mechanism and Necessary Conditions”. In : *Neural Networks* 10.4 (juin 1997), p. 721-735 (cf. p. 25).

[121] S. Wager, S. Wang et P. S. Liang. “Dropout Training as Adaptive Regularization”. In : *Advances in Neural Information Processing Systems 26*. Sous la dir. de C. J. C. Burges et al. Curran Associates, Inc., 2013, p. 351-359 (cf. p. 24).

[122] B. Wang et al. “With Great Training Comes Great Vulnerability : Practical Attacks against Transfer Learning”. In : *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD : USENIX Association, août 2018, p. 1281-1297 (cf. p. 57).

[123] K. Q. Weinberger et L. K. Saul. “Distance Metric Learning for Large Margin Nearest Neighbor Classification”. In : *J. Mach. Learn. Res.* 10 (juin 2009), p. 207-244 (cf. p. 62).

[124] Xavier Glorot, Antoine Bordes et Yoshua Bengio. “Deep Sparse Rectifier Neural Networks”. In : *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Sous la dir. de Geoffrey Gordon, David Dunson et Miroslav Dudík. PMLR, juin 2011, p. 315-323 (cf. p. 26).

[125] Xavier Glorot et Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In : sous la dir. d’Yee Whye Teh et Mike Titterington. PMLR, mars 2010, p. 249-256 (cf. p. 28).

[126] B. Xu et al. “Empirical Evaluation of Rectified Activations in Convolutional Network”. In : *CoRR* abs/1505.00853 (2015). arXiv : 1505.00853 (cf. p. 27).

[127] H. Xu et al. “Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications”. In : *arXiv e-prints* (fév. 2018), arXiv :1802.03903 (cf. p. 93).

[128] Y. LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In : *Neural Computation* 1.4 (déc. 1989), p. 541-551 (cf. p. 7).

[129] Y. Lee, S. -. Oh et M. W. Kim. “The effect of initial weights on premature saturation in back-propagation learning”. In : *IJCNN-91-Seattle International Joint Conference on Neural Networks*. T. i. Juill. 1991, 765-770 vol.1 (cf. p. 25).

[130] Yaroslav Ganin et Victor Lempitsky. “Unsupervised Domain Adaptation by Backpropagation”. In : *Proceedings of the 32nd International Conference on Machine Learning*. Sous la dir. de Francis Bach et David Blei. PMLR, juin 2015, p. 1180-1189 (cf. p. 68, 119).

[131] S. P. Yoon, T. L. Song et T. H. Kim. “Automatic target recognition and tracking in forward-looking infrared image sequences with a complex background”. In : *International Journal of Control, Automation and Systems* 11.1 (fév. 2013), p. 21-32 (cf. p. 46).

[132] J. Yosinski et al. “How transferable are features in deep neural networks ?” In : *Advances in Neural Information Processing Systems 27*. Sous la dir. de Z. Ghahramani et al. Curran Associates, Inc., 2014, p. 3320-3328 (cf. p. 31).

[133] Z. Zhou et al. “Fine-Tuning Convolutional Neural Networks for Biomedical Image Analysis : Actively and Incrementally”. In : *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Juill. 2017, p. 4761-4772 (cf. p. 31).

[134] M. D. Zeiler. “ADADELTA : An Adaptive Learning Rate Method”. In : *arXiv e-prints* (déc. 2012), arXiv :1212.5701 (cf. p. 28).

[135] M. D. Zeiler et R. Fergus. “Visualizing and Understanding Convolutional Networks”. In : *Computer Vision – ECCV 2014*. Sous la dir. de D. Fleet et al. Springer International Publishing, 2014, p. 818-833 (cf. p. 63).

[136] Q.-s. Zhang et S.-c. Zhu. “Visual interpretability for deep learning : a survey”. In : *Frontiers of Information Technology & Electronic Engineering* 19.1 (jan. 2018), p. 27-39 (cf. p. 54).

DOCTORAT BRETAGNE LOIRE / MATHSTIC

ENSTA Bretagne

Titre : Réseaux de neurones profonds pour la classification d'objets en imagerie infrarouge : apports de l'apprentissage à partir de données synthétiques et de la détection d'anomalies

Mots-clés : Apprentissage Profond, Infrarouge, Transfert de domaine, Détection d'anomalie

Résumé :

Les performances des technique d'apprentissage profond et plus particulièrement les réseaux de neurones convolutifs, ou CNN, sont conditionnés par la taille et la qualité des bases de données d'entrainement. Dans un contexte comme celui de l'identification de véhicules militaires dans des images infrarouges, il est difficile de constituer de telles bases d'apprentissage. Pour y remédier, il est possible d'utiliser la simulation pour générer ces ensembles de données. Cependant, les architectures issues de l'état de l'art généralisent mal sur des données réelles après un entrainement sur données simulées. Dans cette thèse, nous proposons un réseau convolutif spécifique, appelée cfCNN, qui permet d'obtenir de meilleures performances que les modèles de l'état de l'art que nous avons testés. Nous supposons que les images qui lui seront présentées seront issues d'un module de détection qui peut être imparfaite. Nous évaluons donc la robustesse du cfCNN face à des translations et des changements d'échelle de la cible dans l'image d'entrée. Face à ces perturbations, le cfCNN montre une meilleure robustesse par rapport à des réseaux convolutifs issus de l'état de l'art. Pour améliorer la confiance dans les prédictions du cfCNN, nous proposons un module de détection d'anomalies de classification, basé sur le Local Outlier Factor. Cette approche montre de bonnes performances sur des données d'entrainement et de test homogènes. Cependant elle est moins performante pour un problème de transfert entre des données simulées et réelles. Pour compenser cette baisse de performance nous proposons alors un schéma en cascade qui exploite les informations issues de différentes couches du cfCNN.

Title: Deep neural networks for object classification in infrared imagery : contribution of learning from synthetic data and anomaly detection.

Keywords: Deep-Learning, Infrared, Domain Transfert, Anomaly Detection

Abstract:

Neural network performance is directly linked to the quality and quantity of training data available, especially for Convolutional Neural Networks. Building exhaustive datasets is difficult for automatic target recognition and identification in a military context. To compensate data scarcity, it is possible to use simulation to create a dataset large enough to train neural networks. However, state of the art CNN models may not be able to properly generalize on real data after a training phase on simulated images. In this thesis we propose a purpose-built CNN called cfCNN which is able improve over state of the art models. We assume that the cfCNN is processing images from a detection stage which may not be fully accurate. As a result, we test our model against various detections errors and show that it outperforms various CNN models from the literature. Confidence in a model's predictions can substantially help its deployment, especially for military applications. In this work, we propose an anomaly detection module to detect classification errors during inference. Based on the Local Outlier Factor algorithm, it showed great results on homogenous training and testing datasets. However, it lacked performance when transferring the detector from simulated to real data. To compensate the performance loss in this scenario, we propose a cascade detector which uses multiple inputs from the cfCNN to detect classification errors.