Apprentissage de représentation différenciées dans des modèles d'apprentissage profond : détection de classes inconnues et interprétabilité

Quentin Christoffel

UNIVERSITÉ DE STRASBOURG

ÉCOLE DOCTORALE MATHEMATIQUES, SCIENCES DE L'INFORMATION ET DE L'INGENIEUR

Équipe « Complex Systems & Translational Bioinformatics »
— Laboratoire ICube — UMR 7357

THÈSE **présentée par / DISSERTATION presented by :**

Quentin CHRISTOFFEL

soutenue le /defended on : **15 novembre 2024**

pour obtenir le grade de **/** to obtain the grade of **:**
Docteur de l'Université de Strasbourg / Strasbourg University Doctor

Discipline/Spécialité **/** Discipline/Specialty : Informatique

Apprentissage de représentations différenciées dans des modèles d'apprentissage profond : détection de classes inconnues et interprétabilité

THÈSE dirigée par / DISSERTATION supervisor :
Dr DERUYVER Aline MCF, HDR, Université de Strasbourg

RAPPORTEURS :
Pr STEFFENEL Luiz Angelo PR, Université de Reims Champagne-Ardenne
Pr DE LOOR Pierre PR, École Nationale d'Ingénieurs de Brest

AUTRES MEMBRES DU JURY / OTHER MEMBERS OF THE JURY :
Pr FRYDMAN Claudia PR, Aix-Marseille Université
Pr LAMPERT Thomas PR, Telecom Physique Strasbourg, Université de Strasbourg

---

INVITÉS (le cas échéant) / INVITED MEMBERS (if applicable) :
Dr JEANNIN-GIRARDON Anne MCF, HDR, Université de Strasbourg
Dr AYADI Ali MCF, Université de Strasbourg

Remerciements

Une aventure s'achève, qui avait déjà commencé bien avant la thèse, depuis mes premiers stages dans l'équipe CSTB. Anne, je dois d'abord te remercier pour la confiance que tu m'as accordée toutes ces années, pour les sujets que tu m'as proposés qui m'ont lancé dans l'Intelligence Artificielle et mené jusqu'ici, puisque c'est toi qui m'as encouragé à faire une thèse. Je te remercie aussi pour la liberté que tu m'as laissée d'explorer d'autres sujets, ce qui nous a finalement menés à DIFAIR, et pour tous les échanges que nous avons pu avoir durant ces six années.

Je remercie bien sûr chaleureusement mon jury, Pr Luiz Angelo Steffenel, Pr Pierre De Loor, Pr Claudia Frydman et Pr Thomas Lampert, pour avoir accepté d'examiner mon travail et de relire ce manuscrit. Je vous remercie particulièrement pour la pertinence et la bienveillance des commentaires que j'ai reçus, et pour toutes les idées et la curiosité que vous avez manifestées envers mes travaux.

Bien entendu, tout cela n'aurait pas eu lieu sans Aline, Ali et Anne. Je vous remercie, Aline pour avoir accepté de diriger cette thèse, Anne pour avoir proposé le sujet et pour toutes les raisons que j'ai déjà listées, et Ali, qui est arrivé un peu plus tard dans le projet, pour tout le temps que tu as passé à m'aider, à relire mes écrits, pour tous nos échanges et ton soutien. Tu étais toujours prêt à prendre du temps pour moi, tout comme Anne, même lorsque vous étiez déjà surchargés.

Durant ma thèse, j'ai rencontré tellement de monde. Je vous remercie tous énormément pour tous ces moments passés ensemble. Je suis arrivé en stage il y a 6 ans, timide et renfermé, je repars bien plus confiant et tellement bien entouré. La liste d'invitations pour ma thèse m'a permis de vraiment en

prendre conscience et cette liste serait trop longue ici, mais pour raccourcir : merci à toute l'équipe CSTB, actuelle et passée, aux anciens stagiaires qui se reconnaîtront. Je n'oublierai jamais tous ces moments passés ensemble, tous les pots, les fêtes, les départs, en un mot : toutes ces émotions. En particulier, je te remercie, Romain, pour toutes les discussions qu'on a pu avoir pendant mes stages et parfois pendant la thèse, en m'encadrant et me guidant.

Enfin, je remercie évidemment ma famille pour m'avoir accompagné et soutenu toutes ces années d'études qui prennent finalement fin. Je remercie aussi mes amis, que j'ai depuis toujours et ceux que j'ai rencontré en dehors du contexte de la thèse. Et forcément, Fred, ces 5 dernières années auraient été bien différentes sans toi. Tu as contribué énormément aux changements qui me sont arrivés depuis qu'on s'est rencontrés, et je suis heureux de la personne que je suis devenu à tes côtés. Pour tout ça, et l'aide, le soutiens et tout ce que tu as pu m'apporter pendant ma thèse, je te remercie.

En relisant ces lignes, je me rends compte que je ne réussis pas à exprimer à quel point je suis reconnaissant envers tous ceux avec qui j'ai pu travailler ou échanger. C'est pourquoi je remercie encore une fois tous ceux que j'ai rencontrés pendant ces années. Cette aventure, vécue à vos côtés, est une période de ma vie que je ne pourrai jamais oublier, et j'espère bien continuer à pouvoir échanger et voir un maximum d'entre vous.

Table des matières

Table des figures

Liste des tableaux

Liste des acronymes

ARPL *Adversarial Reciprocal Points Learning* 63
AUROC *Area Under the Receiver Operating Characteristic curve*
C2AE *Class Conditioned Auto-Encoder* 58
CAC *Class Anchor Clustering* 72
CELR *Cross-Entropy Learned Representation* 105
CNN *Convolutional Neural Network* 15
CROSR *Classification Reconstruction learning for OSR* 55
ConOSR *Contrastive OSR* 67
DCHS *Deep Compact HyperSphere* 69
DHRNet *Deep Hierarchical Reconstruction Net* 56
DIFAIR *DIFferentiAted Images Representations* 4
ECS Exactitude en *Closed-Set* 42
EVT *Extreme Value Theory* 36
FN faux négatif 41
FP faux positif 41
FiLM *Feature-wise Linear Modulation* 58
GAN *Generative Adversarial Network* 52
IA Intelligence Artificielle 1
MLP *Multi-Layer Perceptron* 10
MLS *Maximum Logit Score* 66
MOS *Maximum Output Score* 105
MSP *Maximum Softmax Probability* 34
OSR *Open-Set Recognition* 3
OSRCI *OSR with Counterfactual Images* 53

PCA *Principal Component Analysis* 95
PH Pourcentage dans les Hypersphères
ROC *Receiver Operating Characteristic* 42
ReLU *Rectified Linear Unit* 13
ResNet *Residual Network*
SGD *Stochastic Gradient Descent* 29
SNE *Stochastic Neighbor Embedding*
SVHN *Street View House Numbers*
VGG *Visual Geometry Group*
VN vrai négatif 41
VP vrai positif 41
t-SNE *t-distributed Stochastic Neighbor Embedding* 95

Introduction

L'utilisation de systèmes d'Intelligence Artificielle (IA) est de plus en plus répandue dans notre quotidien : assistants vocaux (Google, 2024), saisie prédictive sur nos téléphones ou ordinateurs (BBC, 2019), systèmes de recommandation (Schrage, 2020), générateurs de contenus textuels tels que ChatGPT (OpenAI, 2024), ou d'images, tels que Midjourney (MidJourney, 2024), ou encore comme instruments dans différentes pratiques scientifiques (Jumper et al., 2021). Ces systèmes sont de plus en plus utilisés par des personnes non-expertes en IA, y compris par le grand public : cela est rendu possible par l'amélioration significative de leur accessibilité et par leur incorporation dans toute sorte de services que nous utilisons quotidiennement, sans nécessairement réaliser qu'un système d'IA s'y loge. Il n'est cependant pas évident que nous ayons pleinement conscience des défis et problèmes sous-jacents à ces différentes IA. Si ces systèmes peuvent apporter un support important dans la réalisation de diverses tâches, ils sont aussi utilisés dans des domaines que l'on peut qualifier de critiques, comme la santé, la justice, la finance et le transport, y compris la conduite autonome. En réalité, il est difficile de trouver un domaine qui ne fasse pas usage de systèmes d'IA à un niveau ou à un autre.

Dans le domaine de la santé, par exemple, l'un des attraits des systèmes d'IA est l'aide au diagnostic par l'analyse automatique d'images médicales (Liu et al., 2020) ; d'autres systèmes peuvent assister les médecins lors d'interventions chirurgicales (Knudsen et al., 2024). Dans le cas de la conduite autonome, un système d'IA analyse l'environnement du véhicule en détectant la chaussée et les objets environnant tels que les autres usagers (véhicules, piétons, cyclistes, ...) ou encore la signalisation (Yurtsever et al., 2020). Dans ces deux domaines, ainsi que dans de nombreux autres, des sys-

tèmes d'IA sont largement utilisés comme moyen d'analyser des images : les approches basées sur les réseaux de neurones convolutifs (LeCun et al., 1998), par exemple, ont depuis longtemps prouvé leur efficacité pour traiter ce type de données.

Pour introduire la problématique de notre travail, prenons l'exemple d'un système d'IA détectant des objets dans le cadre de la conduite autonome de véhicules. Un tel système devra être capable de détecter différents éléments. Il sera donc nécessaire de disposer de données dites étiquetées (c'est-à-dire dont l'objet présent dans l'image est identifié), représentatives de ces différents objets, afin que le modèle apprenne à les caractériser pour les reconnaître en situation réelle. En mars 2018 à Tempe, en Arizona, alors que Elaine Herzberg pousse son vélo sur une route, un véhicule autonome la renverse fatalement ; véhicule qui n'a jamais freiné face à l'obstacle et c'est la conductrice qui a finalement freiné 0.7 seconde après l'impact (NTSB, 2019; Macrae, 2022). L'enquête du *National Transportation Safety Board* des États-Unis a révélé, entre autres éléments, que le système n'est parvenu ni à détecter Mme Herzberg comme piétonne, réassignant sa classification entre « véhicule », « vélo » et « autre », ni à anticiper sa trajectoire probable. Même si le système avait disposé d'une procédure lui permettant de gérer les objets classés « autres » (ce qui n'était pas le cas), cet accident dramatique soulève plus largement un point critique dans le domaine de l'apprentissage automatique supervisé : comment faire face à une situation dans laquelle une image analysée contient un objet inconnu, un objet que le modèle n'a pas appris à reconnaître lors de son entraînement ?

Une donnée passée à un modèle donnera nécessairement lieu à l'assignation d'une classe connue par ce dernier. Généralement, les modèles sont entraînés et leurs performances évaluées sur un ensemble de classes donné et fixe : ce scénario de classification est nommé classification en *closed-set* (Scheirer et al., 2012). Mais développer et entraîner un modèle sert certainement une application dans un environnement réel, dynamique, où les types d'objets peuvent évoluer et où de nouveaux types d'objets peuvent apparaître. Ainsi, il est nécessaire de développer des modèles robustes face à ces situations, pour que ceux-ci soient *a minima* en mesure de détecter qu'ils font face à une donnée de type inconnu, ou plutôt, comme nous allons le voir, de *détecter l'absence d'objets connus*.

En outre, l'utilisation de systèmes d'IA dans des situations critiques soulève également la question de l'interprétabilité des résultats des modèles. Depuis plusieurs années, une sous-branche du domaine est consacrée au développement d'approches algorithmiques pour tenter de répondre à

cette problématique : le XAI (*eXplainable Artificial Intelligence*) (Arrieta et al., 2020). Derrière le néologisme *explicabilité* se trouve un ensemble de techniques visant à faciliter l'interprétabilité de différents types de modèles et de leurs résultats, et pour divers types de données (images, textes, *etc.*). Nous sommes donc face à deux grands verrous scientifiques : d'une part, celui de concevoir des modèles plus robustes à la variabilité des environnements réels ; d'autre part, celui de permettre une meilleure compréhension ou interprétation des résultats des modèles, voire des modèles eux-mêmes. Bien que les fondements derrières les modèles d'apprentissage automatique ne soient pas nécessairement *compliqués* (car un modèle de type réseau de neurones n'est formellement représenté que par une composition de fonctions), le fonctionnement intrinsèque de ces modèles n'en est pas moins *complexe*. En effet, pour déterminer la nature des objets présents dans une donnée qui lui est passée en entrée (dans le cadre d'un problème de classification supervisée), le modèle réalise de multiples combinaisons non-linéaires de centaines, de milliers, de millions, voire de milliards de paramètres [1].

Le concept d'*Open-Set Recognition* (OSR), introduit par Scheirer et al. (2012), est un paradigme d'apprentissage automatique dans lequel les modèles sont évalués sur un jeu de données contenant à la fois des données dont la classe est connue et des données dont la classe est inconnue, c'est-à-dire des données d'un type jamais vu par les modèles lors de leur entraînement. L'évaluation d'un modèle en OSR représente donc un scénario plus représentatif d'une situation réelle. L'objectif d'un modèle entraîné en contexte d'OSR est donc double : détecter des données inconnues d'une part et, d'autre part, être capable de classer correctement les données de types connus.

Nos travaux se situent dans le domaine de l'OSR. Nous nous sommes en particulier intéressés aux réseaux de neurones convolutifs, qui apprennent à extraire des *attributs* des images d'entraînement qui leur sont fournies. Ces architectures, dont le fonctionnement est bien compris, présentent l'avantage d'être bien éprouvées sur des tâches de classification d'images. Après apprentissage, les attributs extraits, qui doivent être suffisamment généraux, sont utilisés pour prédire la classe d'une image : si le modèle détecte des attributs x, y et z dans cette image, que ces attributs sont représentatifs de la classe C, et que les attributs d'autres classes n'ont pas été détectés, alors le modèle conclut que l'image contient un objet de type C. Pour chaque classe présente dans le jeu de données d'entraînement, l'ensemble des at-

1. À l'instar de GPT-3, le modèle utilisé dans le système ChatGPT en date de juin 2023, qui contient 175 milliards de paramètres

tributs pouvant être extraits constitue un vecteur de représentation de la classe. Ce vecteur peut être considéré comme un vecteur de coordonnées dans un *espace de représentation* appris par le modèle.

À l'origine de nos travaux se trouve alors la question suivante : **dans le contexte de la classification d'images, dans quelle mesure pouvons-nous exploiter la représentation apprise par le modèle (appelée également *représentation latente*) pour détecter que le modèle fait face à des données dont la classe lui est inconnue (c'est-à-dire des classes non représentées dans les données ayant servi à l'entraîner), afin d'éviter que le modèle réalise une prédiction erronée ?**

Pour répondre à cette question, nous proposons une approche nommée *DIFferentiAted Images Representations* (DIFAIR), dont l'objectif est d'apprendre une représentation spécifique des données d'entraînement qui devra permettre la détection de classes inconnues. DIFAIR est une approche applicable sur tous types de réseaux de neurones convolutifs. Contrairement aux méthodes classiques, DIFAIR optimise *directement l'espace de représentation du modèle*. En général, l'optimisation de l'espace latent est indirecte et découle de l'optimisation de la portion du modèle qui assigne une classe à la donnée d'entrée, c'est-à-dire le classifieur, qui travaille sur les attributs extraits en amont dans le modèle.

Puisque nous cherchons à optimiser la représentation latente apprise par le modèle, nous avons également cherché à inclure dans le processus d'optimisation des *contraintes d'interprétabilité* de cette représentation. Conceptuellement, nous souhaitons associer chaque dimension de la représentation à une classe donnée, et optimiser cette représentation de sorte qu'une dimension donnée soit exploitée (nous dirons par la suite « activée ») uniquement si la classe associée est présente dans la donnée d'entrée, indiquant alors la détection de caractéristiques spécifiques à cette classe : les représentations apprises seraient alors *différenciées*. Partant du principe qu'une donnée dont la classe est inconnue ne devrait pas présenter *l'ensemble* des caractéristiques de données de classe connues, et devrait donc exhiber une représentation avec moins d'activations qu'une donnée connue, il devrait alors être possible de détecter qu'une telle donnée est inconnue au modèle, et ne devrait pas faire l'objet d'une classification. Une donnée inconnue peut présenter des caractéristiques communes avec des classes connues du modèle : dans ce cas, nous souhaitons que la représentation qu'en extrait le modèle exhibe des activations sur des dimensions connues, permettant alors d'effectuer un *rapprochement sémantique* entre la donnée inconnue et

les classes connues du modèle. Bien que nous ne soyons pas allés jusque-là dans nos travaux, on pourrait imaginer, grâce à un tel mécanisme, réaliser une propagation de labels sur des données inconnues (c'est-à-dire inférer leur classe à grand traits) : cela permettrait, par exemple, d'étiqueter des données, ou de s'assurer de la qualité d'un étiquetage existant.

Ainsi, pour répondre avec DIFAIR aux deux grandes problématiques soulevées précédemment (la détection de données n'appartenant pas à des classes connues et la possibilité d'interpréter les résultats d'un modèle), nous avons identifié plus précisément les questions et verrous suivants :

1. Considérant que les données inconnues peuvent être de types très divers, nous nous sommes interrogés sur la possibilité de contraindre la représentation apprise par un modèle en utilisant uniquement des données de classes connues.
2. Partant de là, la question se pose de formaliser la fonction objectif du modèle, qui permettra de contraindre la représentation apprise. Cette formalisation est délicate, car le modèle peut parvenir d'une manière inattendue (et non souhaitée) à optimiser cette fonction, ainsi que l'explique par exemple Stuart Russell dans son ouvrage *Human Compatible : Artificial Intelligence and the problem of control* (Russell, 2019). Comme nous le verrons, nous avons, à notre échelle, également été confrontés à ce problème.
3. Nous nous sommes également posé la question de l'évaluation de notre modèle : s'il existe différentes métriques pour évaluer des modèles, tant en apprentissage en *closed set* qu'en *open set*, nous nous sommes interrogés sur *ce que mesurent ces métriques* et leur adéquation à notre problème.
4. Enfin, nous visons également à étudier l'interprétabilité des représentations apprises à travers notre approche. Il s'agit ici, assez modestement, de permettre à un utilisateur d'interpréter les résultats du modèle sans passer par un modèle proxy (qui va, par exemple, déterminer quelles sont les caractéristiques les plus saillantes d'une donnée d'entrée pour une classification donnée), mais en exploitant directement l'espace de représentation construit par le modèle : il n'est donc pas seulement question de mieux interpréter des résultats, mais aussi le modèle en lui-même.

Nous abordons ces différentes questions au fil de ce document, organisé comme suit :

— **Chapitre 1 : Contexte**
Ce chapitre présente les notions essentielles à la compréhension des travaux que nous proposons, en détaillant les réseaux de neurones et en introduisant les concepts d'incertitude et d'OSR.

— **Chapitre 2 : Open-Set Recognition : état de l'art**
Ce chapitre présente un état de l'art de l'OSR, en commençant par les méthodes d'évaluation des modèles, puis en présentant différentes approches existantes, par rapport auxquelles nous positionnons nos travaux.

— **Chapitre 3 : DIFAIR : DIFferentiAted Image Representations**
Ce chapitre présente notre approche : DIFAIR. Nous détaillons son fonctionnement et les différentes hypothèses sous-jacentes à notre approche, avant de présenter les techniques d'analyse utilisées pour évaluer nos modèles.

— **Chapitre 4 : Analyse conceptuelle du modèle DIFAIR**
Ce chapitre présente les premiers résultats obtenus avec DIFAIR et le processus d'analyse des représentations mis en place pour identifier les points à améliorer. Des solutions sont proposées pour adresser ces points, puis une recherche par grille est réalisée pour trouver la meilleure configuration de DIFAIR.

— **Chapitre 5 : Application de DIFAIR en OSR**
Ce chapitre présente l'évaluation en OSR de DIFAIR v1, dont la configuration et les hyperparamètres ont été sélectionnés *via* une recherche par grille. Une analyse approfondie des représentations obtenues permet d'identifier des limites de cette nouvelle version.

— **Chapitre 6 : Pistes pour dépasser les limites de DIFAIR v1**
Suite à l'identification de certaines limites de DIFAIR v1, nous proposons dans ce chapitre, de nature exploratoire, différentes pistes pour dépasser ces limites, et présentons des résultats préliminaires.

Cette thèse se termine par un chapitre de conclusion, dans lequel nous résumons les contributions de nos travaux, les limites rencontrées, et les perspectives de recherche envisagées.

Chapitre 1

Contexte

Résumé

Ce chapitre présente le contexte général de la thèse, en commençant par une introduction aux réseaux de neurones entièrement connectés et aux réseaux de neurones convolutifs, suivie de la description de la méthode d'apprentissage des réseaux de neurones. Suite à cela, nous introduisons le problème de la détection des classes inconnues par les réseaux de neurones et explorons deux champs de recherche qui pourraient permettre de traiter ce problème : l'incertitude et l'Open-Set Recognition (OSR).

Sommaire

1.1 Réseaux de neurones

Les travaux proposés dans cette thèse s'inscrivent dans le domaine de l'apprentissage profond, et s'intéressent à des concepts fondamentaux des réseaux de neurones appliqués à la classification d'images, par exemple la reconnaissance d'un objet dans une image. En particulier, une étude approfondie des réseaux de neurones convolutifs, principalement utilisés pour la classification d'images, est proposée dans cette thèse. Pour cette raison, nous présentons dans cette section une description détaillée des réseaux de neurones, en commençant par une description du *perceptron*, l'architecture

la plus simple de réseau de neurones, et en établissant une l'analogie avec les neurones biologiques. Nous décrivons ensuite les réseaux de neurones entièrement connectés, qui combinent plusieurs couches de plusieurs neurones. Suite à cela, nous présentons les réseaux de neurones convolutifs, qui sont au cœur de notre approche. Nous décrivons également l'algorithme de *rétropropagation du gradient* (Rumelhart et al., 1986), permettant aux réseaux de neurones d'apprendre à résoudre le problème qui leur est posé.

1.1.1 Le perceptron : un modèle bio-inspiré

Le *perceptron* est l'architecture la plus simple de réseau de neurones. Proposé en 1957 par Frank Rosenblatt, puis publié en 1958 (Rosenblatt, 1958), il permet de réaliser une classification binaire et est inspiré du fonctionnement des neurones biologiques. L'idée de cette section n'est pas d'explorer en détails les aspects biologiques des neurones, mais simplement de mettre en lumière l'analogie entre les neurones biologiques et les neurones artificiels.

Un neurone biologique reçoit des signaux électriques de plusieurs autres neurones. Ces signaux sont transmis par des synapses, qui relient des neurones entre eux. Si le neurone reçoit des signaux électriques au-delà d'un certain seuil, il « s'active » et envoie à son tour un signal électrique aux neurones auquel il est connecté. Une information, sous forme d'impulsion électrique d'une certaine intensité, est donc transmise de neurone en neurone dans le cerveau.

De la même manière, un perceptron (un neurone artificiel) est une fonction qui reçoit en entrée des signaux qui correspondent à des attributs *pondérés*. La figure 1.1 schématise cette situation pour un perceptron avec n attributs en entrée. La somme des entrées pondérées est effectuée, puis une *fonction d'activation* g est appliquée à cette somme : cette fonction d'activation détermine la valeur renvoyée par le neurone. Dans un perceptron, g était initialement une fonction en escalier, par exemple la fonction de Heaviside, renvoyant 0 (pas d'activation) pour une valeur négative et 1 pour une valeur positive.

Avec une telle architecture, il est possible de séparer des classes linéairement séparables, le perceptron apprenant un plan qui sépare les données.

Chaque entrée n du neurone est pondérée par un poids w_n. Un biais, permettant de modifier le seuil d'activation du neurone est généralement ajouté à la somme pondérée réalisée par le neurone. Ce biais peut être représenté par l'ajout d'une entrée supplémentaire de valeur 1, qui est aussi

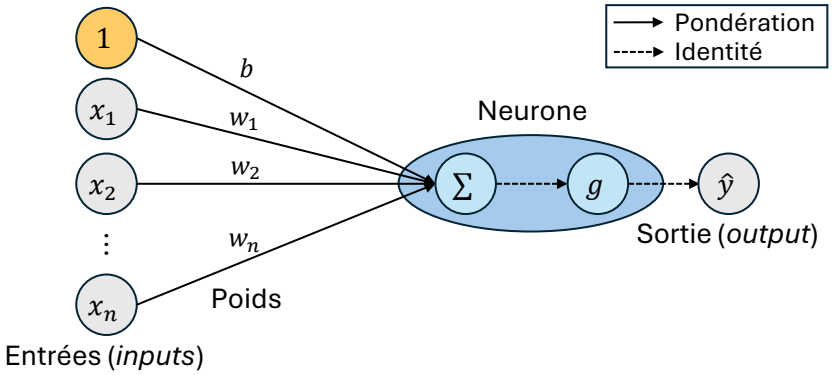


FIGURE 1.1: **Schéma d'un perceptron**, avec n variables en entrée. Un neurone de biais (en jaune), pondéré par un poids b, peut être ajouté afin de modifier le seuil d'activation du neurone.

pondérée par un poids b. La pondération des attributs permet de déterminer *l'importance de chaque attribut* dans la prédiction. Un poids élevé signifie que l'attribut est significatif pour la prédiction, inversement avec un poids faible.

Mathématiquement, le fonctionnement d'un perceptron peut être résumé comme suit pour n attributs en entrée, $\{x_1, \ldots, x_n\}$:

$$
\begin{aligned}
z &= b + \sum_{i=1}^{n} w_i x_i, \\
a &= g(z), \\
\hat{y} &= a,
\end{aligned}
$$

où b est le biais, les w_i sont les poids appliqués à chaque entrée, g est la fonction d'activation, a correspond à la valeur en sortie du neurone et $\hat{y}$ est la prédiction du perceptron. Dans ce cas, puisqu'il n'y a qu'un neurone, la prédiction est la valeur de sortie du neurone. Grâce aux opérations matricielles, il est possible d'écrire $\hat{y} = g(\boldsymbol{w}\boldsymbol{x}^T + b)$, où $\boldsymbol{w}$ est le vecteur des poids et $\boldsymbol{x}$ le vecteur des attributs.

Il est possible d'utiliser plusieurs perceptrons pour effectuer une prédiction multi-classes, chaque perceptron effectuant une prédiction binaire quant à l'appartenance à une classe. Cependant, un perceptron étant limité à l'apprentissage d'une séparation linéaire des données, cela n'est pas adapté pour des problèmes plus complexes.

1.1.2 Réseaux de neurones entièrement connectés

Afin de parvenir à résoudre des tâches plus complexes, il est possible de combiner plusieurs perceptrons, organisés sur plusieurs couches ; les sorties des perceptrons d'une couche étant les entrées des perceptrons de la couche suivante. Dans la suite, le terme « neurone » est utilisé à la place de perceptron, bien que l'architecture présentée soit souvent appelée *Multi-Layer Perceptron* (MLP).

Un MLP est constitué de plusieurs couches de neurones où chaque neurone d'une couche est connecté à *tous les neurones* de la couche précédente. Ce type de réseau est aussi appelé réseau de neurones *entièrement connecté* (*fully connected* ou *dense* en anglais). Cette architecture est présentée dans la figure 1.2. Un MLP peut être décomposé en trois types de couches : les entrées du réseau peuvent être considérées comme une couche de neurones : la couche d'entrée ; puis viennent une ou plusieurs couches cachées contenant des neurones, chacun fonctionnant à la manière d'un perceptron. Lorsqu'il y a plus de deux couches cachées dans le modèle, le terme *réseau de neurone profond* est généralement employé. Enfin, la couche de sortie contient autant de neurones que de valeurs à prédire. Notons que la fonction d'activation appliquée aux neurones de la couche de sortie est parfois différente de celle appliquée aux autres neurones du modèle. Par exemple, pour prédire un chiffre à partir d'une image, la couche de sortie contiendra 10 neurones et utilisera la fonction d'activation softmax (présentée ci-dessous) pour obtenir une distribution de probabilités sur les classes possibles. Dans le cas d'une classification binaire, il est possible d'utiliser un seul neurone en sortie et de définir un seuil sur la valeur de sortie pour déterminer la classe prédite.

Dans un MLP, les attributs en entrée du modèle sont pondérés puis combinés pour créer de nouveaux attributs en sortie de la première couche cachée. Ces nouveaux attributs sont ensuite pondérés et combinés sur la couche suivante, et ainsi de suite. Cela permet au modèle d'apprendre une hiérarchie d'attributs et de capturer des relations complexes entre les attributs en entrée du modèle, qui ne peuvent pas être capturées par un simple perceptron.

Mathématiquement, nous notons C_K un MLP avec L couches (sans compter la couche d'entrée) prédisant K valeurs en sortie. L'ensemble des notations présentées dans la suite est illustré sur un modèle avec $L = 3$ couches dans la figure 1.3. Chaque couche $l \in \{0, \dots, L\}$ a n_l neurones. La couche d'indice $l = 0$ correspond à la couche d'entrée, la couche $l = L$ à la couche de sortie. Les poids de la couche $l \in \{1, \dots, L\}$ sont représentés

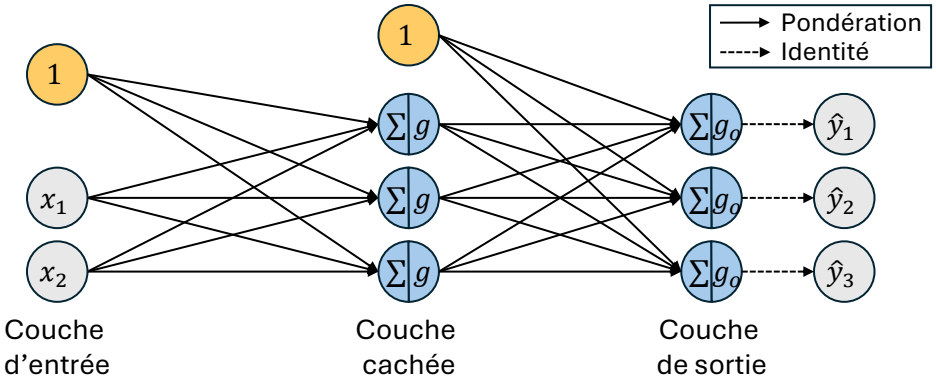


FIGURE 1.2: **Schéma d'un réseau de neurones entièrement connecté à 1 couche cachée.** Ce modèle prend en entrée 2 attributs et prédit 3 valeurs en sortie. La fonction d'activation des neurones est notée g sur la couche cachée. La fonction d'activation g_o des neurones de la couche de sortie peut être différente de g. Les neurones représentés en jaune correspondent aux biais.

par une matrice $\boldsymbol{W}^{(l)}$ de taille $n_l \times n_{l-1}$, dont les coefficients $W_{ij}^{(l)}$ correspondent au poids connectant le neurone $j \in \{1, \dots, n_{l-1}\}$ de la couche $l-1$ au neurone $i \in \{1, \dots, n_l\}$ de la couche l. L'ordre des coefficients dans la matrice est justifié par la multiplication matricielle effectuée pour calculer efficacement la valeur des neurones de la couche l à partir des valeurs des neurones de la couche $l-1$. Les biais de la couche l sont représentés par le vecteur $\boldsymbol{b}^{(l)}$ de taille n_l. La fonction d'activation des neurones de la couche l est notée $g^{(l)}$.

Notons $\boldsymbol{x} \in \mathbb{R}^{n_0}$ le vecteur des attributs en entrée du modèle. En utilisant les notations définies pour décrire le modèle C_K, les valeurs calculées sur la couche $l \in \{1, \dots L\}$ sont les suivantes :

$$\begin{aligned} \boldsymbol{z}^{(l)} &= \boldsymbol{W}^{(l)} \boldsymbol{a}^{(l-1)} + \left(\boldsymbol{b}^{(l)}\right)^T, \\ \boldsymbol{a}^{(l)} &= g^{(l)}(\boldsymbol{z}^{(l)}), \end{aligned}$$

avec $\boldsymbol{a}^{(0)} = \boldsymbol{x}^T$. La prédiction du modèle est donnée par $\hat{\boldsymbol{y}} = \boldsymbol{a}^{(L)}$. À noter que le vecteur $\boldsymbol{z}^{(L)}$ est souvent appelé vecteur de *logits*, que nous notons par la suite $\boldsymbol{o}$, correspondant à la sortie du modèle sans application de la fonction d'activation de la dernière douche.

En représentant les calculs de cette manière, il est possible de réaliser une prédiction sur plusieurs instances à la fois. En effet, si $\boldsymbol{X}$ est une matrice de taille $m \times n_0$, contenant une ligne par instance et une colonne par

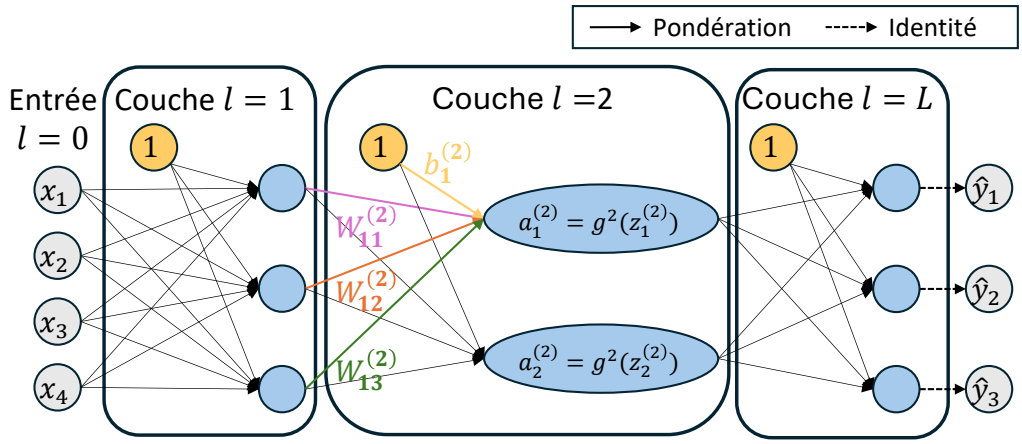


Figure 1.3: **Notations pour un réseau de neurones entièrement connecté.** Sur la couche $l = 2$, les valeurs $z_i^{(2)}$ et $a_i^{(2)}$ correspondent respectivement à la somme pondérée et à la valeur du neurone d'indice i de cette couche.

attribut, alors la seule différence avec le calcul précédent est que les sommes pondérées et les sorties de chaque couche sont des matrices $\boldsymbol{Z}^{(l)}$ et $\boldsymbol{A}^{(l)}$, avec $\boldsymbol{A}^{(0)} = \boldsymbol{X}^T$. Le biais correspond toujours à un vecteur et il doit donc être transposé et ajouté à chaque colonne de la matrice $\boldsymbol{W}^{(l)}\boldsymbol{A}^{(l-1)}$.

Pour introduire de la non-linéarité dans le réseau, d'autres fonctions d'activations ont été proposées afin de permettre au modèle d'apprendre des relations plus complexes entre les attributs. Sans non-linéarité dans le réseau, alors même un réseau très profond est équivalent à un réseau à une seule couche cachée (Géron, 2019). En effet, la somme de plusieurs fonctions linéaires est une fonction linéaire. Par exemple, si $f(x) = 3x + 4$ et $g(x) = 2x - 1$, alors $f(g(x)) = 3(2x - 1) + 4 = 6x + 1$ est aussi une fonction linéaire.

Nous présentons les principales fonctions d'activation existantes, ainsi que la fonction LeakyReLU (Maas et al., 2013) qui est utilisée dans le réseau étudié dans nos travaux. La valeur z correspond à la somme pondérée des entrées du neurone, c'est donc la valeur sur laquelle la fonction d'activation est appliquée. Ces fonctions d'activation sont :

— La fonction sigmoïde renvoie une valeur comprise entre 0 et 1 et est définie par :

$$\sigma(z) = \frac{1}{1 + \exp(-z)}\,.$$

— La fonction tangente hyperbolique est similaire à la fonction sigmoïde, mais renvoie une valeur comprise entre -1 et 1, elle est définie par :

$$\tanh(z) = \frac{1 - \exp(-2z)}{1 + \exp(-2z)}.$$

— La fonction *Rectified Linear Unit* (ReLU) (Nair & Hinton, 2010) renvoie 0 si z est négatif et z sinon, elle est définie par :

$$\mathrm{ReLU}(z) = \max(0, z).$$

— La fonction LeakyReLU (Maas et al., 2013) est une variante de la fonction ReLU qui autorise une activation du neurone pour une valeur négative de z. Elle est paramétrée par un hyperparamètre α définissant la pente sur la partie négative de la fonction et est définie par :

$$\mathrm{LeakyReLU}(z) = \begin{cases} z & \text{si } z \geq 0, \\ \alpha z & \text{sinon.} \end{cases}$$

— La fonction softmax est utilisée principalement sur la dernière couche du réseau lors d'une classification multi-classes. Elle transforme les valeurs des neurones en sortie du réseau en une distribution de probabilités sur les K classes possibles, elle est définie pour la classe i par :

$$\mathrm{softmax}(z)_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}.$$

Ces fonctions d'activation sont représentées dans la figure 1.4 à l'exception de la fonction softmax qui ne peut pas être représentée en deux dimensions, car elle s'applique sur un vecteur de dimension K et renvoie un vecteur de dimension K également. Nous pouvons par contre illustrer son fonctionnement par l'exemple suivant, où la fonction softmax est appliquée à un vecteur $\boldsymbol{o}$ de logits :

$$\boldsymbol{o} = \begin{pmatrix} 1.3 \\ 3.7 \\ 0.8 \\ 2 \end{pmatrix}, \quad \mathrm{softmax}(\boldsymbol{o}) = \begin{pmatrix} 0.07 \\ 0.75 \\ 0.04 \\ 0.14 \end{pmatrix} ; \quad \sum_{i=1}^{4} \mathrm{softmax}(\boldsymbol{o})_i = 1.$$

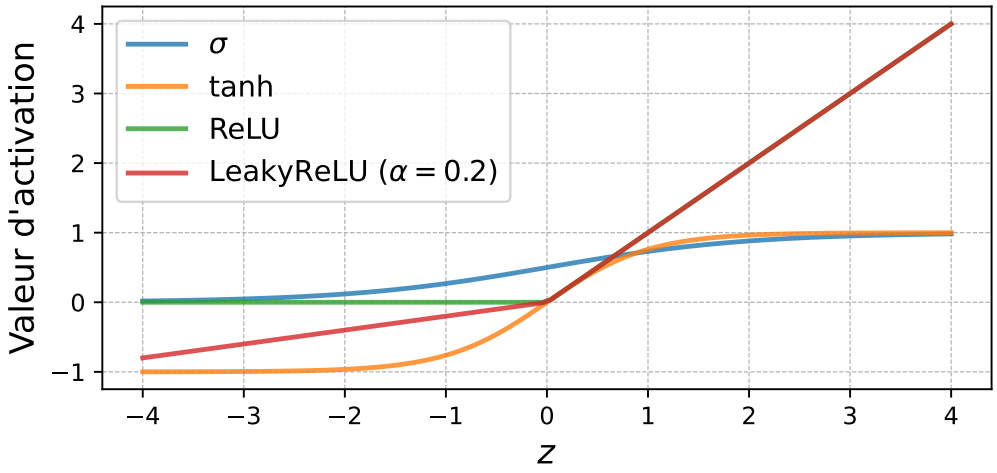


Figure 1.4: **Fonctions d'activation** : sigmoïde, tangente hyperbolique, ReLU et LeakyReLU (avec un pente de 0.2 sur les valeurs négatives). Les parties positives des fonctions ReLU et LeakyReLU sont superposées.

1.1.3 Réseaux de neurones convolutifs

Les réseaux de neurones entièrement connectés sont adaptés pour des tâches de régression ou de classification sur des données tabulaires, mais leur utilisation pour le traitement d'images est limitée. Une image est une donnée structurée : l'ordre des pixels a une importance pour la prédiction (là où l'ordre des colonnes d'un tableau n'en a pas). Une image est représentée en 3 dimensions : hauteur (h), largeur (w) et canaux (d). La plupart des images ont 3 canaux (rouge, vert, bleu) mais il est possible d'en avoir plus, par exemple des images satellites peuvent avoir un canal supplémentaire pour l'infrarouge. Le problème de l'utilisation d'un réseau entièrement connecté pour traiter une image est que, dans une image, un élément à reconnaître ne sera pas forcément toujours situé au même endroit. Or, un MLP ne prend pas en compte la structure spatiale des données, un neurone étant toujours connecté aux mêmes neurones de la couche précédente. Par exemple, si un MLP devait apprendre à reconnaître un chat dans une image, il devrait apprendre à reconnaître un chat dans chaque position possible de l'image, ce qui est inefficace.

Un autre problème de l'utilisation d'un MLP pour classifier des images est le nombre de paramètres (poids) nécessaires. Pour une image de taille $100 \times 100 \times 3$ (hauteur, largeur, canaux), un MLP avec une première couche cachée de 100 neurones aurait $100 \times 100 \times 3 \times 100 = 3\,000\,000$ de poids à

apprendre sur la première couche. Cela signifie aussi que chaque neurone de la première couche est connecté à 30 000 entrées. Sachant qu'un MLP ne prend pas en compte la structure spatiale des données, il faudrait sûrement bien plus que les 100 neurones de cet exemple sur la première couche cachée.

C'est pourquoi des architectures spécifiques ont été développées pour le traitement des images. Des études du cortex visuel de chats menées par David H. Hubel et Torsten Wiesel à la fin des années 50 (Hubel, 1959; Hubel et al., 1959) ont montré que les neurones du cortex visuel ont un *champ récepteur* localisé, c'est-à-dire que les neurones ne réagissent qu'à des stimuli situés dans une région restreinte du champ de vision. Plusieurs neurones possèdent des champs récepteurs qui se superposent et permettent ainsi d'observer un objet dans son intégralité. Suite à plusieurs travaux sur le sujet, les *réseaux de neurones convolutifs* ont été introduits par LeCun et al. (1998) pour répondre à la problématique de la classification d'images de chiffres manuscrits, et ces réseaux sont devenus les plus utilisés pour le traitement d'images (jusqu'à l'arrivée des réseaux de neurones *Transformer* (Dosovitskiy et al., 2021b) dans ce domaine).

Couche de convolution. Les réseaux de neurones convolutifs (*Convolutional Neural Network* (CNN) en anglais) utilisent des couches de *convolution* à la place de couches de neurones entièrement connectés. La différence entre les deux types de couches est qu'un neurone d'une couche entièrement connectée d'indice l est relié à tous les neurones de la couche $l-1$, tandis qu'un neurone d'une couche de convolution d'indice l a un champ récepteur restreint : il est connecté à un sous-ensemble de neurones spatialement proches sur la couche $l-1$, à travers tous les canaux. L'ensemble de poids connectant des neurones de la couche $l-1$ à un neurone de la couche l dans une couche de convolution est appelé un *filtre*. Par conséquent, un filtre peut être représenté par une matrice à trois dimensions, de taille $\mathrm{f}_h \times \mathrm{f}_w \times \mathrm{d}_{l-1}$, où f_h et f_w sont respectivement la hauteur et la largeur du filtre et d_{l-1} le nombre de canaux de la couche $l-1$. La figure 1.5 montre un champ récepteur d'un neurone et illustre le filtre associé. L'hypothèse sous-jacente au fonctionnement d'une couche de convolution, est qu'un filtre va détecter un motif particulier dans l'image, par exemple un contour, une texture ou une couleur situé dans le champ récepteur du neurone.

Sachant qu'un filtre a pour objectif la détection d'un motif dans un champ récepteur (Géron, 2019), le même filtre est appliqué à toute l'image pour détecter ce motif indépendamment de sa position dans l'image. Pour imaginer cette opération, partons du principe que le filtre est appliqué au

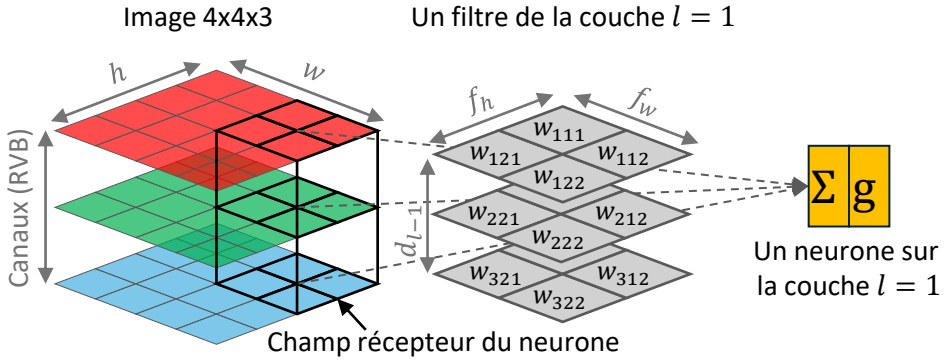


Figure 1.5: **Filtre d'une couche de convolution.** Le champ récepteur d'un neurone sur la couche l est un sous-ensemble de neurones de la couche précédente. Les neurones dans le champ récepteurs sont chacun pondérés puis additionnés avant de passer par la fonction d'activation g. L'ensemble des poids pondérant les neurones d'un champ récepteur est appelé un filtre. La taille du filtre est de $2 \times 2 \times 3$.

champ récepteur en haut à gauche de l'image. L'application du filtre permet d'obtenir la valeur d'un premier neurone (somme pondérée et activation) pour la couche suivante. Ensuite, le filtre est décalé horizontalement d'un pixel vers la droite et la valeur d'un nouveau neurone est calculée. Ce processus est répété jusqu'à ce que le filtre ait parcouru toute l'image, en effectuant aussi un décalage verticalement. L'ensemble des valeurs obtenues *via* l'application des filtres sur l'image forme une *carte d'activation* ou *carte d'attributs*, étant entendu que cette carte représente les localisations de l'image où le filtre a reconnu le motif qui lui est associé. Autrement dit, l'ensemble des neurones d'une carte d'activation partagent les mêmes poids, ce qui réduit considérablement le nombre de paramètres du modèle.

Afin d'obtenir une carte d'activation de la même taille que l'entrée de la couche de convolution, il est possible d'agrandir l'entrée en ajoutant des zéros autour (qui ne modifieront donc pas la valeur du neurone), ce qui est appelé *zero padding*. La figure 1.6 schématise l'application de filtres avec du *zero padding* sur une entrée pour obtenir une carte d'activation de même dimension que l'entrée. Le décalage du filtre le long de la hauteur ou de la largeur est appelé le *pas* (*stride* en anglais) et ne vaut pas forcément 1 : augmenter la valeur du pas permet de réduire les dimensions spatiales (hauteur et largeur) de la carte d'activation résultante et implique que les champs récepteurs sont moins chevauchants.

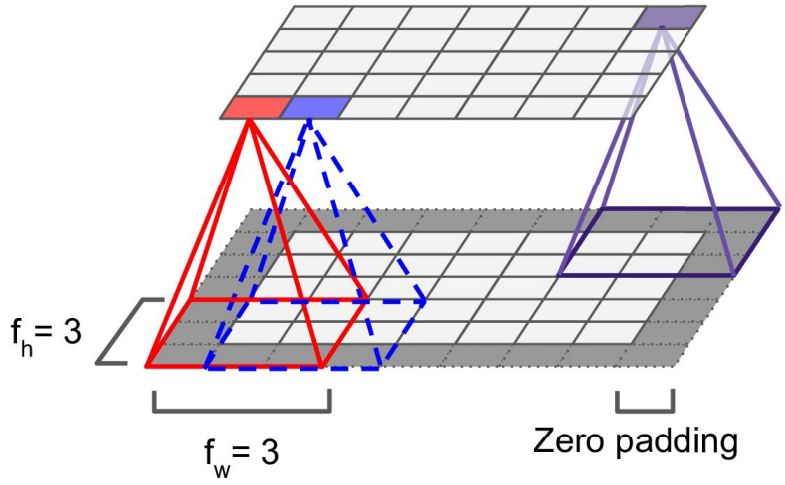


FIGURE 1.6: **Application de filtres sur une image.** Les filtres sont appliqués sur l'image en effectuant un décalage de 1 pixel à chaque fois. Le *zero padding* permet d'obtenir une carte d'activation de la même taille que l'entrée. Pour simplifier la visualisation, la profondeur de l'entrée (nombre de canaux) n'est pas représentée.
Source : Géron (2019).

Une couche de convolution contient plusieurs filtres, de façon à ce que chacun détecte un motif différent. Les cartes d'activation obtenues avec l'application de chacun des filtres sont empilées pour former une matrice en trois dimensions. Les dimensions en hauteur et en largeur de cette matrice correspondent aux dimensions spatiales, indiquant à quel endroit une activation a eu lieu. La troisième dimension correspond à la *profondeur* (*depth*) : il s'agit du nombre de cartes d'activations superposées (et aussi du nombre de filtres). Cette matrice est l'entrée de la couche de convolution suivante. Ainsi, la succession de plusieurs couches de convolution permet de détecter des motifs de complexité croissante dans l'image, la première couche détecte des motifs de bas niveau, tandis que les couches suivantes combinent ces motifs pour créer des représentations plus complexes. La figure 1.7 illustre l'empilement de cartes d'activations et leur utilisation en entrée d'une couche de convolution suivante, ainsi qu'un exemple de motifs détectés à chaque couche.

Grâce aux couches de convolution, le nombre de paramètres à apprendre est considérablement réduit par rapport à un MLP. En reprenant l'exemple précédent pour une image de taille $100 \times 100 \times 3$, alors qu'un neurone de la première couche cachée d'un MLP nécessitait 30 000 poids, un filtre de

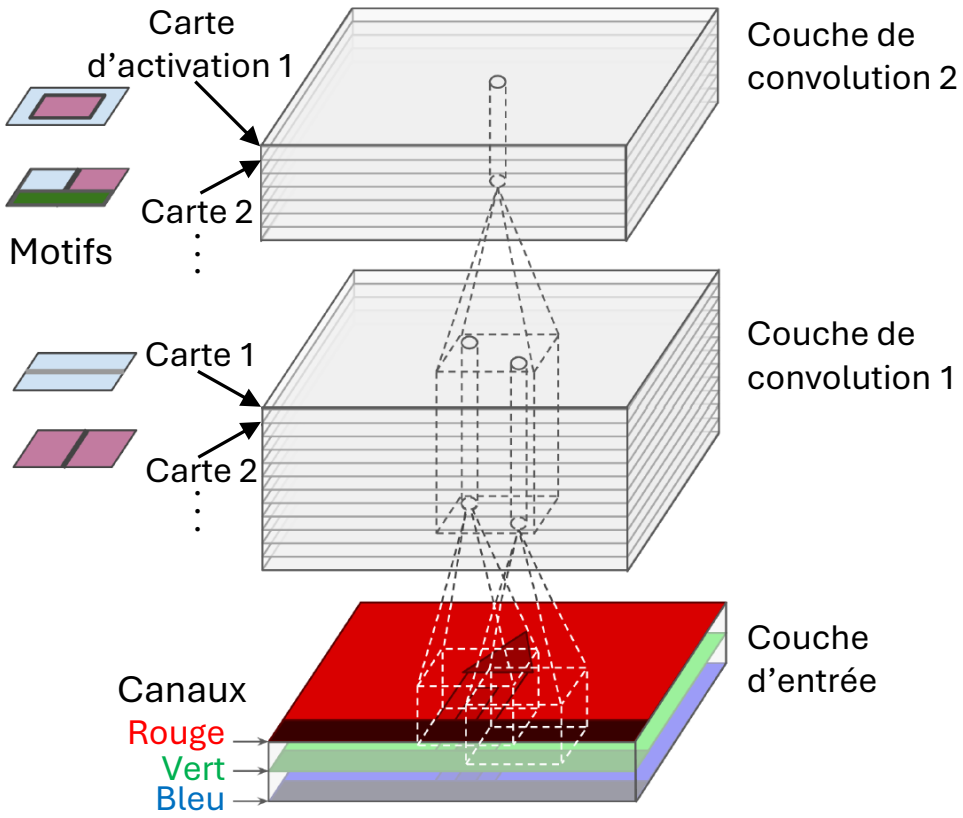


Figure 1.7: **Schéma d'une suite de couches de convolution.** Les cartes d'activation obtenues avec l'application de chacun des filtres sur l'image d'entrée sont empilées pour former une matrice en 3 dimensions. Cette matrice est l'entrée de la couche de convolution suivante. Les filtres de la première couche de convolution détectent des motifs de bas niveau, les filtres des couches suivantes combinent ces motifs en motifs de plus haut niveau.
Source : Géron (2019).

taille $f_h = 7$, $f_w = 7$ et $d_{l-1} = 3$ nécessite $7 \times 7 \times 3 = 147$ poids. Si la couche de convolution contient 64 filtres, alors le nombre de poids à apprendre sur la couche est seulement de $147 \times 64 = 9\,408$.

Couche de *pooling*. En plus des couches de convolution, les réseaux de neurones convolutifs utilisent des couches de *pooling* (Géron, 2019) afin de réduire la dimension spatiale des cartes d'activation. Cette réduction de

dimension est importante pour diminuer le nombre de calculs nécessaires (pour appliquer les filtres) et l'utilisation de la mémoire par le réseau (pour le stockage des cartes d'activation).

Un neurone d'une couche de *pooling* est similaire à un neurone d'une couche de convolution, sauf qu'il n'a pas de poids. L'objectif est d'appliquer une opération d'agrégation (moyenne, maximum, *etc.*) aux neurones situés dans le champ récepteur. Généralement, le pas de décalage du filtre est égal à la taille du filtre afin qu'il n'y ait pas de chevauchement entre les champs récepteurs (cela correspondrait à l'agrégation de la même information plusieurs fois). Il faut tout de même noter que contrairement aux opérations réalisées dans une couche de convolution, le *pooling* n'est pas appliqué en profondeur, mais uniquement sur les dimensions spatiales. La matrice de carte d'activation résultante est donc de profondeur égale à la profondeur de la matrice de carte d'activation en entrée. La figure 1.8 illustre l'application d'une opération de *pooling*.

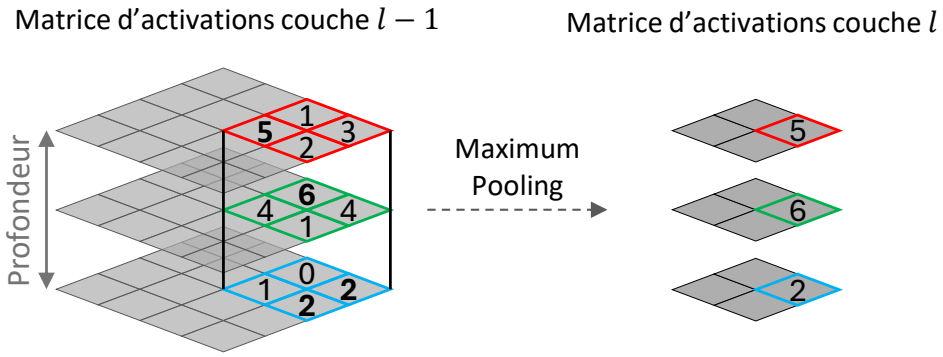


FIGURE 1.8: **Opération de *maximum pooling*.** L'opération de *pooling* est appliquée sur un champ récepteur de la matrice de cartes d'activations. L'opération d'agrégation utilisée est le maximum et est appliquée uniquement sur les dimensions spatiales. Le pas de décalage du filtre est égal à la taille du filtre.

Clarification terminologique. Avant de présenter l'architecture complète d'un réseau de neurones convolutif (CNN), précisons le vocabulaire que nous utilisons pour décrire l'action d'un CNN. Ces termes sont ceux que nous utilisons dans la suite de ce document pour décrire nos travaux. Considérons une image contenant une classe à reconnaître. Le terme « caractéristiques » correspond à des éléments inhérents à la classe présente dans l'image.

Le terme « attributs » désigne les éléments extraits de l'image par le réseau de neurones. Chaque couche de convolution extrayant des motifs de plus en plus complexes, la sortie de la dernière couche de convolution est considérée comme étant une représentation de l'image en termes d'attributs pertinents pour la tâche de classification. En pratique, la sortie de la dernière couche de convolution est une matrice de cartes d'activation, mais chaque carte d'activation représente la détection d'un attribut particulier dans l'image.

Il est important de noter qu'il est très difficile de déterminer quelles caractéristiques sont représentées par les attributs extraits : les attributs étant des combinaisons de motifs détectés par les filtres, ils peuvent par conséquent être représentatifs de la présence de plusieurs caractéristiques à la fois. De plus, cette interprétation dépend de la granularité prise en compte en définissant à quoi correspondent les caractéristiques (par exemple pour reconnaître un camion : jante, pneu, habitacle complet, *etc.*).

Architecture complète. À partir des différentes couches de réseau de neurones définies, nous présentons l'architecture globale d'un réseau de neurones convolutif (CNN) utilisé pour une tâche de classification d'images. Généralement, l'architecture peut être décomposée sous forme de *blocs de convolution* (Simonyan & Zisserman, 2015). Le bloc de convolution le plus basique est constitué de plusieurs couches de convolution, utilisant généralement du *zero padding* afin de ne pas réduire les dimensions spatiales au sein du bloc. L'objectif est d'extraire des informations de haut niveau à une même échelle, alors qu'une compression de l'entrée après peu de couches pourrait mener à une perte d'information. Après ces couches de convolution, un bloc se termine par une opération de réduction des dimensions spatiales, soit en utilisant une couche de *pooling*, soit en utilisant une couche de convolution avec un pas d'application des filtres plus grand. Plusieurs blocs de convolution sont mis bout à bout pour former un *extracteur d'attributs*.

En sortie de l'extracteur d'attributs, une matrice de cartes d'activation est obtenue. Puis, une couche de *global average pooling* (Lin et al., 2014) est utilisée : cette opération consiste à appliquer du *pooling* en utilisant la moyenne (*average*) comme opérateur d'agrégation, avec une taille de filtre égale à la taille de la carte d'activation (*global*). Chaque carte d'activation est donc réduite à une seule valeur, nommée « attribut » et cet attribut est « activé » (détecté par le modèle) si sa valeur est élevée. Le vecteur d'attributs obtenu est la *représentation* de l'image, qui a été extraite par le réseau de neurones, notée $\boldsymbol{z}$. Puisque le CNN extrait une représentation de

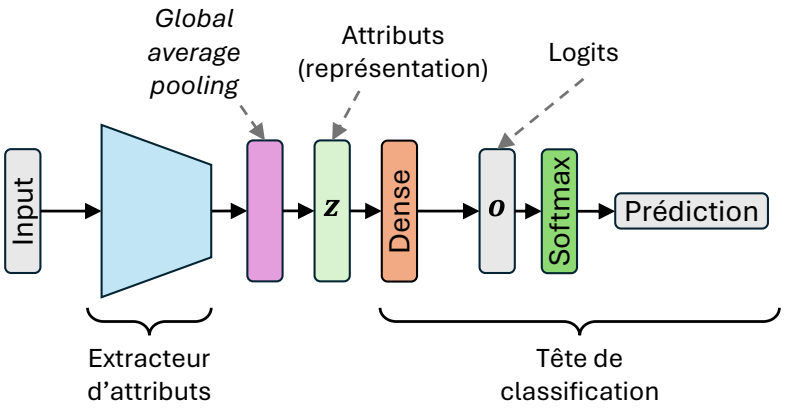


FIGURE 1.9: **Architecture d'un réseau de neurones convolutif.** L'architecture est composée de plusieurs blocs de convolution, regroupés en un extracteur d'attributs. La sortie de l'extracteur d'attributs est suivie d'une opération de *global average pooling* afin d'obtenir la représentation $\boldsymbol{z}$ (vecteur d'attributs). Un MLP composé d'une couche cachée forme la tête de classification et réalise la prédiction à partir de la représentation $\boldsymbol{z}$.

l'image, le réseau possède un *espace de représentation* dans lequel les images sont placées en fonction des attributs extraits. La représentation vectorielle obtenue correspond donc à des coordonnées dans cet espace : ce concept est au cœur de nos contributions.

Le vecteur d'attributs extraits est ensuite utilisé en entrée d'un MLP pour réaliser la prédiction. La sortie non activée du MLP est appelée *logits* et est notée $\boldsymbol{o}$. Les logits sont ensuite transformés par une fonction d'activation. Dans le cas de la classification multi-classes, la fonction softmax est utilisée pour obtenir une distribution de probabilités sur les classes à partir des logits. La partie du modèle utilisant le MLP est appelée la *tête de classification*. L'ensemble de l'architecture décrite pour un CNN est illustré dans la figure 1.9. Globalement, l'architecture d'un CNN correspond à une situation classique en *machine learning* : extraire des attributs pertinents des données en entrée, puis utiliser ces attributs pour effectuer une prédiction. La différence se situe dans le fait que les attributs sont extraits des données *automatiquement* grâce aux couches de convolution.

Pour synthéthiser, deux architectures de réseaux de neurones ont été présentées : les réseaux de neurones entièrement connectés (MLP) et les réseaux de neurones convolutifs (CNN). Ces deux architectures utilisent des poids pour déterminer l'importance des différentes entrées pour la prédiction. Ces

poids, initialisés aléatoirement, doivent être optimisés, c'est-à-dire ajustés, afin de minimiser l'erreur commise par le réseau sur la tâche. L'ajustement des poids du réseau est appelé *l'entraînement*. La section suivante présente les mécanismes d'entraînement et d'évaluation des réseaux de neurones.

1.1.4 Comment un réseau de neurones apprend-il ?

Fonction de perte. Avant de présenter les mécanismes d'entraînement des réseaux de neurones, il est nécessaire de pouvoir évaluer l'erreur commise par le modèle sur la tâche à résoudre : dans notre cas, la classification d'images. Dans cette situation, la sortie du modèle est un vecteur de taille K, avec K le nombre de classes à distinguer. Dans un modèle de classification où une seule classe doit être prédite en sortie, la fonction d'activation utilisée sur la dernière couche est la fonction softmax, appliquée aux logits $\boldsymbol{o}$. La fonction softmax permet de transformer le vecteur de logits en un vecteur dont les valeurs sont entre 0 et 1, et dont la somme vaut 1. Ce vecteur est considéré comme un vecteur de probabilités sur les classes à reconnaître.

Pour une image appartenant à la classe $c \in \{1, 2, \ldots, K\}$, la vérité terrain de l'entrée $\boldsymbol{X}$ est un vecteur $\boldsymbol{y}$ de taille K, correspondant à une distribution de probabilités, dont toutes les valeurs sont 0 sauf l'élément y_c qui vaut 1. Cette représentation binaire de la classe c se nomme un vecteur *one-hot*.

Afin d'évaluer la prédiction du réseau, il faut utiliser une fonction de perte (ou fonction de coût) qui évalue l'erreur commise par le réseau par rapport au vecteur *one-hot*. L'objectif de l'entraînement est de minimiser cette fonction de perte sur les données en ajustant les poids du réseau.

Dans le cadre de la classification d'images avec l'utilisation de la fonction softmax sur la couche de sortie, la fonction de perte d'*entropie croisée* est la plus utilisée. La fonction d'entropie croisée est basée sur l'entropie de Shannon (Shannon, 1948), qui mesure la quantité d'information d'une distribution. Par exemple, une distribution uniforme a une entropie élevée, car chaque événement a la même probabilité d'occurrence, il est donc difficile de prédire l'événement. À l'inverse, une distribution où un événement a une probabilité de 1 a une entropie nulle, car l'événement est certain.

L'entropie croisée est définie entre deux distributions de probabilités et permet de mesurer la différence entre ces deux distributions, en exprimant la quantité d'information moyenne nécessaire pour passer d'une distribution à l'autre. Dans le cas de la classification d'images, la distribution de

probabilités prédite par le modèle est comparée à la distribution *one-hot* représentant la vraie probabilité de la classe. Une valeur élevée de l'entropie croisée indique donc que la distribution de probabilités prédite est très éloignée de la vraie probabilité et inversement. La fonction d'entropie croisée $\mathrm{H}(\boldsymbol{y}, \hat{\boldsymbol{y}})$ est définie pour K classes, avec $\boldsymbol{y}$ le vecteur *one-hot* représentant la vraie classe et $\hat{\boldsymbol{y}}$ le vecteur de probabilités prédit par le modèle. Elle est définie par la fonction :

$$\mathrm{H}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{i=1}^{K} y_i \log(\hat{y}_i).$$

En ajustant les poids de manière à minimiser l'entropie croisée (c'est-à-dire l'erreur de prédiction) sur l'ensemble des données utilisées, le réseau apprend à prédire le vecteur *one-hot* correspondant à la vraie classe de l'image.

Jeu de données. Afin de pouvoir entraîner un réseau de neurones à reconnaître des images, il est nécessaire de disposer d'un jeu de données contenant des images étiquetées, c'est-à-dire dont la classe est connue. Les réseaux de neurones profonds nécessitent beaucoup de données pour être entraînés afin de pouvoir extraire des attributs pertinents pour la tâche. Par extension, ces réseaux nécessitent aussi beaucoup de ressources de calcul.

Notons qu'il a été démontré que les réseaux de neurones profonds sont des *approximateurs universels* de fonctions (Hornik et al., 1989; Zhou, 2020). Cela signifie qu'un réseau de neurones avec suffisamment de paramètres (de poids) est capable d'approximer le résultat de n'importe quelle fonction. Pour cette raison, si un réseau n'est pas entraîné avec suffisamment de données, plus particulièrement s'il contient beaucoup plus de paramètres que de données disponibles pour l'entraînement, le modèle apprend « par cœur » les données : cette situation se nomme le *sur-apprentissage* et n'apporte pas de bénéfice pour une tâche de classification d'images. Ainsi, l'objectif de l'apprentissage est de permettre au réseau d'extraire des attributs qui lui permettront de *généraliser* la prédiction à des images qui n'ont jamais été vues au cours de l'entraînement.

Dans le but de vérifier qu'un réseau de neurones ne sur-apprend pas, le jeu de données initial est généralement divisé en trois sous-ensembles *disjoints* de données :

— Le jeu de données d'*entraînement*, utilisé pour ajuster les poids du modèle.

— Le jeu de données de *validation*, utilisé pour détecter le sur-apprentissage et parfois pour optimiser des hyperparamètres du modèle.

— Le jeu de données de *test*, utilisé pour évaluer la capacité du modèle à généraliser les connaissances extraites. Ces données n'ont jamais été vues par le modèle lors de l'entraînement.

Pour limiter le sur-apprentissage dans le cas où le nombre de données est trop faible, plusieurs techniques dites de *régularisation* ont été proposées. Par exemple, les régularisations L_1 et L_2 ajoutent un terme à la fonction de perte pour pénaliser la norme des poids du réseau. L'objectif est de contraindre la valeur possible des poids pour éviter que des poids très élevés ne soient appris, qui impacteraient la capacité de généralisation du modèle. L'*augmentation de données* est une autre technique utilisée pour limiter le sur-apprentissage. Cette méthode consiste à modifier légèrement les images du jeu de données, sans altérer la vraie classe (par exemple en appliquant une rotation, un zoom, *etc.*), pour augmenter la quantité de données disponibles pour l'entraînement.

Une technique de régularisation souvent utilisée est l'ajout de couches de *dropout* (Srivastava et al., 2014). Cette couche désactive aléatoirement des neurones d'une couche avec une probabilité pré-déterminée. Cela empêche le réseau de devenir trop dépendant d'un groupe restreint de neurones pour réaliser la prédiction, puisqu'ils sont susceptibles d'être désactivés. Le réseau utilise ainsi davantage de neurones pour réaliser sa prédiction, ce qui améliore sa capacité à généraliser. Les couches de *dropout* sont utilisées uniquement pendant l'entraînement du modèle, et sont entièrement désactivées lors d'une utilisation sur le jeu de test ou dans un cas réel. Une couche de *dropout* peut aussi être ajoutée dans un bloc de convolution, afin de réduire le sur-apprentissage.

Les réseaux de neurones convolutifs peuvent également utiliser des couches de *batch normalization* (Ioffe & Szegedy, 2015), une technique de régularisation permettant de rendre l'apprentissage plus stable et plus rapide. Une couche de *batch normalization* normalise les données en entrée de la couche, en les centrant en 0 et en réduisant la variance à 1. Cette couche peut être ajoutée après chaque couche de convolution dans un réseau de neurones convolutif.

Rétropropagation du gradient. Avec un jeu de données d'entraînement et une fonction de perte mesurant l'erreur commise par le modèle, celui-ci peut maintenant être entraîné. L'objectif de l'entraînement est de modifier les poids du réseau pour minimiser la fonction de perte. Autrement dit, le problème de minimisation revient à déterminer un ensemble de

poids pour lequel la fonction de perte est la plus faible possible. Pour cela, l'algorithme de *rétropropagation du gradient* (*backpropagation*) (Rumelhart et al., 1986) est utilisé.

Cet algorithme se base sur l'algorithme de *descente de gradient* utilisé pour minimiser une fonction f itérativement. À partir d'un point initial, choisi aléatoirement, l'algorithme a pour objectif de modifier itérativement les coordonnées de ce point pour qu'il atteigne le minimum de la fonction. Pour cela, à chaque itération, le gradient ∇f de la fonction à minimiser est calculé. Le gradient d'une fonction est le vecteur des dérivées partielles de la fonction par rapport à chacune de ses variables. Les dérivées partielles dans le gradient indiquent à quel point la fonction f change lorsqu'un petit changement est effectué sur chacune des variables.

Pour illustrer le processus, considérons une fonction à deux variables $f(x, y) = x^2 + 2y^3$ et un point initial $(x_0, y_0) = (1, 2)$ au pas de temps $t = 0$. Le gradient de la fonction est donné par :

$$\nabla f(x, y) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right] = \left[2x, 6y^2\right].$$

Le gradient indique la direction dans laquelle la fonction croît le plus rapidement. Par conséquent, au pas de temps t, si le point actuel ne correspond pas déjà à un minimum de la fonction, il faut modifier les valeurs de x_t et y_t dans la direction opposée au gradient pour minimiser la fonction :

$$x_t = x_{t-1} - \eta \frac{\partial f}{\partial x} = x_{t-1} - \eta 2x_{t-1},$$
$$y_t = y_{t-1} - \eta \frac{\partial f}{\partial y} = y_{t-1} - \eta 6y_{t-1}^2.$$

La valeur η est appelée le *taux d'apprentissage* et contrôle la taille du pas effectué vers le minimum à chaque itération. Si η est trop grand, l'algorithme risque de ne pas converger et peut osciller autour du minimum. À l'inverse, si η est trop petit, l'algorithme nécessitera beaucoup de temps pour converger vers le minimum. En continuant avec l'exemple précédent, si $\eta = 0.1$, alors :

$$x_1 = 1 - 0.1 \times 2 \times 1 = 0.8 \quad \text{et} \quad y_1 = 2 - 0.1 \times 6 \times 2^2 = 1.2.$$

Les valeurs se rapprochent bien du minimum de la fonction f qui est $(0, 0)$.

Appliquons maintenant le même principe à la minimisation de la fonction de perte d'un réseau de neurones. Nous présentons l'algorithme de rétropropagation du gradient dans le cas plus simple d'un réseau de neurones entièrement connecté (MLP) mais le principe est similaire pour un réseau de neurones convolutif (CNN).

Un MLP correspond à une composition de fonctions. Pour un réseau à L couches, la sortie $\hat{\boldsymbol{y}}$ du réseau pour un vecteur d'entrée $\boldsymbol{x}$ est donnée par :

$$\hat{\boldsymbol{y}} = g^{(L)}\left(\boldsymbol{W}^{(L)} g^{(L-1)}\left(\boldsymbol{W}^{(L-1)} \dots g^{(2)}\left(\boldsymbol{W}^{(2)} g^{(1)}\left(\boldsymbol{W}^{(1)}\boldsymbol{x}\right)\right)\right)\right),$$

où $g^{(l)}$ est la fonction d'activation de la couche l et $\boldsymbol{W}^{(l)}$ est la matrice de poids de la couche l. Les biais sont omis pour simplifier la notation dans cette équation. Le calcul de $\hat{\boldsymbol{y}}$ à partir de $\boldsymbol{x}$ est appelé *propagation avant* (*forward pass* en anglais).

Dans le cas de l'entraînement d'un réseau de neurones, la fonction à minimiser est la fonction de perte $\mathcal{L}(\boldsymbol{y}, \hat{\boldsymbol{y}})$ qui prend en entrée le vecteur one-hot $\boldsymbol{y}$ représentant la vraie classe et le vecteur de probabilités $\hat{\boldsymbol{y}}$ prédit par le modèle. La valeur de la fonction de perte dépend des poids du réseau, des données d'entrée $\boldsymbol{x}$, ainsi que du vecteur one-hot de la vraie classe $\boldsymbol{y}$. L'objectif est de trouver les poids qui minimisent cette fonction sur l'ensemble des données d'entraînement. Au pas $t = 0$, les poids sont initialisés avec des petites valeurs distribuées aléatoirement [1] autour de 0.

La dérivée de la fonction de perte par rapport aux poids du réseau peut être calculée en utilisant le théorème de dérivation des fonctions composées ou règle de la chaine (*chain rule* en anglais). Cette règle indique que si $z = f_1(y)$ et $y = f_2(x)$, alors la dérivée de z par rapport à x est donnée par :

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}.$$

Le gradient de la fonction de perte est calculé par rapport à chacun des poids du réseau. L'objectif est de déterminer à quel point la fonction de perte change lorsque les poids du réseau sont légèrement modifiés, le gradient indique alors la sensibilité du coût par rapport à chaque poids. La figure 1.10 illustre la dépendance de la fonction de perte $\mathcal{L}$ par rapport à la dernière couche du réseau.

Commençons par calculer le gradient de la fonction de perte par rapport aux poids de la dernière couche du réseau, $\boldsymbol{W}^{(L)}$ et $\boldsymbol{b}^{(L)}$, en utilisant la règle de la chaîne appliquée au graphe de dépendances décrit par la figure 1.10. Les dérivées partielles calculées grâce à la règle de la chaîne sur la couche

1. L'initialisation des poids constitue un pan important de la recherche en apprentissage profond (Narkhede et al., 2022).

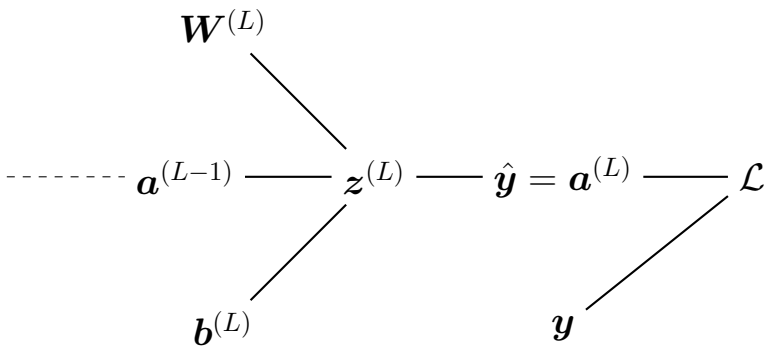


FIGURE 1.10: **Dépendance de la perte $\mathcal{L}$ par rapport à la dernière couche.**

L sont :

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^{(L)}} \cdot \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}} \cdot \frac{\partial \boldsymbol{z}^{(L)}}{\partial \boldsymbol{W}^{(L)}},$$
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^{(L)}} \cdot \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}} \cdot \frac{\partial \boldsymbol{z}^{(L)}}{\partial \boldsymbol{b}^{(L)}}.$$

Sachant que :

$$\boldsymbol{z}^{(L)} = \boldsymbol{W}^{(L)}\boldsymbol{a}^{(L-1)} + \boldsymbol{b}^{(L)},$$
$$\boldsymbol{a}^{(L)} = g^{(L)}(\boldsymbol{z}^{(L)}),$$

alors les dérivées partielles sur la couche L peuvent être réécrites comme :

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^{(L)}} \cdot g'^{(L)}(\boldsymbol{z}^{(L)}) \cdot \boldsymbol{a}^{(L-1)^T},$$
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^{(L)}} \cdot g'^{(L)}(\boldsymbol{z}^{(L)}) \cdot 1.$$

Pour la couche $L-1$, les dérivées partielles sont calculées de la même manière :

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(L-1)}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^{(L)}} \cdot \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}} \cdot \frac{\partial \boldsymbol{z}^{(L)}}{\partial \boldsymbol{a}^{(L-1)}} \cdot \frac{\partial \boldsymbol{a}^{(L-1)}}{\partial \boldsymbol{z}^{(L-1)}} \cdot \frac{\partial \boldsymbol{z}^{(L-1)}}{\partial \boldsymbol{W}^{(L-1)}},$$
$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{(L-1)}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^{(L)}} \cdot \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}} \cdot \frac{\partial \boldsymbol{z}^{(L)}}{\partial \boldsymbol{a}^{(L-1)}} \cdot \frac{\partial \boldsymbol{a}^{(L-1)}}{\partial \boldsymbol{z}^{(L-1)}} \cdot \frac{\partial \boldsymbol{z}^{(L-1)}}{\partial \boldsymbol{b}^{(L-1)}}.$$

Puisque la règle de la chaîne est utilisée pour calculer les dérivées partielles, certaines dérivées sont communes entre la couche L et la couche $L-1$. L'algorithme de rétropropagation évite de calculer indépendamment les dérivées partielles pour chaque poids du réseau : les dérivées partielles sont calculées en partant de la sortie du réseau et réutilisées pour calculer les dérivées partielles des poids des couches précédentes.

Notons $\delta^{(l)}$, l'« erreur estimée » sur la couche l :

$$\delta^{(L)} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^{(L)}} \cdot \frac{\partial \boldsymbol{a}^{(L)}}{\partial \boldsymbol{z}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{a}^{(L)}} \cdot g'^{(L)}(\boldsymbol{z}^{(L)}),$$

$$\delta^{(l)} = \delta^{(l+1)} \cdot \frac{\partial \boldsymbol{z}^{(l+1)}}{\partial \boldsymbol{a}^{(l)}} \cdot \frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}} = \boldsymbol{W}^{(l+1)^T} \delta^{(l+1)} \odot g'^{(l)}(\boldsymbol{z}^{(l)}) \text{ pour } l \neq L,$$

où $\odot$ est le produit élément par élément (nommé aussi produit de Hadamard). Dans le cas de l'utilisation de la fonction d'entropie croisée avec la fonction d'activation softmax sur la dernière couche, $\delta^{(L)}$ se simplifie et peut être calculé directement comme $\delta^{(L)} = \hat{\boldsymbol{y}} - \boldsymbol{y}$.

En utilisant les notations définies, les dérivées partielles de chacun des poids sont notées :

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(l)}} = \delta^{(l)} \cdot \boldsymbol{a}^{(l-1)^T},$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{(l)}} = \delta^{(l)}.$$

Ces formules montrent que la somme pondérée $\boldsymbol{z}^{(l)}$ et les activations $\boldsymbol{a}^{(l)}$ sont nécessaires pour calculer le gradient et doivent donc être conservées lors de la propagation avant.

À partir du gradient, les poids du réseau sont mis à jour en utilisant l'algorithme de descente de gradient, avec un taux d'apprentissage η :

$$\boldsymbol{W}^{(l)} \leftarrow \boldsymbol{W}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{W}^{(l)}},$$

$$\boldsymbol{b}^{(l)} \leftarrow \boldsymbol{b}^{(l)} - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{b}^{(l)}}.$$

Le calcul du gradient ainsi que la mise à jour des poids sont effectués pour chaque couche du réseau, en partant de la dernière couche jusqu'à la première couche, c'est pourquoi cette opération est appelée propagation arrière (*backward pass* en anglais) et que l'algorithme est nommé rétropropagation du gradient de l'erreur.

Afin de minimiser la fonction de perte, le gradient de la fonction de perte doit être moyenné sur l'ensemble des données d'entraînement ; cette moyenne permet d'obtenir la direction optimale pour modifier les poids à partir de l'ensemble des données.

Algorithme d'entraînement. Il n'est pas habituel de réaliser une descente de gradient sur l'ensemble des données d'entraînement en une seule itération : c'est une opération qui peut s'avérer coûteuse en mémoire et en temps de calcul. Généralement, les données sont divisées en *mini-batchs* (sous-lots) de B données et l'algorithme de rétropropagation est appliqué sur chaque mini-batch.

L'entraînement d'un modèle avec cette méthode est appelé *descente de gradient stochastique* (*Stochastic Gradient Descent* (SGD) en anglais). Le fait de modifier les poids du réseau sur un mini-batch à la fois permet d'augmenter le nombre de mises à jour des poids par parcours du jeu de données complet (appelé une *époque*), et de réduire le temps de calcul en parallélisant les calculs sur les mini-batchs. L'hypothèse sous-jacente à cette idée est qu'un mini-batch aléatoire peut approximer suffisamment bien le gradient global pour mettre à jour les poids du réseau. La stochasticité de l'algorithme peut aussi permettre au modèle de sortir de minima locaux. À l'inverse, entraîner le réseau sur un seul exemple à la fois est moins efficace, car le calcul du gradient sur un seul exemple n'est pas assez représentatif du gradient de l'ensemble des données et rend l'apprentissage bruité.

Pour récapituler, la suite d'opérations pour entraîner un réseau de neurones est la suivante :

1. Initialiser les poids du réseau aléatoirement.
2. Diviser les données d'entraînement en mini-batchs.
3. Pour chaque mini-batch, appliquer l'algorithme de rétropropagation du gradient :
   a) Réaliser une prédiction pour chaque donnée du mini-batch et sauvegarder les sommes pondérées $\boldsymbol{z}^{(l)}$ et les activations $\boldsymbol{a}^{(l)}$ pour chaque couche l (propagation avant).
   b) Calculer la moyenne du gradient de la fonction de perte par rapport aux poids du réseau sur le mini-batch et mettre à jour les poids du réseau en utilisant le gradient calculé (propagation arrière).
4. Répéter l'étape 3 pendant un certain nombre d'époques, ou jusqu'à ce qu'une condition d'arrêt soit rencontrée (par exemple la détection de sur-apprentissage grâce au jeu de données de validation).

Après avoir présenté les réseaux de neurones et leur entraînement, nous allons maintenant nous intéresser à la principale problématique de cette thèse : la détection de classes inconnues.

1.2 Détection de classes inconnues

La présentation précédente des réseaux de neurones soulève une question importante. En effet, un réseau de neurones n'est rien d'autre qu'une fonction mathématique qui, pour la classification d'images, prend en entrée une matrice et renvoie un vecteur représentant une distribution de probabilités sur K classes à reconnaître. Dans ce cas, une question fondamentale se pose : dans quelle mesure pouvons-nous avoir confiance en la prédiction réalisée par le modèle (Ovadia et al., 2019) ?

Dans nos travaux, nous nous intéressons particulièrement au cas où l'image d'entrée du modèle contient une classe qui n'a *jamais* été vue lors de l'entraînement. Une telle classe est appelée une classe inconnue, par opposition aux classes connues qui, elles, sont vues à l'entraînement et que le modèle devient capable d'identifier. Le problème est que, peu importe la matrice en entrée, le réseau de neurones renvoie toujours un vecteur de probabilités, même si la matrice ne représente pas une image (des valeurs aléatoires, par exemple). Par exemple, un réseau entraîné à classer des animaux auquel une image d'une table est présenté prédira qu'il s'agit d'un animal : l'usage de la fonction softmax en sortie du réseau force l'attribution d'une probabilité aux classes connues et il n'est pas possible que le modèle ne prédise *aucune* classe, car la somme des sorties doit toujours valoir 1. Ainsi, interpréter la sortie du modèle comme une distribution de probabilités n'est pas toujours adapté : cette distribution concerne les classes connues uniquement et suppose que l'entrée appartient à une classe connue.

Par conséquent, il est nécessaire de disposer d'une méthode pour détecter les entrées appartenant à des classes inconnues : d'une part, cela peut améliorer la confiance qu'un utilisateur peut avoir dans les prédictions du modèle et, d'autre part, éviter des erreurs de classification qui peuvent s'avérer coûteuses, voire dangereuses, notamment dans des domaines comme la médecine ou la conduite autonome, où une erreur de classification peut avoir de lourdes conséquences pour les patients ou les usagers.

1.2.1 Incertitude dans les réseaux de neurones

En plus de réaliser des prédictions sur des classes inconnues, les réseaux de neurones profonds sont souvent trop confiants dans leurs prédictions sur des classes connues, c'est-à-dire que la probabilité prédite par le modèle est supérieure à la probabilité réelle que l'image appartienne à cette classe (Guo et al., 2017). Un modèle correctement *calibré* est celui pour lequel la probabilité prédite se rapproche au mieux de la probabilité réelle que l'image appartienne à cette classe parmi les classes connues. Par exemple, parmi toutes les images prédites avec une probabilité de 80%, l'exactitude de la classification devrait être de 80%. En pratique, les probabilités en sortie des modèles sont souvent mal calibrées (Guo et al., 2017). Un modèle correctement calibré permettrait l'utilisation de la probabilité comme mesure de l'incertitude de la prédiction.

La calibration n'est pas le seul moyen de mesurer l'incertitude d'un modèle. Disposer d'une distribution de prédictions permet de mesurer la *variance* de la prédiction, qui représente l'incertitude du modèle. Une variance faible est signe de faible incertitude, contrairement à une variance élevée. Un réseau de neurones tel que nous l'avons présenté est déterministe, c'est-à-dire que pour une entrée donnée, la prédiction sera toujours la même. Pour obtenir une distribution de prédictions, et ainsi pouvoir estimer une incertitude sur la prédiction, plusieurs solutions existent telles qu'utiliser un ensemble de réseaux de neurones (Lakshminarayanan et al., 2017) ou bien utiliser des réseaux de neurones bayésiens (Blundell et al., 2015) ou leur approximation en utilisant des couches de *dropout* (Gal & Ghahramani, 2016). Ce sont des déclinaisons de la même idée : disposer de plusieurs versions d'une même architecture pour réaliser plusieurs prédictions différentes. Un ensemble de réseaux avec des poids différents permet d'obtenir plusieurs prédictions pour une entrée donnée. Les réseaux de neurones bayésiens utilisent des distributions pour représenter les poids, plutôt que des valeurs scalaires. À chaque prédiction réalisée avec un réseau de neurones bayésien, de nouveaux poids sont échantillonnés à partir des distributions apprises, ce qui permet là aussi de réaliser plusieurs prédictions pour une même entrée. La méthode proposée par Gal & Ghahramani (2016) consiste à ajouter des couches de *dropout* (section 1.1.4) à un réseau de neurones et à les conserver au moment de l'inférence : à chaque prédiction, des poids sont désactivés aléatoirement, ce qui permet de calculer plusieurs prédictions pour une entrée donnée.

Au début de notre travail, nous avons envisagé d'exploiter une évaluation de l'incertitude des modèles pour détecter les données appartenant à des classes inconnues : si une entrée donne lieu à une distribution de prédictions de variance importante, alors cela peut indiquer que cette entrée appartient à une classe inconnue du modèle. Des réseaux de neurones bayésiens auraient pu être adaptés au test de notre hypothèse, cependant, l'entraînement de ces modèles est plus coûteux en temps de calcul, et la mesure d'incertitude nécessite de réaliser plusieurs prédictions pour une même image et plusieurs échantillonnages de poids.

Une approche alternative, et radicalement différente de celle d'exploiter les distributions de prédictions, est de s'intéresser aux représentations latentes apprises par les modèles : si l'on peut savoir « où » les classes connues sont représentées dans l'espace latent, il devrait être possible de déduire qu'une donnée non représentée dans les sous-espaces latents connus appartiendrait donc à une classe inconnue. Pour exploiter cette hypothèse, nous nous sommes orientés vers l'*Open-Set Recognition* (OSR) en utilisant une méthode parcimonieuse en ressources de calcul et ne nécessitant pas de réaliser plusieurs inférences.

L'OSR se focalise particulièrement sur la détection de classes inconnues dans un espace de représentation réduit, par exemple la représentation latente apprise par un réseau de neurones, et non pas directement dans l'espace de représentation des images. Le problème est que ce dernier est un espace trop complexe pour être exploré directement : il est plus pratique de manipuler un espace de représentation latent, comprimé, dans lequel des sous-espaces latents peuvent être identifiés pour chaque classe connue. Cependant, la compression de l'information dans un espace latent peut induire une perte d'informations potentiellement indicatrices de la présence de classes inconnues (Yoshihashi et al., 2019).

1.2.2 Open-Set Recognition (OSR)

Concept général. Les réseaux de neurones sont généralement entraînés en *closed-set* (Scheirer et al., 2012), c'est-à-dire que les classes vues à l'entraînement, nommées classes *connues*, sont a priori les mêmes que celles rencontrées lors du test du modèle. Cependant, dans un cas réel, avec un environnement variable, il est probable que le réseau soit confronté à des classes *inconnues*, qui n'ont jamais été vues lors de l'entraînement : dans un environnement réel, les classes peuvent changer au cours du temps ou bien

de nouvelles classes peuvent apparaitre. Il est donc nécessaire de disposer d'un modèle qui soit robuste face à ces classes inconnues, pour éviter de réaliser des prédictions erronées.

Scheirer *et al.* ont introduit le concept d'*Open-Set Recognition* (OSR) (Scheirer et al., 2012), représentant un scénario de classification plus réaliste que le cas en *closed-set*. L'objectif est toujours d'entraîner le modèle sur un ensemble de classes connues, mais il sera testé sur un ensemble de classes connues et inconnues (on dit alors que l'ensemble des classes possibles au moment du test est *ouvert*). Les objectifs, au moment du test, sont de détecter les classes inconnues (et ainsi de rejeter la prédiction) tout en étant capable de classer les classes connues.

Domaines similaires. D'autres domaines de recherche, tels que la détection de nouveauté (*novelty detection*)(Pimentel et al., 2014) et la détection d'anomalies (*anomaly detection*)(Pang et al., 2021), se rapprochent de l'OSR (Salehi et al., 2022) : une situation normale est connue, et l'objectif est de détecter des différences par rapport à cette situation. Plus précisément, ce qui distingue l'OSR de ces deux autres domaines est que la capacité à classer des données connues (ou normales) est préservée. Les approches en détection de nouveauté et en détection d'anomalies utilisent des jeux de données d'entraînement qui représentent une situation normale (au sens « statistique »), considérée comme une unique classe, et un jeu de données de test binaire contenant des données normales ou anormales : il s'agit donc d'apprendre à distinguer les points normaux des points anormaux.

En détection d'anomalies, le jeu de données d'entraînement peut contenir des anomalies, ce qui n'est généralement pas le cas en détection de nouveauté ou en OSR. L'objectif est de détecter des données aberrantes, pouvant représenter une erreur, ou encore une fraude, par exemple. Les anomalies peuvent être rares par rapport aux données normales : par exemple dans le cas de mesures de la fréquence cardiaque, la majorité des données correspondront à un comportement normal et pas à des anomalies.

Par opposition, dans le domaine de la détection de nouveauté, l'ensemble d'entraînement est correctement délimité et le point de vue sur les données détectées est différent : celles-ci ne représentent pas des erreurs, mais des données d'intérêt potentiel, comme des nouvelles classes dans un problème de classification ou un comportement émergent, par exemple un changement de comportement des utilisateurs sur un site web au cours du temps.

L'OSR se rapproche aussi du domaine de la détection de données hors-distribution (*out-of-distribution detection*) (Salehi et al., 2022), où la classification multi-classe est également nécessaire. La principale différence réside dans les distributions du jeu de données d'entraînement et du jeu de donnée de classes inconnues : en OSR, toutes les données utilisées proviennent généralement de la même distribution, alors que les méthodes de détection de données hors-distribution sont entraînées sur des données provenant toutes d'une même distribution, et leur capacité à détecter des données provenant d'une autre distribution est évaluée. Par exemple, un modèle peut être entraîné à reconnaître des images naturelles et être évalué sur sa capacité à détecter des chiffres ou des images bruitées comme des données hors-distribution. Les deux domaines sont donc assez proches, pourtant les approches proposées dans chaque domaine ont pu être très différentes (Salehi et al., 2022). Cependant, certaines approches commencent à émerger pour relier les deux domaines (Zhang et al., 2020; Cevikalp et al., 2023).

Il arrive, dans ces domaines, que les modèles soient tout de même exposés à des données identifiées comme anormales (Salehi et al., 2022), afin d'apprendre au réseau à modéliser ces données et à conditionner son comportement face à ces données. Ces données sont généralement appelées *background* ou *outliers*. En OSR, le terme utilisé est *inconnues connues* (*known unknowns*) (Geng et al., 2021), car utiliser de telles données revient bien à avoir une connaissance préalable des données qui pourraient être considérées comme inconnues.

Ces méthodes reposent sur un score permettant de prendre une décision sur la nature de la donnée, que ce soit une anomalie, une classe inconnue, *etc.* (Salehi et al., 2022). En OSR, ce score est appelé score d'OSR et nous le notons $S(y \in \mathbb{C}|\boldsymbol{X})$, où $\mathbb{C}$ est l'ensemble des classes connues et $\boldsymbol{X}$ est l'image à classer. Un seuil est fixé sur ce score afin de déterminer si la donnée est une classe connue ou inconnue. Le score d'OSR le plus basique, appliqué à un modèle entraîné avec la fonction de perte d'entropie croisée et utilisé pour les expériences de contrôle (*baseline*) est la probabilité maximale prédite par le modèle, *Maximum Softmax Probability* (MSP), qui consiste non pas à rejeter les classes inconnues, mais à rejeter les prédictions *trop incertaines* (Bendale & Boult, 2015). La MSP est définie comme :

$$S(y \in \mathbb{C}|\boldsymbol{X}) = \max \operatorname{softmax}(\boldsymbol{o}) = \max \hat{\boldsymbol{y}},$$

où $\boldsymbol{o}$ représente le vecteur des logits et $\hat{\boldsymbol{y}}$ est le vecteur de probabilités prédit par le modèle. Cependant, ce score ne permet pas de limiter le risque de classification en *open-space*.

Risque de classification en *open-space*. Dans la définition de (Scheirer et al., 2012) de l'OSR, un objectif plus subtil que la détection des classes inconnues est mis en avant. L'idée est de minimiser le risque de classification en *open-space*, c'est-à-dire minimiser le risque d'assigner une étiquette à une donnée « éloignée » des classes vues à l'entraînement (une donnée inconnue). Dans l'article, le mot « éloigné » reste à définir, aucune mesure de distance, ni même l'espace dans lequel cette distance est mesurée n'est précisé. Il pourrait s'agir de l'espace de représentation des images ou bien de l'espace de représentation latent d'un modèle. La figure 1.11 illustre ce concept en réalisant un parallèle entre la classification en *closed-set* et en *open-set* : pour minimiser le risque de classification en *open-space*, il faut qu'il existe une frontière de décision séparant les données connues des données inconnues.

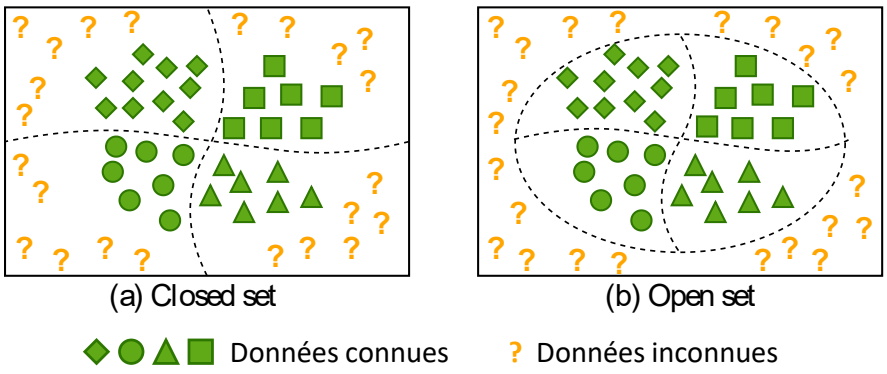


FIGURE 1.11: **Représentation de l'OSR et de l'*open space*.** Dans la figure (a), les frontières de décision séparent les classes connues, mais les données inconnues peuvent être classées comme appartenant à une classe connue. Le point de vue de (Scheirer et al., 2012) est de minimiser le risque de classification en *open-space*, c'est-à-dire de minimiser le risque d'assigner une étiquette à une donnée inconnue. Il faut donc disposer d'une frontière de décision qui sépare les données connues des données inconnues, comme dans la figure (b).
Source : Zhang et al. (2020).

Suivant cette définition, une approche utilisant la fonction d'activation softmax ne permet pas de minimiser le risque de classification en *open-space*. En effet, si une valeur est beaucoup plus élevée qu'elle ne l'est généralement dans le vecteur des logits (ce qui correspondrait à une anomalie), la normalisation par la fonction softmax résultera en une probabilité élevée pour une classe connue et la prédiction sera donc acceptée.

Théorie des valeurs extrêmes. Pour tenter de répondre à la problématique de l'OSR, plusieurs méthodes (présentées dans le chapitre 2) utilisent la *théorie des valeurs extrêmes*(*Extreme Value Theory* (EVT)) (Kotz & Nadarajah, 2000). Il s'agit d'une méthode statistique consistant à modéliser les valeurs extrêmes d'une distribution, c'est-à-dire les événements rares de la distribution, mais dont la prise en compte est nécessaire (il ne s'agit donc pas de les considérer comme des aberrations à éliminer).

Son utilisation a été proposée par Scheirer et al. (2011) lors de la présentation de concepts de *meta-recognition*, consistant à analyser les scores de sortie d'un modèle (pas spécifiquement de réseaux de neurones), pour déterminer si ces scores sont cohérents par rapport à un comportement observé initialement.

Dans le cas de l'OSR, la distribution étudiée correspond souvent à une distribution de distances par rapport à un centre représentant des données connues. Pour simplifier, on considère qu'une instance trop éloignée d'un centre connu correspond à une donnée inconnue. Afin de déterminer si une donnée est trop éloignée du centre, la distribution des distances des données connues par rapport au centre est étudiée. Les valeurs extrêmes de cette distribution de distances sont modélisées, ce qui permet d'estimer la probabilité que la distance d'une nouvelle donnée par rapport au centre corresponde à une valeur extrême. Ainsi, plus une donnée est éloignée du centre, plus la probabilité d'être une donnée inconnue augmente.

Un exemple de modélisation des valeurs extrêmes est visible dans la figure 1.12. Dans cet exemple, la distribution des distances des données connues par rapport à un centre est synthétique, mais elle décrit une situation observée en cas réel : il y a plus d'instances proches du centre (instances connues) que d'instances éloignées (instances mal reconnues ou mal classifiées). La courbe rouge représente la densité de probabilité de la distribution de valeurs extrêmes modélisée à partir des 20 valeurs les plus éloignées du centre sur un ensemble de 500 valeurs. La courbe verte représente la fonction de répartition de la distribution modélisée, indiquant la probabilité cumulée d'observer un événement extrême (de faire face à une donnée inconnue) : on remarque que la probabilité augmente, plus la distance augmente.

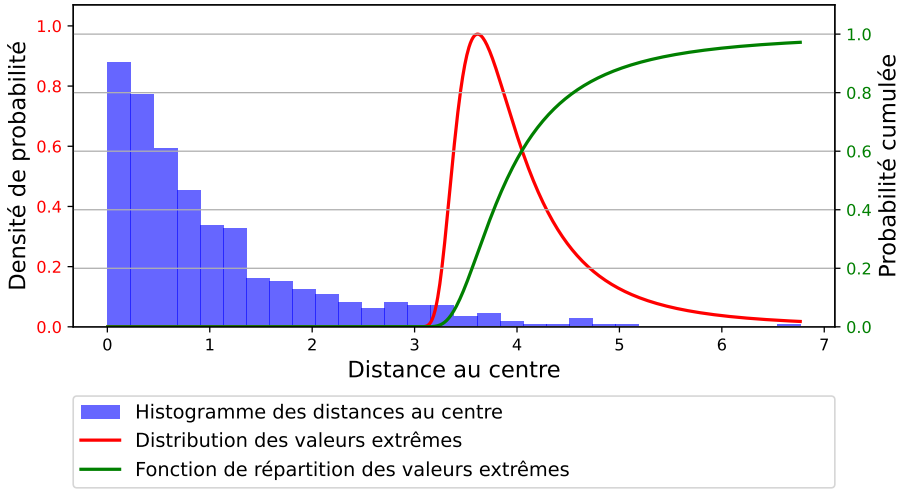


FIGURE 1.12: **Exemple de modélisation des valeurs extrêmes.** La distribution de distances des données connues par rapport à un centre est synthétique.

En raison de notre intérêt pour la manipulation des représentations latentes, nous avons choisi de nous orienter vers le domaine de l'OSR. Le chapitre suivant (chapitre 2) présente un état de l'art des méthodes en OSR ainsi que le processus d'évaluation de ces méthodes. Nous présentons ensuite en détails notre méthode basée sur l'utilisation de la représentation latente des réseaux de neurones convolutifs pour la détection de classes inconnues, dans le chapitre 3.

Chapitre 2

Open-Set Recognition : état de l'art

Résumé

Ce chapitre présente un état de l'art des méthodes d'apprentissage profond appliquées en Open-Set Recognition (OSR) dans le cadre de la classification d'images. Il débute par une introduction des notations et des mécanismes d'évaluation des méthodes en OSR. Ensuite, plusieurs approches proposées depuis la première utilisation de l'apprentissage profond en OSR sont détaillées. Enfin, nous proposons un tableau récapitulatif pour comparer les différentes approches. Cette comparaison globale des méthodes permet d'exposer notre positionnement par rapport à ces approches.

Sommaire

2.1 Évaluation des méthodes

Pour évaluer les approches en *Open-Set Recognition* (OSR), différentes métriques peuvent être utilisées. Nous introduisons tout d'abord des notations et définitions facilitant la compréhension de ces métriques. Par la suite, ces métriques sont présentées, suivies d'un benchmark devenu une référence pour l'évaluation des performances des méthodes. Enfin, nous décrivons en

détails une architecture de réseau de neurones utilisée conjointement à ce benchmark. L'utilisation combinée de ce benchmark et de cette architecture permet de comparer efficacement les performances des nouvelles méthodes proposées, car initialement chaque méthode était évaluée sur différentes tâches, parfois avec des métriques différentes, ce qui rendait la comparaison de différentes approches difficile.

2.1.1 Notations et définitions

Rappelons qu'en OSR, le premier objectif est de détecter si un modèle fait face à une donnée contenant une classe inconnue (qui ne fait pas partie des classes d'entraînement) afin de rejeter la prédiction le cas échéant. Si la donnée n'est pas identifiée comme étant inconnue, le second objectif consiste à prédire correctement la classe de la donnée. Avant de rentrer dans les détails des métriques, des notations et définitions sont introduites pour caractériser le domaine de la classification d'images en OSR.

Définissons $\mathbb{C}$ comme l'ensemble des classes connues et $\mathbb{U}$ l'ensemble des classes inconnues, où K est le nombre de classes connues. Ces deux ensembles sont disjoints, $\mathbb{C} \cap \mathbb{U} = \emptyset$. Dans le scénario de l'OSR, le jeu de données de test peut donc être divisé en deux sous-ensembles : $\mathbb{D}_{\text{test}} = \mathbb{D}_C \cup \mathbb{D}_U$, avec $\mathbb{D}_{\text{C}}$ l'ensemble des images connues et $\mathbb{D}_{\text{U}}$ l'ensemble des images inconnues.

Notons $\mathbb{D}_C = \{(\boldsymbol{X}_i, y_i) \mid y_i \in \mathbb{C}\ ,\ i = 1, \ldots, n_C\}$, où $\boldsymbol{X}$ est une image, y son étiquette, n_C le nombre d'images connues. De même, l'ensemble des images inconnues est défini par $\mathbb{D}_U = \{(\boldsymbol{X}_j, y_j) \mid y_j \in \mathbb{U}\ ,\ j = 1, \ldots, n_U\}$ avec n_U le nombre d'images inconnues.

Soit C_K un modèle de classification parmi K classes, capable de détecter les classes inconnues en comparant un score d'OSR $S(y \in \mathbb{C}|\boldsymbol{X})$ par rapport à un seuil ϵ pré-déterminé servant à rejeter ou accepter des prédictions. $\hat{y}$ est l'indice de la classe prédite par le modèle pour une image $\boldsymbol{X}$. Dans le cas où le score d'OSR permettrait de rejeter la prédiction, c'est-à-dire de déterminer qu'une image est inconnue, alors $\hat{y} = -1$. Pour résumer, $\hat{y}$ est défini comme suit :

$$\hat{y} = \begin{cases} -1 & \text{si } S(y \in \mathbb{C}|\boldsymbol{X}) < \epsilon, \\ \arg\max C_K(\boldsymbol{X}) & \text{sinon.} \end{cases}$$

Toutefois, selon l'interprétation du score d'OSR calculé, le rejet de la prédiction peut aussi être effectué si $S(y \in \mathbb{C}|\boldsymbol{X}) > \epsilon$. De même, pour certains modèles de classification, la classe prédite peut être $\arg\min C_K(\boldsymbol{X})$.

Afin de faciliter la compréhension des métriques présentées, nous proposons d'utiliser la fonction indicatrice $\mathbb{1}_{\{c\}}$ qui vaut 1 si la condition c est vraie, 0 sinon. Par exemple, grâce à cette notation, nous pouvons noter le nombre d'images connues correctement classées par le modèle, ainsi :

$$\sum_{i=1}^{n_C} \mathbb{1}_{\{\hat{y}_i = y_i\}}.$$

En considérant la tâche de détection des classes inconnues, les vrai positif (VP) correspondent aux images inconnues correctement détectées, les faux positif (FP) aux images connues incorrectement détectées comme inconnues, les vrai négatif (VN) aux images connues correctement détectées comme connues et les faux négatif (FN) aux images inconnues incorrectement détectées comme connues. Ces notations sont résumées dans le tableau 2.1.

	Prédite connue	**Prédite inconnue**
Image connue	VN	FP
Image inconnue	FN	VP

TABLEAU 2.1: **Matrice de confusion pour l'OSR.**

En utilisant les notations introduites, nous pouvons définir VP, FP, VN et FN comme suit :

$$\begin{aligned}
\text{VP} &= \sum_{j=1}^{n_U} \mathbb{1}_{\{\hat{y}_j = -1\}}, \\
\text{FP} &= \sum_{i=1}^{n_C} \mathbb{1}_{\{\hat{y}_i = -1\}}, \\
\text{VN} &= \sum_{i=1}^{n_C} \mathbb{1}_{\{\hat{y}_i \neq -1\}}, \\
\text{FN} &= \sum_{j=1}^{n_U} \mathbb{1}_{\{\hat{y}_j \neq -1\}}.
\end{aligned}$$

2.1.2 Mesures de performances

Plusieurs métriques peuvent être définies pour évaluer les performances d'un modèle en OSR. Parmi les métriques les plus fréquemment utilisées figurent l'exactitude en *closed-set*, l'AUROC et la F-mesure. Ces métriques sont présentées dans les paragraphes suivants.

Exactitude en *closed-set*. Cette première métrique consiste à évaluer l'Exactitude en *Closed-Set* (ECS) du modèle, c'est-à-dire mesurée uniquement sur les classes connues du jeu de données $\mathbb{D}_C$. L'exactitude est le pourcentage d'images connues correctement classées. Cette métrique permet de s'assurer qu'une approche mise en place spécifiquement pour l'OSR ne dégrade pas les performances de classification par rapport à un modèle de base. Elle est définie par l'équation suivante :

$$\text{ECS} = \frac{\sum_{i=1}^{n_C} \mathbb{1}_{\{\hat{y}_i = y_i\}}}{n_C}.$$

AUROC. L'AUROC est la métrique la plus utilisée pour évaluer les approches en OSR. Elle permet d'évaluer la capacité d'un modèle à distinguer les classes connues des classes inconnues. Sa valeur, située entre 0 et 1, reflète la capacité du modèle à différencier les classes connues des inconnues ; une valeur plus élevée indique une meilleure capacité de détection. Une valeur de 0.5 indique une classification aléatoire. L'AUROC est parfois exprimée en pourcentage. Cependant, l'AUROC évalue seulement la capacité de séparation des classes connues et inconnues, sans prendre en compte la performance de classification des classes connues.

Plus précisément, l'AUROC est l'aire sous la courbe *Receiver Operating Characteristic* (ROC), qui exprime le taux de vrais positifs en fonction du taux de faux positifs pour différents seuils de rejet ϵ. La figure 2.1 illustre un exemple de courbe ROC. Les taux de vrais positifs (TPR) et de faux positifs (FPR) sont calculés pour chaque seuil ϵ, selon les équations suivantes :

$$\text{TPR} = \frac{\text{VP}}{\text{VP} + \text{FN}},$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{VN}}.$$

F-mesure. La F-mesure est une métrique qui combine le rappel et la précision. Elle est utilisée pour évaluer simultanément la performance de classification des données connues et de détection des données inconnues. Pour cela, une redéfinition des VP, FP et FN est nécessaire. Bendale & Boult (2015) proposent que :

— les VP correspondent aux classifications correctes sur les données connues,

— les FP correspondent aux classifications incorrectes sur les données connues,

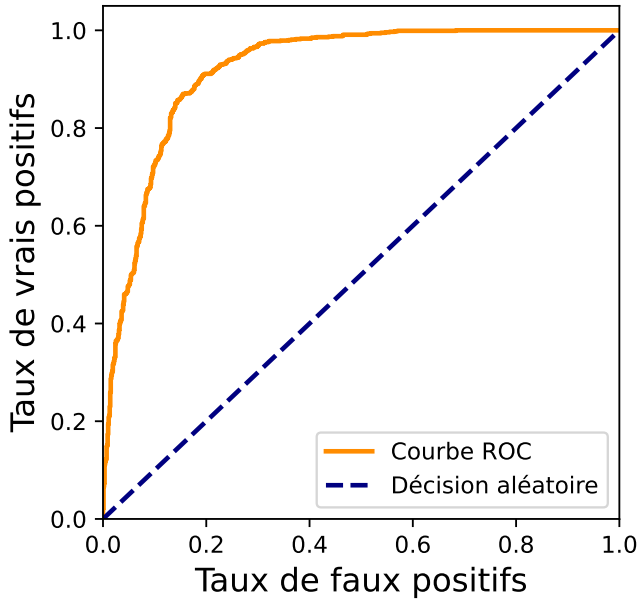


FIGURE 2.1: **Exemple de courbe ROC pour une tâche de détection de classes inconnues.** L'AUROC mesurée pour cette courbe est de 0.92. Ce résultat est issu de la tâche de détection de classes inconnues CIFAR+10 du benchmark de Neal et al. (2018) (section 2.1.3), en utilisant un modèle entraîné avec entropie croisée et le score d'OSR *Maximum Softmax Probability* (MSP) (section 1.2.2).

— les FN correspondent aux données inconnues détectées comme étant connues.

Dans ce contexte on aura :

$$\mathrm{VP} = \sum_{i=1}^{n_C} \mathbb{1}_{\{\hat{y}_i = y_i\}}, \qquad \mathrm{FP} = \sum_{i=1}^{n_C} \mathbb{1}_{\{\hat{y}_i \neq y_i\}}, \qquad \mathrm{FN} = \sum_{j=1}^{n_U} \mathbb{1}_{\{\hat{y}_j \neq -1\}}.$$

La F-mesure est définie comme la moyenne harmonique de la précision et du rappel, selon l'équation suivante :

$$\text{F-mesure} = 2 \cdot \frac{\text{Précision} \cdot \text{Rappel}}{\text{Précision} + \text{Rappel}},$$

avec :

$$\text{Précision} = \frac{\mathrm{VP}}{\mathrm{VP} + \mathrm{FP}}, \qquad \text{Rappel} = \frac{\mathrm{VP}}{\mathrm{VP} + \mathrm{FN}}.$$

Cependant, d'autres auteurs n'ont pas clairement défini la signification des VP, FP et FN pour la F-mesure, à l'instar de Bendale & Boult (2015), rendant ainsi difficile les comparaisons entre approches. D'où la nécessité de disposer d'un benchmark avec une métrique clairement établie.

2.1.3 Benchmark pour l'OSR

Pour évaluer les approches d'OSR, Neal et al. (2018) ont présenté un benchmark contenant six tâches de détection de classes inconnues, créées à partir de cinq jeux de données. Ce benchmark est devenu une référence dans le domaine de l'OSR. L'extracteur d'attributs [1] du réseau de neurones proposé dans l'approche de Neal et al. (2018) a aussi été utilisé comme référence par la plupart des articles en OSR, c'est pourquoi son architecture est présentée en section 2.1.4. La métrique utilisée pour évaluer la performance des modèles à détecter des classes inconnues sur ces tâches est l'AUROC. Cette adoption généralisée de l'architecture en plus des tâches d'évaluation proposées permet de comparer efficacement les performances des différentes approches.

Le benchmark de Neal et al. (2018) inclu les jeux de données suivants :

— **MNIST** (LeCun et al., 1998) : contient 70 000 images en noir et blanc de chiffres manuscrits, dont 60 000 pour l'entraînement et 10 000 pour le test. Les images ont une taille de 28×28 pixels. Un exemple d'image pour chaque classe est présenté dans la figure 2.2.

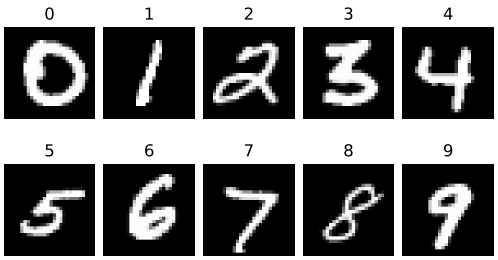


Figure 2.2: **Exemples d'images du jeu de données MNIST.**

— **SVHN** (Netzer et al., 2011) : contient 99 289 images de chiffres, représentant des numéros de maisons (SVHN signifiant *Street View House Numbers*), dont 73 257 pour l'entraînement et 26 032 pour le test. Les images ont une taille de 32×32 pixels. Un exemple d'image pour chaque classe est présenté dans la figure 2.3.

1. L'extracteur d'attributs est la partie d'un réseau de neurones qui permet d'extraire une représentation d'une image (section 1.1.3 page 20).

FIGURE 2.3: **Exemples d'images du jeu de données SVHN**. Il est possible qu'il y ait des éléments perturbateurs sur les côtés du chiffre à reconnaître. Par exemple pour la classe 0, il y a aussi un 4 sur l'image.

— **CIFAR-10** (Krizhevsky, 2009) : contient 60 000 images de 10 classes différentes, dont 50 000 pour l'entraînement et 10 000 pour le test. Six classes représentent des animaux et quatre classes des véhicules. Les images ont une taille de 32×32 pixels. Un exemple d'image pour chaque classe est présenté dans la figure 2.4.

FIGURE 2.4: **Exemples d'images du jeu de données CIFAR-10.**

— **CIFAR-100** (Krizhevsky, 2009) : contient 60 000 images de 100 classes différentes, dont 50 000 pour l'entraînement et 10 000 pour le test. Les classes sont très variées (animaux, véhicules, objets du quotidien, *etc.*). Les images ont une taille de 32×32 pixels. Un exemple d'image pour dix classes est présenté dans la figure 2.5.

— **TinyImageNet** (Le & Yang, 2015) : contient 120 000 images de 200 classes différentes, dont 100 000 pour l'entraînement, 10 000 pour la validation et 10 000 pour le test dont les étiquettes ne sont pas publiques. L'usage est donc d'utiliser les données de validation pour les tests. Il s'agit d'un sous-ensemble du jeu de données d'ImageNet (Deng et al., 2009) qui contient 1000 classes. Les images ont été redimensionnées vers une taille de 64×64 pixels (dans ImageNet, les images ont leur résolution d'origine). Dans ce jeu de données, cer-

taines classes sont très proches : il y a par exemple différentes races de chiens. Une forte variation intra-classe peut aussi être observée : la classe chat égyptien contient par exemple des statues de chats. Un exemple d'image pour dix classes est présenté dans la figure 2.6.

Figure 2.5: **Exemples d'images du jeu de données CIFAR-100.** Seules dix classes sont représentées.

Figure 2.6: **Exemples d'images du jeu de données TinyImageNet.** Seules dix classes sont représentées.

À partir de ces cinq jeux de données, Neal et al. (2018) ont proposé six tâches pour évaluer les approches d'OSR, en mesurant l'AUROC. Les jeux de données sont divisés en deux ensembles : un ensemble de classes connues et un ensemble de classes inconnues. L'objectif est d'entraîner un modèle uniquement sur les classes connues et de tester sa capacité à détecter les classes inconnues. La séparation des classes en deux groupes est effectuée cinq fois de manière aléatoire, afin d'obtenir différents ensembles de classes connues/inconnues. Pour l'évaluation d'une tâche, la moyenne de l'AUROC sur les cinq séparations est calculée. Les six tâches sont les suivantes :

— **MNIST, SVHN, CIFAR-10** : six classes sont définies comme connues et quatre inconnues.

— **CIFAR+10, CIFAR+50** : les quatre véhicules de CIFAR-10 sont les classes connues et 10 ou 50 classes sont aléatoirement sélectionnées parmi les classes de CIFAR-100 pour représenter des classes inconnues.

— **TinyImageNet** : 20 classes sont connues et les 180 restantes inconnues.

L'utilisation du benchmark n'est pas évidente ; en particulier, les séparations des jeux de données peuvent avoir une grande influence sur les performances. Notamment, si deux classes sémantiquement proches sont toutes les deux connues ou inconnues, la tâche est plus simple que si l'une est connue et l'autre inconnue. Bien qu'il y ait cinq séparations aléatoires réalisées, les probabilités que des classes proches ne soient pas séparées restent élevées sur CIFAR-10. De plus, il est à noter que les auteurs qui utilisent ce benchmark n'utilisent pas systématiquement les mêmes séparations, ou alors cette information n'est pas précisée.

2.1.4 Architecture de référence

Le benchmark proposé par Neal et al. (2018) a permis d'évaluer l'approche *OSR with Counterfactual Images* (OSRCI), également proposée par ces auteurs et présentée dans la section 2.2.1. L'extracteur d'attribut du réseau de neurones proposé dans cette approche est resté associé au benchmark pour l'évaluation de l'OSR. Cette section détaille l'architecture de référence utilisée pour entraîner les modèles destinés à résoudre les tâches en OSR.

L'architecture repose sur un réseau de neurones convolutif, avec des blocs de convolution inspirés par l'architecture VGG (Simonyan & Zisserman, 2015). Un bloc de convolution tel qu'utilisé dans l'architecture proposée est détaillé dans la figure 2.7. Au sein d'un bloc, une réduction de dimensions est effectuée en utilisant un pas de 2 dans la troisième couche de convolution.

L'architecture globale du réseau comprend trois blocs de convolution successifs, suivis d'une couche de *global average pooling* (Lin et al., 2014) afin de réduire les dimensions de la sortie des blocs. La sortie de cette couche correspond à la représentation latente, notée $\boldsymbol{z}$, apprise par le modèle. Enfin, une couche dense, entièrement connectée, permet de réaliser une prédiction à partir de la représentation $\boldsymbol{z}$. Une schématisation de cette architecture est présentée dans la figure 2.8.

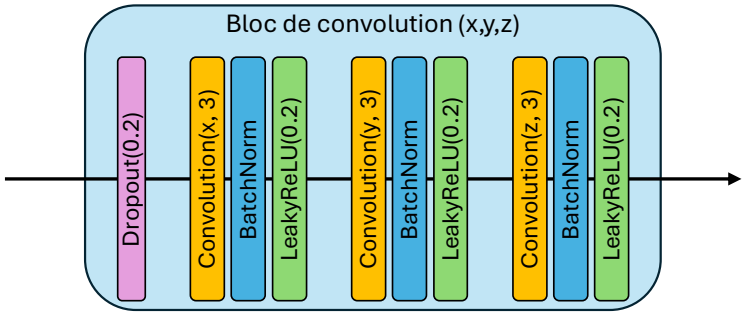


Figure 2.7: **Schéma d'un bloc de convolution dans l'architecture de Neal et al. (2018).** Un bloc de convolution est paramétré par le nombre de filtres de chaque couche de convolution. Les différents types de couches sont présentés dans la section 1.1.3.

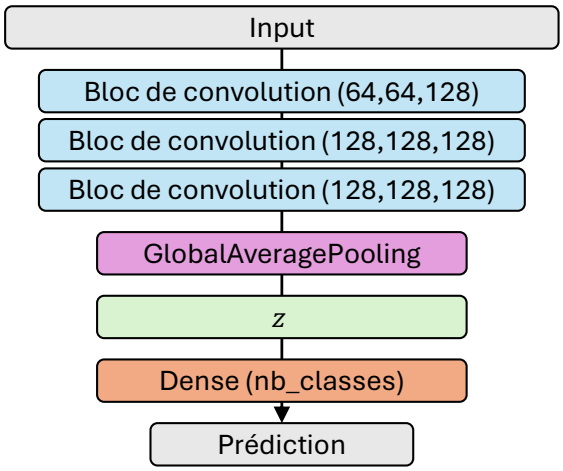


Figure 2.8: **Architecture complète du réseau utilisé en OSR.** Les différents types de couches sont présentés dans la section 1.1.3.

2.2 Détection des classes inconnues

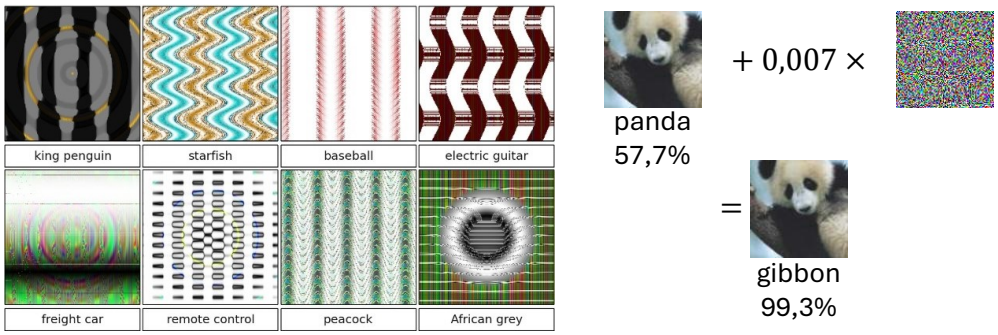
Cette section est dédiée à la présentation des méthodes d'*Open-Set Recognition* (OSR), appliquées aux réseaux de neurones profonds utilisés pour la classification d'images. Nous proposons d'abord un historique des différentes méthodes développées pour détecter les classes inconnues. Ensuite, des approches à l'état de l'art dans ce domaine sont présentées. L'approche *Class Anchor Clustering* (CAC) de Miller et al. (2021), qui est liée à nos travaux du chapitre 3, est détaillée. Enfin, pour récapituler cette section, un tableau comparatif des différentes approches est proposé, ainsi que notre positionnement par rapport à ces approches.

2.2.1 Une succession d'idées pour l'OSR

Les premiers travaux significatifs sur l'OSR appliqué en apprentissage profond ont été introduits en 2015 par Bendale & Boult (2015). Il est important de noter que la plupart des réseaux de neurones utilisent la fonction softmax en sortie pour réaliser une prédiction. Or, la nature fermée de softmax signifie qu'une probabilité est *toujours* attribuée à chaque classe rencontrée lors de l'entraînement, mais que se passe-t-il lorsqu'une image ne contient aucune des classes connues ? En réponse, Bendale & Boult (2015) proposent OpenMax, une extension de la couche softmax qui permet la prédiction d'une probabilité d'appartenance à une classe inconnue. En tant que pionniers dans l'apprentissage profond, OpenMax ou des concepts particuliers de cette méthode sont réutilisés dans les approches postérieures.

OpenMax. Bendale & Boult (2015) s'attaquent en particulier à la problématique de la classification par les réseaux de neurones d'images dites *trompeuses* (Nguyen et al., 2015) et d'images dites *adversariales* (Goodfellow et al., 2015). Les images trompeuses, visuellement différentes des images d'entraînement, sont pourtant souvent identifiées à tort par les réseaux de neurones comme contenant une classe vue lors de l'entraînement, parfois avec certitude (une probabilité élevée). Les images adversariales ont pour objectif d'être indistinguables d'une vraie image, mais sont optimisées afin qu'un réseau de neurones prédise qu'elles appartiennent à une autre classe. Des exemples d'images de chaque catégorie sont visibles sur la figure 2.9. La question posée par les auteurs est « Y a-t-il un espace de représentation latent dans un réseau de neurones, où les images adversariales et trompeuses, seraient éloignées des images connues, et donc séparables ? »

Pour aborder cette problématique, Bendale & Boult (2015) proposent d'utiliser des méthodes de *meta-recognition* (Scheirer et al., 2011), consistant à analyser les scores de sortie d'un modèle de classification. En l'occurrence, ce sont les scores en sortie d'un réseau de neurones dans l'espace de représentation des *logits*, avant application de la fonction softmax, qui sont utilisés. Ces scores sont désignés par le terme *vecteur d'activation* (VA), que nous notons $\boldsymbol{o} \in \mathbb{R}^K$, dans le cadre d'une classification à K classes. L'hypothèse sous-jacente de ces travaux est que chaque dimension o_c du vecteur exprime une *similarité* ou une *relation* avec la classe correspondante c. Imaginons un jeu de données contenant plusieurs races de chats, de chiens et des véhicules. Le vecteur d'activation correspondant à une image d'une race spécifique de chat présentera aussi des activations pour les autres races de chat, des activations plus faibles pour les classes de chien et des activations



(a) Images trompeuses.
Source : Nguyen et al. (2015).

(b) Image adversariale.
Source : Goodfellow et al. (2015).

Figure 2.9: **Exemples d'images trompeuses et adversariales.** (a) Un humain remarque tout de suite que ces images ne correspondent à aucune classe, cependant un réseau de neurone leur attribuera quand même une catégorie, avec une certitude élevée. (b) Pour un humain, il s'agit évidemment d'un panda. Cependant, un bruit imperceptible (non aléatoire) a été ajouté à chaque pixel de cette image afin qu'un réseau de neurone préalablement choisi prédise qu'il s'agit d'un gibbon.

quasiment nulles pour les véhicules. Une image adversariale, dont l'objectif est de maximiser la valeur de sortie d'une classe choisie (par exemple d'une race de chat) pourrait ne pas suivre la même distribution d'activations qu'une vraie image de chat.

Pour détecter les distributions anormales de scores, les auteurs commencent par calculer un *vecteur d'activation moyen* (VAM) pour chaque classe, à partir des VA des images d'entraînement correctement reconnues. La distance entre chaque instance correctement reconnue et son VAM associé est ensuite mesurée, afin d'obtenir une distribution des distances par rapport à chaque VAM. Puis, à l'aide de la théorie des valeurs extrêmes (présentée en section 1.2.2), K distributions sont modélisées pour les η distances les plus éloignées (les valeurs extrêmes) des K VAM. À partir de ces distributions, il est possible d'estimer une probabilité que le VA d'une instance soit situé à une distance aberrante par rapport au VAM de chaque classe.

Ces probabilités servent à recalibrer les scores de sortie du réseau et une dimension supplémentaire correspondant à une classe « inconnue » est ajoutée au VA. Enfin, la fonction softmax est appliquée sur le vecteur de scores nouvellement obtenu (recalibré), pour assigner une probabilité à chaque classe et à la classe inconnue.

En plus de faire une prédiction des classes inconnues, les auteurs prennent en compte l'incertitude du modèle pour déterminer si la classe est inconnues. En effet, pour qu'une prédiction soit rejetée, il faut que la classe inconnue ait la probabilité maximale, ou bien que la probabilité maximale parmi toutes les classes connues soit inférieure à un seuil ϵ, sous-entendant que le modèle n'a été certain de sa prédiction sur aucune classe. L'expression de l'incertitude du modèle par la probabilité de sortie est tout de même à nuancer, comme nous l'avons discuté dans la section 1.2.1 du chapitre 1.

L'approche OpenMax a été évaluée sur la détection d'images trompeuses (Nguyen et al., 2015) et d'images inconnues provenant du jeu de données ImageNet (Deng et al., 2009), qui contient des images naturelles. Les performances ont été mesurées en utilisant la F-mesure, telle que décrite dans la section 2.1.2, pour évaluer conjointement la capacité de classification et de détection des données inconnues. Selon cette métrique, OpenMax est plus performant que softmax pour tous les seuils testés, atteignant une F-mesure maximale au seuil de probabilité $\epsilon = 0.2$ avec environ 0.595 contre 0.582 pour softmax.

Ces résultats montrent qu'OpenMax permet de rejeter des images inconnues et trompeuses. Les auteurs mentionnent que leur méthode rejette « certaines images adversariales » mais sans présenter de résultats quantifiés. Il a également été observé un rejet des images contenant plusieurs classes, dû à une divergence dans la distribution des VA de ces images par rapport aux VAM qui ont été estimés pour des images contenant majoritairement une seule classe. L'évaluation des performances en OSR sur des images trompeuses et adversariales est moins courante dans les articles qui ont suivi, qui utiliseront le benchmark de Neal et al. (2018) présenté en section 2.1.3, ce dernier évaluant les performances sur des images inconnues provenant de la même distribution que le jeu de données d'entraînement.

Une piste d'amélioration envisagée est de calculer plusieurs centres par classes plutôt qu'un seul VAM. Cela permettrait de mieux représenter la variabilité des instances d'une même classe, telle que la rotation de la classe ou encore l'action (par exemple, un chat assis, qui cours, *etc.*).

Bien que les idées proposées reposent sur des hypothèses solides, et que les auteurs aient démontré mathématiquement que leur approche limite le risque de classification en *open-space* (section 1.2.2), les performances d'OpenMax restent faiblement supérieures à softmax, notamment lorsqu'elles sont comparées à d'autres approches plus récentes.

Generative OpenMax. Dans le but d'améliorer les résultats d'OpenMax, Ge et al. (2017) présentent l'approche *Generative OpenMax* (G-OpenMax). La principale nouveauté de G-OpenMax réside dans l'utilisation de *données inconnues connues* afin d'apprendre au modèle *à modéliser des données inconnues*. Selon les auteurs, avoir une représentation explicite des données inconnues dans le modèle permet de mieux apprendre les frontières de décision, et ainsi réduire le risque de classification en *open-space*. La modélisation des classes inconnues dans l'espace de représentation permet donc d'estimer à quel point une nouvelle donnée se trouve proche des classes inconnues, et pas uniquement à quel point elle est proche des classes connues. Ge et al. (2017) sont les premiers à utiliser des données supplémentaires considérées comme inconnues, dans le cadre de l'OSR appliqué à l'apprentissage profond.

Dans cette situation, le défi principal est d'obtenir des données inconnues qui ne correspondent pas à des classes connues. Pour cela, les auteurs proposent d'utiliser un réseau de neurones génératif, spécifiquement un *Generative Adversarial Network* (GAN). Le principe de fonctionnement des GANs repose sur la compétition entre deux réseaux de neurones : un générateur, qui génère des images à partir d'un vecteur de bruit aléatoire et un discriminateur, qui a pour but de déterminer si l'image qui lui est fournie a été générée ou est réelle. Ce paradigme force donc le réseau générateur à générer des images qui ressemblent à des images réelles, dans le but de tromper le discriminateur.

Plus particulièrement, les auteurs utilisent un GAN conditionnel, un type de GAN où le générateur reçoit en entrée un vecteur de bruit, mais aussi un vecteur *one-hot* spécifiant la classe à générer, conditionnant ainsi le processus de génération. Le GAN étant entraîné à partir des images connues, la difficulté consiste à générer des images inconnues qui soient « plausibles » et distinctes des classes connues. Ces travaux se basent sur l'hypothèse que les classes inconnues ne seront généralement pas diamétralement opposées aux classes connues : par exemple, si l'apprentissage est réalisé sur un jeu de données de classification d'objets, le réseau ne devrait pas faire face à des images contenant des caractères écrits. Dans ces conditions, générer des images inconnues à partir d'un réseau entraîné sur les images connues est une option acceptable.

S'appuyant sur cette hypothèse, Ge et al. (2017) proposent de conditionner le générateur avec un vecteur $\boldsymbol{m}$ représentant un mélange de classes, avec la somme des composantes de $\boldsymbol{m}$ valant 1 ($\sum_i \boldsymbol{m}_i = 1$), comme c'est le cas pour un vecteur *one-hot*. Cette technique permet de générer des images

contenant des caractéristiques de plusieurs classes connues. En théorie, dans l'espace de représentation, ces images seraient situées à proximité des images connues, mais n'en faisant pas partie, elles permettraient de délimiter une frontière entre les classes connues et inconnues.

Globalement, l'approche nécessite l'utilisation de quatre modèles et est décomposée en différentes étapes comme suit :

— L'entraînement d'un modèle de classification, C_K, sur les données connues, utilisé par la suite pour déterminer si les images générées correspondent où non à une vraie classe.

— Le GAN, entraîné sur les données connues, sert ensuite à générer des images inconnues, identifiées comme telles grâce à C_K.

— L'entraînement d'un modèle de classification C_{K+1} sur les données connues et générées, pour prédire $K+1$ classes, avec K le nombre de classes connues et la classe $K+1$ correspondant à la prédiction d'une classe inconnue.

— Les prédictions du modèle C_{K+1} sont recalibrées, à la manière d'OpenMax, en utilisant la théorie des valeurs extrêmes.

G-OpenMax est évaluée sur plusieurs tâches de détection de classes inconnues composées à partir de deux jeux de données : MNIST (LeCun et al., 2010) et HASYv2 (Thoma, 2017) contenant respectivement des chiffres et des caractères manuscrits. Les performances sont évaluées en utilisant la F-mesure, sur des tâches de difficulté croissante : pour chaque jeu de donnée, un sous-ensemble de classes connues est sélectionné, puis à partir des classes restantes, un nombre croissant de classes est considéré comme inconnu. Les courbes présentées dans l'article, exprimant la F-mesure en fonction de la difficulté de la tâche, indiquent que G-OpenMax est meilleur que softmax classique et OpenMax de manière consistante sur les deux jeux de données. Cependant, les auteurs indiquent, sans reporter de résultats, que l'évaluation de G-OpenMax sur les mêmes tâches que dans l'article OpenMax (Bendale & Boult, 2015) n'a pas permis d'améliorer significativement les performances. Les auteurs notent que G-OpenMax a plus de difficultés sur les images naturelles, leur hypothèse est que les données générées ne sont pas « plausibles » en tant que classes inconnues.

***OSR with Counterfactual Images* (OSRCI).** Une approche de l'OSR basée sur des principes similaires à ceux de Ge et al. (2017), a été proposée par Neal et al. (2018). Tout comme dans l'approche précédente, un GAN est utilisé pour générer des images inconnues, mais avec une architecture et un

objectif qui diffèrent. Neal et al. (2018) génèrent des images inconnues en utilisant comme base les images connues, pour éviter de générer des images qui seraient trop éloignées des données d'entraînement, afin de palier le manque possible de « plausibilité » des images générées par G-OpenMax.

À la place d'un GAN conditionnel, les auteurs adoptent une architecture de GAN encodeur-décodeur. Cette architecture est composée d'un réseau encodeur E, qui transforme l'image d'entrée $\boldsymbol{X}$ en une représentation latente $\boldsymbol{z} = E(\boldsymbol{X})$, et d'un réseau décodeur G qui, à partir de la représentation latente $\boldsymbol{z}$, reconstruit l'image $\boldsymbol{X}$. Le réseau discriminateur D a toujours pour objectif de déterminer si l'image donnée en entrée est une vraie image ou une image générée par G.

La méthode commence par l'entraînement d'un réseau de classification C_K, entraîné sur les K classes connues. Le GAN encodeur-décodeur est ensuite entraîné sur ces mêmes images. Pour générer des images inconnues, Neal et al. (2018) proposent d'optimiser une représentation latente $\boldsymbol{z}^*$, proche de la représentation $\boldsymbol{z} = E(\boldsymbol{X})$ de l'image réelle (telle que $\|E(\boldsymbol{X}) - \boldsymbol{z}^*\|_2$ soit minimisée, où $\|\cdot\|_2$ représente la norme euclidienne), et pour laquelle le modèle initial C_K prédit une probabilité faible pour chaque classe. En d'autres termes, l'objectif est de générer une image qui ressemble à une image connue, mais qui est située à la frontière des classes connues et inconnues, car l'image n'est pas reconnue par le modèle de classification.

Puis, un réseau de classification C_{K+1} est entraîné sur les images connues et générées, avec l'objectif de différencier $K + 1$ classes, où la classe $K + 1$ correspond à une classe inconnue. Ce réseau est initialisé avec les poids du réseau C_K, afin de simplifier l'apprentissage.

Il y a au total cinq réseaux différents : un pour valider la génération des images inconnues, trois constituants le GAN, et un dernier dédié à l'application en OSR. Il s'agit donc, comme pour G-OpenMax, d'une approche demandant des ressources considérables pendant l'entraînement, là où un unique modèle est utilisé pendant l'inférence.

Contrairement aux approches précédentes, Neal et al. (2018) ont choisi de ne pas utiliser la théorie des valeurs extrêmes pour recalibrer les scores de sortie de leur modèle. Ils postulent, toutefois, que prendre en compte la certitude du modèle concernant les classes connues permettrait d'améliorer les résultats. Ainsi, ils proposent un score d'OSR qui tient compte de la probabilité attribuée à la classe inconnue $P(y_{K+1}|\boldsymbol{X})$ et de la probabilité

maximale obtenue pour une classe connue, selon la formule suivante :

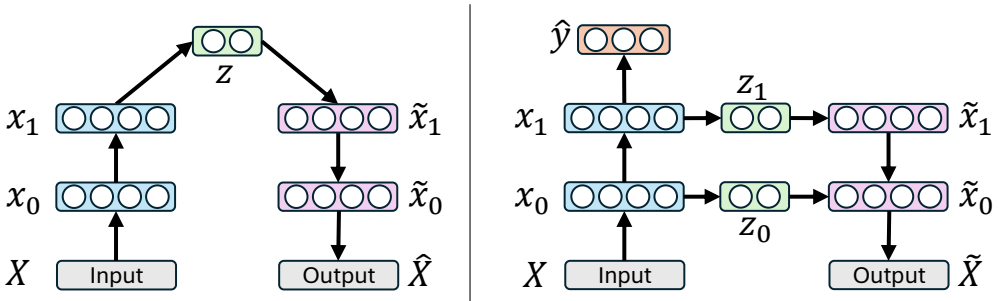
$$S(y \in \mathbb{C}|\boldsymbol{X}) = P(y_{K+1}|\boldsymbol{X}) - \max_{i \leq K} P(y_i|\boldsymbol{X}).$$

Les auteurs affirment que cette modification permet d'améliorer les performances, mais aucun facteur d'amélioration, ni de résultats comparatifs, ne sont présentés.

Neal et al. (2018) ont proposé leur propre benchmark pour l'évaluation de leur méthode OSRCI ; benchmark qui est désormais considéré comme une référence dans l'évaluation des modèles en OSR (c'est pourquoi nous le présentons en détails dans la section 2.1.3). OSRCI a surpassé les méthodes antérieures sur les six tâches proposées dans leur benchmark enregistrant des AUROC de 69.9% et 58.6% sur les tâches les plus exigeantes, CIFAR-10 et TinyImageNet, respectivement. Ces résultats dépassent ceux de la *baseline* utilisant softmax, qui sont de 67.7% et 57.7%. OpenMax et G-OpenMax affichent des performances intermédiaires. OSRCI présente également un léger gain d'Exactitude en *Closed-Set* (ECS) (2% sur CIFAR-10 (Krizhevsky, 2009)) par rapport à softmax classique, ce qui est bénéfique, démontrant qu'il ne semble pas y avoir de compromis lié à l'utilisation d'OSRCI pour faire de la classification en *closed-set*.

***Classification Reconstruction learning for OSR* (CROSR).** Yoshihashi et al. (2019) ont développé une approche nommée CROSR. Les méthodes proposées jusqu'alors pour l'OSR utilisent toutes des réseaux entraînés de manière supervisée pour la classification d'images. Les représentations apprises, les attributs extraits qui sont ensuite utilisés pour la détection des classes inconnues sont donc optimisés spécifiquement pour la *discrimination* des classes connues : selon Yoshihashi et al. (2019), cela n'est pas pertinent pour détecter des classes inconnues. C'est pourquoi ils proposent d'utiliser un apprentissage non supervisé, *via* la reconstruction de l'image d'entrée, en plus d'un apprentissage supervisé utilisant les étiquettes des données. Cela permet de régulariser la représentation latente apprise, afin que le modèle puisse extraire des informations « générales » sur les données, et non pas uniquement discriminatives. Extraire des attributs plus généraux permettrait ainsi de mieux modéliser les classes inconnues, pour mieux les discriminer des classes connues par la suite.

Contrairement aux autres approches, qui se limitent aux scores en sortie du modèle pour la détection des classes inconnues, Yoshihashi et al. (2019) exploitent également les représentations latentes apprises par le modèle à plusieurs niveaux. Ils proposent une architecture spécifique pour la



(a) Architecture d'un auto-encodeur.

(b) Architecture de DHRNet.

Figure 2.10: **Comparaison des architectures d'auto-encodeur et de DHRNet.** (a) L'objectif d'un auto-encodeur est de reconstruire l'image d'entrée X, à partir d'une représentation comprimée z apprise par l'encodeur. (b) L'architecture de DHRNet est composée de deux parties : un réseau encodeur utilisé pour la classification, qui prédit un vecteur y de prédictions et un réseau décodeur pour la reconstruction. Pour un bloc de niveau l dans l'encodeur, la représentation x_l en sortie de ce bloc est comprimée en z_l, puis décomprimée pour être réutilisée dans le réseau de reconstruction.
Source : figure reproduite de Yoshihashi et al. (2019).

reconstruction, appelée *Deep Hierarchical Reconstruction Net* (DHRNet). Cette architecture se distingue d'un auto-encodeur par l'ajout de goulots d'étranglement (*bottlenecks*) liant chaque bloc de l'encodeur à un bloc associé du décodeur. Le but est de comprimer l'information contenue dans la représentation latente en sortie de chaque bloc et de la réutiliser pour la reconstruction de l'image d'entrée. L'architecture DHRNet est présentée en figure 2.10b, et comparée à l'architecture d'un auto-encodeur en figure 2.10a.

Cette architecture est optimisée par la somme de deux termes de perte : un terme évaluant la classification supervisée à partir de $\hat{\boldsymbol{y}}$, et un autre évaluant la qualité de la reconstruction $\tilde{\boldsymbol{X}}$ de l'image d'entrée $\boldsymbol{X}$.

Les auteurs estiment qu'il est probable que des caractéristiques des classes inconnues se dissipent au fil de la progression dans le modèle. Ainsi, l'avantage des goulots d'étranglement $\boldsymbol{z}_l$ situés entre chaque bloc de l'encodeur et du décodeur est qu'ils retiennent des informations sur les représentations latentes à différentes échelles, permettant de limiter la dissolution des attributs appris. Le fonctionnement précis de ces goulots d'étranglement est présenté dans la figure 2.11.

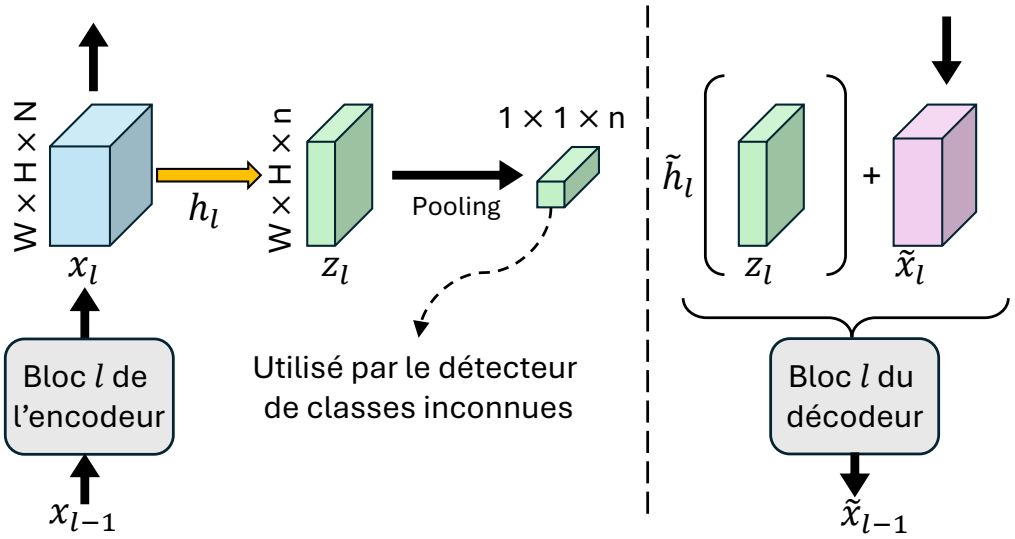


FIGURE 2.11: **Détails d'un niveau de DHRNet.** Dans l'encodeur, la représentation x_l générée par le bloc l est comprimée par une couche de convolution h_l pour obtenir z_l. Dans le décodeur, l'entrée du bloc l reconstruisant $\tilde{x}_l - 1$ est la somme de la représentation z_l projetée vers la dimension adaptée via une convolution $\tilde{h}_l$, et de la sortie de la couche précédente. z_l est encore comprimée via l'application de *global max pooling*, afin d'obtenir un vecteur contenant des informations sur ce qui a été extrait par la couche l. Ce vecteur est ensuite utilisé par le détecteur de classes inconnues.
Source : figure adaptée de Yoshihashi et al. (2019).

Pour détecter les classes inconnues, un nouvel espace de représentation est créé à partir de la concaténation de $\hat{\boldsymbol{y}}$ et de tous les vecteurs $\boldsymbol{z}_l$ après avoir réduit leur dimension par application d'un *global max pooling* réduisant les dimensions spatiales à leur valeur maximale. Les auteurs appliquent ensuite OpenMax sur ce nouvel espace pour recalibrer les scores de sortie et ajouter la prédiction d'une classe inconnue. Si le vecteur recalibré par OpenMax indique que la donnée est connue, alors $\hat{\boldsymbol{y}}$ est utilisé pour déterminer la classe prédite.

CROSR est évalué sur plusieurs jeux de données d'images, en adoptant deux approches différentes pour considérer des classes inconnues. La première approche consiste à séparer les jeux de données en deux ensembles : les classes connues et les classes inconnues. La seconde approche utilise un jeu de données complet comme connu et des données provenant d'autres jeux de données comme classes inconnues. Par exemple, MNIST sert de données connues, tandis qu'Omniglot (Lake et al., 2015), constitué de caractères

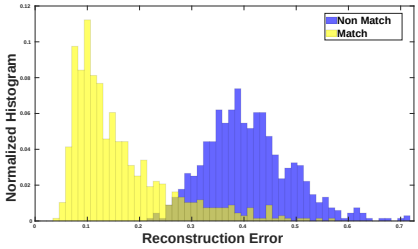
manuscrits de différents langages, représente des données inconnues. Les métriques utilisées sont la F-mesure, et l'AUROC pour la détection des classes inconnues.

CROSR obtient une F-mesure de 0.793 contre 0.78 pour OpenMax et 0.595 pour softmax, pour la détection des données inconnues d'Omniglot, en utilisant l'architecture DHRNet. Les résultats démontrent que l'utilisation d'un réseau permettant la reconstruction hiérarchique confère un avantage sur l'utilisation d'un réseau entraîné de manière supervisée pour la classification uniquement. Pour comparer CROSR à G-OpenMax et OSRCI, tous deux entrainés avec des images de classes inconnues générées par des GANs, les performances ont été évaluées sur trois tâches du benchmark de Neal et al. (2018) en mesurant l'AUROC. CROSR obtient des performances similaires à celles des deux méthodes, avec une AUROC moyenne de 82.6% contre 82.8% pour OSRCI et 82% pour G-OpenMax.

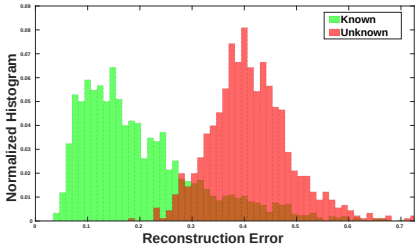
***Class Conditioned Auto-Encoder* (C2AE).** Parallèlement, Oza & Patel (2019) ont travaillé sur une approche utilisant un auto-encodeur, nommée *Class Conditioned Auto-Encoder* (C2AE). Contrairement aux autres méthodes, qui se basent sur les scores de softmax pour la classification en *open-set* (que ce soit en ayant prédit une classe supplémentaire, *via* calibration des scores, ou encore en appliquant un seuil), les auteurs proposent de diviser la tâche d'OSR en deux sous problèmes : d'une part, résoudre la tâche de classification en *closed-set*, et d'autre part, identifier les données inconnues.

La première étape de cette approche consiste à entraîner un encodeur, qui va générer une représentation $\boldsymbol{z}$, et un modèle de classification prenant $\boldsymbol{z}$ en entrée. L'encodeur est entraîné pour la tâche de classification, utilisant la fonction de perte d'entropie croisée, sans opération de reconstruction à cette étape. La deuxième étape consiste à entraîner le décodeur, avec les poids de l'encodeur gelés (non modifiés pendant l'entraînement).

C'est là que réside la particularité de C2AE : le décodeur est entraîné à la fois à correctement reconstruire les images et *à mal les reconstruire*. Pour cela, la représentation $\boldsymbol{z}$ est modifiée en utilisant un vecteur de conditionnement selon une approche nommée *Feature-wise Linear Modulation* (FiLM), proposée par Perez et al. (2018). Il existe deux types de vecteurs de conditionnement : des *match condition vector* (MV), permettant au décodeur de reconstruire l'image correctement ; et des *non-match condition vector* (NMV), qui eux indiquent de mal reconstruire l'image. Il y a un vecteur de conditionnement l_c pour chaque classe c. Pour une image de la classe c,



(a) Erreurs de reconstruction en utilisant des vecteurs de conditionnement *match* (MV) et *non-match* (NMV).



(b) Erreurs de reconstruction pour les classes connues et inconnues sur un jeu de test.

FIGURE 2.12: **Histogrammes normalisés des erreurs de reconstruction mesurées sur le jeu de données SVHN.**
Source : figure issue de l'article de Oza & Patel (2019).

seul le vecteur l_c est un MV, tous les autres sont des NMV. Lors de l'entraînement du décodeur, pour chaque image donnée, la reconstruction est effectuée à partir de la représentation $\boldsymbol{z}$, conditionnée par le MV et par un NMV choisi aléatoirement.

Une fois le décodeur entraîné, deux reconstructions sont effectuées pour chaque image : l'une utilisant le MV associé à la classe de l'image et l'autre en utilisant un NMV choisi aléatoirement. Les erreurs de reconstruction sont ensuite mesurées pour l'ensemble des images reconstruites. La distribution de ces erreurs, mesurées par les auteurs lors de cette étape, est visible sur la figure 2.12a. La figure 2.12b montre la distribution des erreurs de reconstruction pour les classes connues et inconnues sur le jeu de test. Ces deux distributions se sont révélées être très similaires, démontrant que C2AE est capable d'approximer un scénario en *open-set* en reconstruisant des images avec des vecteurs de conditionnement.

Il est ensuite nécessaire de définir un seuil sur l'erreur de reconstruction afin de distinguer les classes connues des classes inconnues. Pour cela, les auteurs appliquent la théorie des valeurs extrêmes pour modéliser les valeurs extrêmes de la queue de droite de la distribution pour les MV et la queue de gauche de la distribution pour les NMV. Ces modèles sont alors utilisés pour déterminer un seuil sur l'erreur de reconstruction minimisant la probabilité d'erreur de classification en *open-set*.

Lors de l'inférence avec C2AE, le modèle de classification réalise une prédiction pour chaque image, tandis que le décodeur génère une image pour *chaque vecteur de conditionnement*, ce qui permet d'obtenir autant d'erreurs de reconstruction que de classes.

Étant donné qu'une image d'une classe connue est correctement reconstruite uniquement lorsque sa représentation est conditionnée par le MV correspondant, les hypothèses de cette approche pour distinguer les classes connues des classes inconnues sont les suivantes :

— face à une classe connue, il devrait y avoir un vecteur de conditionnement correspondant à la classe de l'image originale : une des images doit donc être correctement reconstruite, entraînant une faible erreur de reconstruction pour cette image ;

— face à une classe inconnue, aucun des vecteurs de conditionnement ne devrait correspondre à la classe de l'image, résultant en une mauvaise erreur de reconstruction pour tous les vecteurs de conditionnement.

Enfin, l'erreur de reconstruction minimale parmi toutes les images reconstruites est comparée au seuil d'erreur prédéfini. Si cette erreur dépasse le seuil, l'image est considérée comme inconnue ; sinon, la prédiction du modèle de classification est utilisée.

C2AE a été évaluée sur le benchmark proposé par Neal et al. (2018) et a obtenu les meilleurs résultats sur toutes les expériences, montrant des améliorations significatives sur la plupart des tâches. Notamment, C2AE obtient une AUROC de 89.5% sur la tâche de CIFAR-10, comparé à OSRCI (Neal et al., 2018) qui obtient 69.9%. En moyenne, C2AE obtient une AUROC de 90.8% contre 80.8% pour OSRCI.

Pour conclure, la division de la tâche de l'OSR en classification d'une part et l'identification des classes inconnues de l'autre a permis d'obtenir de bons résultats. De plus, l'apprentissage de la reconstruction à partir d'un vecteur de conditionnement a permis d'approximer correctement le comportement du modèle en *open-set*. Un point notable est que la détection de classes inconnues nécessite de reconstruire autant d'images qu'il y a de classes connues : le temps nécessaire pour tester ou utiliser l'approche est donc augmenté par rapport aux autres méthodes présentées. En revanche, ces dernières sont plus coûteuses à l'entraînement.

2.2.2 Approches de pointe

Après avoir présenté un historique des méthodes appliquées en OSR, nous décrivons ici les approches les plus récentes et obtenant, à notre connaissance, les meilleurs résultats. Les résultats de chacune de ces approches sont récapitulés dans le tableau 2.2 à la fin de cette section et comparés à C2AE, la meilleure approche présentée en section 2.2.1.

OpenHybrid. En 2020, Zhang et al. (2020) ont proposé une approche nommée *OpenHybrid*. Zhang et al. (2020) objectent qu'il n'est pas suffisant de se baser uniquement sur une représentation apprise de manière discriminative sur les classes connues pour ensuite détecter des classes inconnues. Pour remédier à ce problème, ils présentent une approche hybride, qui combine un modèle de classification et un modèle d'estimation de densité pour détecter des données hors-distribution, c'est-à-dire des données inconnues en OSR.

L'estimation de densité consiste à apprendre (estimer) la densité de probabilité d'un ensemble de données, ici des données connues. Autrement dit, la sortie du modèle d'estimation de densité est une probabilité $P(\boldsymbol{X})$, que $\boldsymbol{X}$ ait été échantillonnée de la distribution des données connues. Cette probabilité sert ainsi à estimer directement si une instance est connue ou inconnue. OpenHybrid utilise un modèle d'estimation de densité à base de flots [2].

L'architecture d'OpenHybrid est schématisée dans la figure 2.13. Dans un premier temps, un encodeur apprend une représentation $\boldsymbol{z}$ à partir d'une image $\boldsymbol{X}$. Cette représentation est dite « jointe » car elle est optimisée à la fois pour la classification et pour l'estimation de densité. Le processus d'apprentissage consiste à minimiser deux termes de perte, l'un pour la classification et l'autre pour l'estimation de densité, termes qui sont ensuite additionnés. La particularité d'OpenHybrid, par rapport à d'autres approches utilisant des modèles à base de flots pour la détection d'anomalies, est que le modèle n'est pas appliqué directement dans l'espace des images, mais sur la représentation jointe $\boldsymbol{z}$ apprise par l'encodeur. Lorsqu'il est appliqué directement aux images originales, un modèle à base de flots pourrait être trompé par le fond des images ; par exemple, une donnée inconnue avec un arrière-plan similaire à celui d'une donnée connue pourrait être injustement prédite comme issue de la distribution des données connues.

Afin de déterminer si une instance est connue ou inconnue, un seuil est fixé sur la *log*-probabilité estimée par le modèle à base de flots. Ce seuil est fixé à la *log*-probabilité *minimale* obtenue sur les données d'entraînement (connues), augmentée d'une marge s. Si la probabilité estimée pour une instance est inférieure à ce seuil, l'instance est considérée comme inconnue, sinon la prédiction du classifieur est utilisée.

2. `https://fr.wikipedia.org/wiki/Flot_(mathématiques)`

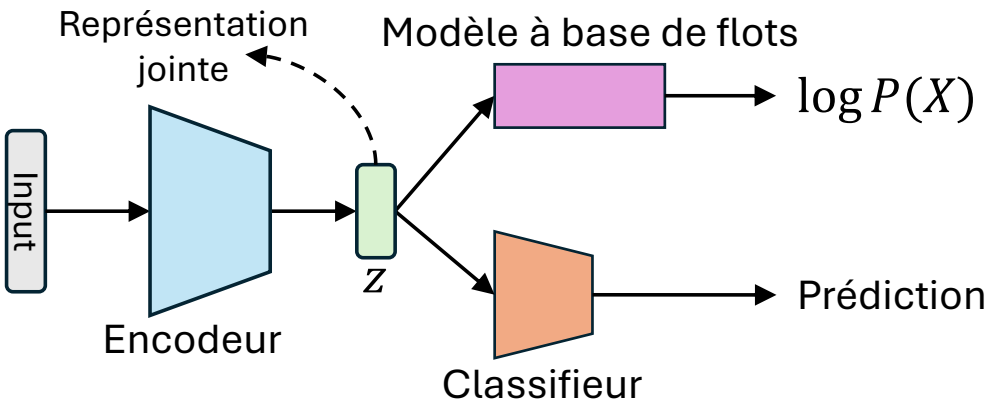


Figure 2.13: **Schéma de l'approche OpenHybrid.** L'architecture est composée d'un encodeur, apprenant une représentation $\boldsymbol{z}$ à partir de laquelle un classifieur prédit la probabilité de chaque classe connue et un estimateur de densité est utilisé pour estimer la log-probabilité que l'image soit issue de la distribution des données connues.
Source : figure adaptée de Zhang et al. (2020).

Pour évaluer leur méthode, les auteurs utilisent le benchmark de Neal et al. (2018) et mesurent les performances à l'aide de l'AUROC (en faisant varier la marge s) et de la F-mesure. OpenHybrid a obtenu les meilleurs résultats sur les six tâches proposées par Neal et al. (2018), avec une amélioration significative par rapport à C2AE. En particulier, OpenHybrid obtient une AUROC de 95.0% sur CIFAR-10, marquant une augmentation de 5.5% par rapport à C2AE, et 79.3% sur TinyImageNet, soit un gain de 4.5%.

Comparé à d'autres méthodes d'estimation de densité basées sur les flots, généralement utilisées pour la détection de données hors distribution, OpenHybrid montre des résultats nettement supérieurs sur le benchmark d'OSR avec une augmentation de l'AUROC d'au moins 20%.

Enfin, dans le but de vérifier l'efficacité de l'apprentissage d'une représentation jointe pour la classification et l'estimation de densité, les performances ont été comparées à celles d'un modèle utilisant une représentation pré-entraînée avec le classifieur uniquement, suivie d'un entraînement du modèle à base de flots à partir de cette représentation figée. Les résultats montrent que l'apprentissage d'une représentation jointe est plus efficace, la F-mesure augmentant sur les 3 jeux de données évalués, notamment de 0.847 pour le modèle pré-entraîné à 0.942 pour le modèle apprenant une représentation jointe sur le jeu de données MNIST.

***Adversarial Reciprocal Points Learning* (ARPL).** Chen et al. (2021) proposent une approche nommée ARPL, qui utilise des prototypes particuliers pour la classification en *open-set*. Kuncheva & Bezdek (1998) définissent un prototype comme étant une moyenne ou le meilleur exemple d'une catégorie, qui peut être utilisé pour représenter la totalité de cette catégorie. D'autres approches parleront plutôt de centre ou d'ancre au lieu de prototype, mais l'idée est la même. Chen et al. (2021) soulignent que les approches qui apprennent la position des prototypes durant l'entraînement ne sont pas adaptées à l'OSR, car ces prototypes peuvent converger vers l'espace où sont représentées les données inconnues, rendant ainsi leur détection plus complexe. L'argument commun à d'autres approches (Ge et al., 2017; Neal et al., 2018) est qu'il ne faut pas seulement modéliser les données connues, mais aussi les données inconnues.

Les auteurs proposent de considérer la classification d'un point de vue différent. Par exemple, face à la question « Qu'est-ce qu'un chat ? », une approche basée sur les prototypes proposerait un point censé représenter globalement la classe *chat*. Chen et al. (2021) suggèrent d'utiliser des prototypes qui ne représentent pas un chat, identifiés comme *non-chat*, afin de définir ce qu'est un chat par *sa différence* à ces points prototypiques. Textuellement, cela revient à dire, « cette image ne correspond à aucun prototype de *non-chat*, donc c'est un *chat* ». Ces prototypes, représentant des données différentes de la vraie classe, sont nommés les *reciprocal points* de cette classe, et permettraient de modéliser l'espace des données inconnues sans nécessiter l'utilisation de ces dernières durant l'entraînement, car la majorité des données inconnues devraient être similaires à des représentations de *non-chat*.

Un *reciprocal point* P^k pour une classe k est considéré comme la représentation latente de toutes les autres données *connues et inconnues*. Les *reciprocal points* sont initialisés de manière uniforme dans l'espace de représentation, puis leurs coordonnées sont apprises pendant l'entraînement via rétropropagation de l'erreur. La contrainte imposée est que la distance entre les données de la classe k et leur *reciprocal point* P^k doit être supérieure à la distance maximale observée entre P^k et *toutes les autres données*.

Une contrainte de marge « adversariale » est ajoutée afin de limiter l'*open space*, l'espace de représentation des données inconnues, ce qui permet donc de limiter le risque de classification en *open space*. Cette contrainte force les données d'autre classes que k (négatives par rapport à la classe k et inconnues) à être au maximum à une distance R du point P^k, tandis que les données de la classe k doivent être situées au moins à une distance R de

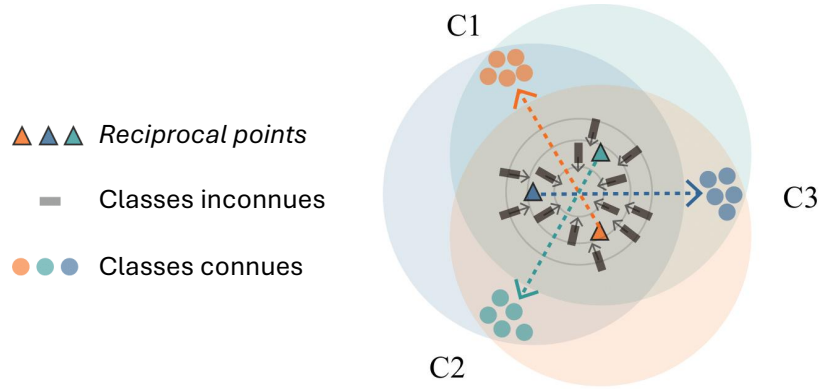


Figure 2.14: **Schéma de l'espace de représentation d'ARPL.** Considérons, par exemple, la classe C1 en orange : son *reciprocal point* associé est représenté par le triangle orange. Toutes les autres données sont représentées dans le disque de rayon R orange, autour du *reciprocal point*. Chaque classe connue est représentée à une distance d'au moins R de son *reciprocal point*. Les données inconnues sont représentées à proximité de tous les *reciprocal points*. Une donnée est classée en fonction du *reciprocal point* le plus éloigné.
Source : figure issue de Chen et al. (2021).

P^k. Une telle configuration de l'espace de représentation est illustrée dans la figure 2.14. Cette contrainte permet ensuite de classer les instances en leur attribuant la classe du *reciprocal point* le plus éloigné.

ARPL a été évaluée sur le benchmark de Neal et al. (2018) en mesurant l'AUROC et a obtenu des performances supérieures à celles de C2AE (Oza & Patel, 2019), la meilleure approche à laquelle les auteurs se comparent, sur toutes les tâches. Cependant, l'approche n'a pas été comparée à OpenHybrid (Zhang et al., 2020) dans l'article, bien que cette dernière montre des performances équivalentes ou supérieures à celles d'ARPL.

Pour améliorer davantage leurs résultats, Chen et al. (2021) ont proposé d'utiliser un GAN pour générer des *confusing samples* (CS). Ces données générées par le générateur perturbent le discriminateur, en étant à la fois inconnues pour le modèle de classification (représentées proches de tous les *reciprocal points*) et identifiées comme réelles par le discriminateur. Ces CS sont ensuite utilisés comme données *inconnues connues* pour améliorer l'apprentissage du modèle, cette version se nomme ARPL+CS.

ARPL+CS a permis d'améliorer les performances d'ARPL de 1 à 2% sur le benchmark. Toutefois, cette approche est toujours battue par OpenHybrid, qui surpasse ARPL+CS de 4% et 1.2% sur CIFAR-10 et TinyImageNet, respectivement.

Une *baseline* améliorée. Bien que la plupart des auteurs précédents aient soutenu qu'un modèle discriminatif n'est pas adapté à la détection des classes inconnues, Vaze et al. (2022) ont souhaité mettre cette affirmation à l'épreuve.

À cet effet, Vaze et al. (2022) ont mené deux expériences démontrant qu'il existe une corrélation entre l'exactitude obtenue par un modèle entraîné avec l'entropie croisée et l'AUROC obtenue par ce même modèle sur la tâche de détection de classes inconnues. La première expérience a consisté à évaluer les performances en OSR d'une seule architecture et de plusieurs méthodes. Entraînés sur plusieurs tâches du benchmark de Neal et al. (2018), les modèles ont révélé une corrélation de Pearson de $\rho = 0.95$ entre l'exactitude et l'AUROC.

La deuxième expérience a évalué différentes configurations d'architectures telles, que VGG (Simonyan & Zisserman, 2015), ResNets (He et al., 2016) et les *Vision Transformers* (Dosovitskiy et al., 2021a) à partir de modèles pré-entraînés sur ImageNet (Deng et al., 2009). Les données inconnues à détecter sont issues du jeu de données ImageNet-21K (Ridnik et al., 2021), contenant plus de classes qu'ImageNet. À partir d'ImageNet-21K, deux jeux de données de détection de classes inconnues ont été proposés par Vaze et al. (2022) : l'un contenant des classes inconnues « faciles » à détecter, car sémantiquement éloignées des classes connues ; et l'autre contenant des classes inconnues « difficiles » à détecter, car proches des classes connues. Pour les données faciles, la corrélation obtenue est de $\rho = 0.63$, tandis que sur les données difficiles, la corrélation est de $\rho = 0.88$. Ces résultats plus faibles sont expliqués par les grandes différences entre les architectures. En considérant à part l'ensemble des ResNets évalués, les corrélations sont de $\rho = 1.00$ et $\rho = 0.99$ pour les données faciles et difficiles, respectivement.

En réponse à l'observation de cette corrélation, Vaze et al. (2022) ont proposé d'utiliser plusieurs techniques de l'état de l'art en classification d'images afin d'améliorer l'exactitude d'un modèle basé sur l'architecture de Neal et al. (2018) (section 2.1.4). Ces techniques consistent à : entraîner le modèle plus longtemps (durant 600 époques), utiliser une meilleure technique d'augmentation de données nommée *RandAugment* (Cubuk et al., 2020), appliquer du *label smoothing* (Szegedy et al., 2016), et utiliser un taux

d'apprentissage variable au cours de l'entraînement (Loshchilov & Hutter, 2016). Le *label smoothing* modifie le vecteur one-hot de la classe cible en réduisant légèrement la certitude attribuée à la classe correcte et en redistribuant équitablement cette certitude parmi les autres classes afin que la somme du vecteur soit toujours égale à 1. Cela permet d'éviter que le modèle soit biaisé vers la réalisation de prédictions avec une haute certitude.

D'autres travaux, comme ceux de Chen et al. (2021), ont déjà montré que la norme du vecteur de logits $\boldsymbol{o}$ en sortie des modèles est plus faible pour les données inconnues que pour les données connues. Pour intégrer cette donnée, Vaze et al. (2022) proposent d'utiliser directement la valeur maximale des logits en tant que score d'OSR, $S(y \in \mathbb{C}|\boldsymbol{X}) = \max \boldsymbol{o}$: cette mesure est nommée le *Maximum Logit Score* (MLS). Ce score est utilisé à la place du *Maximum Softmax Probability* (MSP), habituellement utilisé pour les expériences de contrôle impliquant un modèle entraîné avec la fonction de perte d'entropie croisée (*baseline*). Pour rappel, le MSP correspond à la probabilité maximale après application de softmax au vecteur de logits, $S(y \in \mathbb{C}|\boldsymbol{X}) = \max \text{softmax}(\boldsymbol{o})$, or softmax a pour effet de normaliser les logits et donc fait perdre l'information apportée par la norme.

Cette approche a permis à la *baseline* utilisant le MLS de surpasser de 15.6% en moyenne les résultats initialement reportés par Neal et al. (2018) pour la *baseline* utilisant la MSP (*baseline* dans laquelle les modèles sont entraînés pendant seulement 30 époques), et de dépasser également de 0.7% en moyenne l'approche ARPL+CS de Chen et al. (2021) sur les six tâches de détection du benchmark de Neal et al. (2018).

En appliquant les mêmes améliorations à ARPL+CS, les performances de cette méthode ont pu être améliorées. MLS et la version améliorée d'ARPL+CS présentent des performances très proches, avec une différence maximale de 0.3% sur les différents jeux de données ; aucune des deux versions ne se démarque de manière consistante comme supérieure à l'autre. Sur TinyImageNet, la différence est tout de même de 0.5% en faveur de MLS.

Ces expériences démontrent qu'un modèle utilisant l'entropie croisée, s'il est bien entraîné, est capable d'exhiber des performances comparables à celles d'approches plus complexes, conçues spécifiquement pour résoudre le problème de l'OSR. C'est pourquoi nous considérons que Vaze et al. (2022) proposent une *baseline* améliorée. Cependant, une telle approche ne limite pas le risque de classification en *open-space* (section 1.2.2), car si le

modèle réalise une prédiction trop certaine (où le logit maximal dépasse largement le seuil) pour une image inconnue, cette prédiction est tout de même acceptée alors qu'il pourrait s'agit d'une anomalie.

***Contrastive OSR* (ConOSR).** Xu et al. (2023) ont développé une approche appelée ConOSR en partant de l'hypothèse qu'une meilleure technique d'apprentissage de représentation pourrait améliorer les résultats en OSR : en extrayant des attributs assez représentatifs des classes connues, ces attributs ne devraient pas s'activer en présence de classes inconnues.

Pour obtenir une représentation adaptée, les auteurs proposent d'utiliser du *contrastive learning* supervisé. Le *contrastive learning* est une méthode non supervisée qui consiste à entraîner un modèle à représenter deux versions augmentées d'une même image, proches l'une de l'autre et éloignées des autres images. Dans la variante du *contrastive learning* supervisé, des étiquettes sont à disposition, ce qui permet de rapprocher tous les exemples d'une même classe les uns des autres, tout en éloignant ces représentations des exemples négatifs (c.-à-d. des images d'autres classes). En pratique, un réseau encodeur apprend une représentation $\boldsymbol{z}$ d'une entrée $\boldsymbol{X}$, et un réseau de projection transforme $\boldsymbol{z}$ en un vecteur de projection $\boldsymbol{h}$, qui est ensuite utilisé pour calculer la perte *contrastive*.

La spécificité de ConOSR est l'utilisation de données générées grâce à la méthode d'augmentation de données *mixup* (Zhang et al., 2018) pendant l'entraînement. Cette méthode utilise deux images x_1 et x_2 pour créer une nouvelle image x en les additionnant : $x = \gamma x_1 + (1 - \gamma)x_2$, avec $0 < \gamma < 1$. Puisque cette nouvelle image contient deux classes, les vecteurs *one-hot* des étiquettes sont mélangés selon le même principe. Ces données représentent des classes « vagues », avec une « sémantique ambiguë », afin de représenter des classes *inconnues connues*.

L'utilisation du *label smoothing* et du *mixup* pour augmenter le nombre de données négatives pose un problème en *contrastive learning*, car ces instances sont identifiées comme appartenant à plusieurs classes par leurs vecteurs d'étiquettes (qui ne sont plus des *one-hot*), ce qui ne permet plus de déterminer si les instances doivent être projetées proches les unes des autres ou non. Pour résoudre ce problème, Xu et al. (2023) proposent une fonction de perte adaptée, qui calcule une similarité entre deux vecteurs d'étiquettes pour déterminer à quel point les instances associées doivent être projetées à proximité l'une de l'autre ou éloignées.

Le processus d'entraînement mis en place est le suivant : la première étape consiste à entraîner l'encodeur et le réseau de projection en utilisant du *contrastive learning*. Pour chaque image du jeu de données d'entraînement $\mathbb{D}_{\text{tr}}$, deux images augmentées sont générées en utilisant *RandAugment*, et du *label smoothing* est appliqué aux étiquettes. L'ensemble des images augmentées est nommé $\mathbb{D}_{\text{aug}}$, c'est l'ensemble des données connues. Ensuite, un jeu de données d'exemples virtuels, $\mathbb{D}_{\text{mix}}$, est généré avec *mixup*, en sélectionnant à chaque fois deux images aléatoires de $\mathbb{D}_{\text{aug}}$, pour constituer l'ensemble des données *inconnues, vagues*. Une fois que l'entraînement de l'encodeur a convergé, ses poids sont gelés. Un modèle de classification utilisant la représentation $\boldsymbol{z}$ en entrée est entraîné avec la perte d'entropie croisée, avec des données augmentées *via RandAugment*, et en utilisant du *label smoothing*.

L'approche mise en place pour détecter les classes inconnues se distingue des autres approches présentées précédemment. À partir de l'encodeur et du modèle de classification, pour chaque donnée correctement classée, la probabilité maximale prédite par le réseau est récupérée : pour chaque classe c, un ensemble $\mathbb{T}_c$ de probabilités maximales est donc constitué. Un seuil de rejet ϵ sur la probabilité est ensuite fixé pour chaque classe, en prenant en compte le percentile de valeur λ de chaque ensemble. La valeur $\lambda = 5$ est choisie pour maintenir un taux de faux négatifs de 5% sur le jeu d'entraînement, mais ce taux est en pratique plus élevé sur le jeu de test. Comme l'évaluation des performances utilise l'AUROC, le paramètre varié pour obtenir les différents seuils est λ. Les auteurs notent que les résultats sont très sensibles à la valeur de λ.

Évaluée sur le benchmark de Neal et al. (2018), l'approche ConOSR obtient de meilleurs résultats qu'ARPL+CS, la meilleure méthode à laquelle les auteurs se sont comparés, avec au moins 2% de gains sur l'AUROC. Les résultats de Vaze et al. (2022) n'ont pas été reportés, bien que l'approche ait été évoquée. Nous avons réalisé nous même une comparaison, et ConOSR obtient moins de 1% de gain par rapport à la *baseline* de Vaze et al. (2022), excepté sur TinyImageNet où la *baseline* est plus performante de 3%.

Selon Xu et al. (2023), un avantage du *contrastive learning* est que le réseau apprend à se focaliser sur les attributs les plus distinctifs des classes connues, qui ont moins de probabilité d'exister dans les classes inconnues et donc ne s'activeront pas. En revanche, des expériences supplémentaires menées par les auteurs ont démontré que les attributs extraits par leur méthode ne permettent pas d'être généralisés à d'autres domaines, contrairement à des attributs extraits par un réseau entraîné avec entropie croisée.

Un inconvénient de ConOSR réside dans sa consommation de ressources de calcul. Le *contrastive learning* nécessite davantage d'époques d'entraînement, et la perte proposée requiert plus de mémoire pour être efficace, car chaque batch doit contenir quelques exemples positifs de chaque classe afin que la répartition des classes dans l'espace de représentation soit efficace. Il faut aussi générer deux fois plus d'augmentations avec *RandAugment*, l'utilisation de *mixup* crée ensuite deux fois plus d'exemples que dans le jeu de données initial.

***Deep Compact HyperSphere* (DCHS)** La dernière méthode que nous présentons est DCHS, proposé par Cevikalp et al. (2023). DCHS a pour objectif d'être utilisable dans deux domaines : la détection d'anomalies et l'OSR. Nous nous focalisons ici sur l'étude réalisée en OSR. L'idée proposée est d'obtenir des zones de prédictions compactes autour de centres associés à chaque classe dans l'espace de représentation du réseau, de sorte que tout point représenté en dehors de ces zones soit considéré comme inconnu.

Contrairement à des approches comme G-OpenMax (Ge et al., 2017) ou OSRCI (Neal et al., 2018) qui utilisent des données *inconnues connues* générées, DCHS se distingue en utilisant des données *inconnues connues réelles*. En effet, pour entraîner leur modèle, Cevikalp et al. (2023) utilisent le jeu de données *80 millions tiny images* (Torralba et al., 2008) en tant qu'exemples de données inconnues. Ce jeu de données contient 80 millions d'images de dimensions 32×32, récupérées sur internet à partir des résultats de recherche pour 53 464 noms différents. Un problème avec un tel jeu de données considéré comme inconnu est qu'il peut contenir des classes connues, ce qui pénaliserait l'apprentissage puisque que l'objectif est d'éloigner les données inconnues des données connues. Les auteurs parlent ainsi d'un jeu de données « bruité ».

DCHS utilise trois termes de perte qui sont pondérés et additionnés pour son optimisation. Pour répondre au problème du bruit dans l'ensemble de données inconnues, les auteurs proposent un terme de perte robuste au bruit appelé *robust ramp loss*, pour permettre de séparer d'une certaine marge les données inconnues des données connues. Cependant, si une donnée identifiée comme inconnue est représentée proche d'un centre existant (selon un seuil prédéfini), alors la perte retournée par ce terme pour cette instance vaut 0, afin de ne pas détériorer l'apprentissage en forçant une donnée potentiellement connue à être éloignée de son centre. Le deuxième terme de perte a pour objectif d'éloigner d'une marge définie les données connues des centres qui ne leur correspondent pas, afin de faciliter la tâche

de classification. Enfin, un dernier terme contraint les données connues à être représentées proches de leur centre associé. Les distances sont mesurées avec la distance euclidienne.

La méthode proposée consiste à apprendre des centres pour chaque classe à partir des données d'entraînement. L'intérêt est de permettre l'apprentissage d'une représentation sémantique, en représentant les centres des classes sémantiquement proches à proximité dans l'espace de représentation. Le mécanisme d'apprentissage des centres consiste à mettre à jour les coordonnées des centres après chaque *batch* de données. Pour chaque classe, un centre moyen est recalculé à partir des représentations de cette classe et relativement au centre défini lors du *batch* précédent. À cause des contraintes de marge entre les représentations d'instances de classes différentes et d'instances inconnues, un deuxième terme a pour but d'éloigner les centres en multipliant les coordonnées, tant que les contraintes de marge ne sont pas respectées.

La zone où les prédictions sont acceptées est ensuite représentée par une hypersphère autour des centres, toute donnée représentée en dehors de l'hypersphère est considérée comme inconnue. Le rayon de l'hypersphère peut être fixé en utilisant la courbe ROC.

DCHS a été évalué sur le benchmark de Neal et al. (2018) et obtient les meilleurs résultats sur trois des six tâches, dont TinyImageNet. Notamment, sur CIFAR+10 l'AUROC obtenue est de 99.2%. Cependant, l'obtention des meilleurs résultats sur TinyImageNet peut être liée à l'utilisation d'un réseau de neurones plus profond en lieu et place du réseau classiquement associé au benchmark.

Pour valider l'efficacité de la *robust ramp loss* proposée, une expérience consistant à ajouter progressivement des données connues à l'ensemble de données inconnues montre que les performances diminuent faiblement dans ce scénario. En ajoutant 40% des données connues au jeu de données inconnues, l'AUROC n'a diminué que de 1.7% et la précision de 1.2%. La perte proposée permet donc effectivement de mitiger l'impact, sur l'apprentissage, des données connues qui pourraient se trouver dans l'ensemble de données inconnues.

Pour montrer que les représentations obtenues portent une sémantique, Cevikalp et al. (2023) ont représenté chaque classe du jeu de données CIFAR-10 par leur centre tel que déterminé par DCHS, et mesurent la distance entre chaque centre pour obtenir une matrice de similarité entre classes. Une classification hiérarchique réalisée à partir de cette matrice de

similarité montre que les séparations à chaque niveau de l'arbre sont sémantiquement interprétables : par exemple, le nœud racine sépare les véhicules des animaux ; les camions sont proches des voitures dans la hiérarchie ; de même que les chiens sont proches des chats.

Malgré les performances de pointe obtenues par DCHS, l'utilisation de données inconnues provenant du jeu de données *80 millions tiny images* pourrait biaiser la méthode. En effet, il existe une probabilité que lors de l'évaluation, des images de classes inconnues du jeu de test aient déjà été vues par le réseau pendant l'entraînement si ces classes étaient représentées dans le jeu de données *80 millions tiny images*. Le réseau a donc potentiellement appris à représenter ces données loin des classes connues, expliquant ainsi les performances très élevées. Il est notable que les meilleurs résultats sont obtenus sur les jeux de données d'images naturelles (du même type que *80 millions tiny images*) et non pas sur les chiffres. Cependant, il est aussi possible que l'entraînement avec autant de données supplémentaires ait permis d'apprendre à extraire de meilleurs attributs pour différencier les classes connues, et a amélioré les performances de détection des classes inconnues.

Cevikalp et al. (2023) adoptent une perspective intéressante sur l'utilisation des données *inconnues connues* : ils ne considèrent pas qu'il s'agit d'une seule et même classe (qui devrait par exemple être représentée à l'origine de l'espace de représentation), mais comme des échantillons qui approximent des classes inconnues qui peuvent être répartis dans l'espace de représentation, du moment que ceux-ci sont en dehors des hypersphères.

Récapitulatif des résultats. Pour faciliter la comparaison de toutes les approches de pointe présentées, le tableau 2.2 récapitule les performances de chaque méthode sur le benchmark de Neal et al. (2018). Parmi les méthodes présentées, DCHS (Cevikalp et al., 2023) performe particulièrement sur les images naturelles, tandis que ConOSR se distingue sur les chiffres et est proche de DCHS sur les images naturelles. La *baseline* proposée par Vaze et al. (2022) se distingue également, se classant à la deuxième ou troisième place selon les tâches du benchmark.

Il convient de noter que les résultats peuvent varier en fonction des séparations des jeux de données utilisées pour créer les différentes tâches du benchmark. Par exemple, une séparation contenant les classes chat et chien dans son ensemble de données connues est plus facile à traiter qu'une séparation où l'une des deux classes est dans l'ensemble des données inconnues. Vaze et al. (2022) précisent bien avoir utilisé les mêmes séparations que

Méthode	MNIST	SVHN	CIFAR-10	CIFAR+10	CIFAR+50	TinyImageNet
C2AE (Oza & Patel, 2019)	98.9	92.2	89.5	95.5	93.7	74.8
OpenHybrid (Zhang et al., 2020)	99.5	94.7	**95.0**	96.2	95.5	79.3
ARPL (Chen et al., 2021)	*99.6*	96.3	90.1	96.5	94.3	76.2
ARPL+CS (Chen et al., 2021)	**99.7**	96.7	91.0	97.1	95.1	78.2
MLS (Vaze et al., 2022)	99.3	*97.1*	93.6	97.9	96.5	*83.0*
ConOSR (Xu et al., 2023)	**99.7**	**99.1**	94.2	*98.1*	*97.3*	80.9
DCHS (Cevikalp et al., 2023)	*99.6*	94.5	*94.7*	**99.2**	**98.5**	**83.8**

TABLEAU 2.2: **Performances des approches de pointe en OSR.** Score AUROC (%) mesuré pour la tâche de détection des classes inconnues, moyenné sur cinq séparations de classes connues/inconnues. Les meilleures performances sont indiquées en gras et les deuxièmes meilleures en italique.

Chen et al. (2021). En revanche Cevikalp et al. (2023) ne précisent pas les séparations utilisées et décrivent un protocole de séparation différent pour CIFAR+10 et CIFAR+50.

2.2.3 Class Anchor Clustering

Nous présentons en détails l'approche *Class Anchor Clustering* (CAC) proposée par Miller et al. (2021), en raison de sa proximité avec la méthode que nous proposons dans le chapitre 3. Comparativement aux autres approches, CAC a pour avantage de fonctionner à partir d'une architecture basique de réseau de neurones : aucun GAN ou auto-encodeur n'est utilisé. Le modèle est entraîné uniquement à partir des données connues. CAC s'inspire de Wen et al. (2016), qui présentent une fonction de perte appliquée au domaine de la reconnaissance faciale, appelée *Center Loss*, où un centre pour chaque classe est appris pendant l'entraînement, dans l'espace de représentation latent. Les représentations d'instances d'une même classe sont optimisées pour être représentées de manière compacte autour du centre appris. Plutôt qu'apprendre ces centres, Miller et al. (2021) proposent de les fixer préalablement et parleront alors d'*ancres*, situées à des coordonnées prédéfinies dans l'espace des logits, en sortie du réseau.

Ces ancres ont pour coordonnées des vecteurs *one-hot*, multipliés par une amplitude α. Par exemple, s'il y a 3 classes à prédire, les ancres $\mathcal{A}^{(c)}$, avec c l'indice de la classe, sont placées aux coordonnées :

$$\mathcal{A}^{(1)} = (\alpha, 0, 0), \quad \mathcal{A}^{(2)} = (0, \alpha, 0), \quad \mathcal{A}^{(3)} = (0, 0, \alpha).$$

Considérons maintenant le cas général où un modèle C prédit un vecteur de logits $\boldsymbol{o} = C(\boldsymbol{X})$ pour une instance $\boldsymbol{X}$ appartenant à la classe y, dans un problème de classification à K classes. Les distances $\boldsymbol{d}$ entre les logits

prédits et chaque ancre sont calculées en utilisant la distance euclidienne :

$$\boldsymbol{d} = \text{Eucl}(\boldsymbol{o}, \mathcal{A}) = \left(\|\boldsymbol{o} - \mathcal{A}^{(1)}\|_2, \ldots, \|\boldsymbol{o} - \mathcal{A}^{(K)}\|_2\right).$$

Afin de représenter les instances d'une classe autour de leur ancre, CAC utilise une fonction de perte composée de deux termes. Le premier terme, $\mathcal{L}_T$, est une fonction qui maximise la *marge* entre (1) la distance du logit prédit par rapport à son ancre, et (2) la distance du logit prédit par rapport aux autres ancres. L'effet de ce terme de perte, illustré dans la figure 2.15, est quantifié par l'équation suivante :

$$\mathcal{L}_T(\boldsymbol{X}, y) = \log\left(1 + \sum_{j \neq y}^{K} e^{d_y - d_j}\right).$$

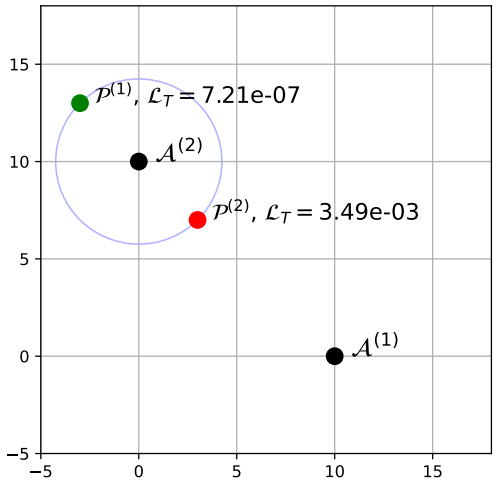


FIGURE 2.15: **Effet de la fonction de perte $\mathcal{L}_T$ de CAC.** Considérons un cas simple de deux classes, avec leurs ancres respectives $\mathcal{A}^{(1)}$ et $\mathcal{A}^{(2)}$. Les points $\mathcal{P}^1$ et $\mathcal{P}^2$ sont les représentations d'instances appartenant à la classe 2, et sont à égale distance de leur ancre $\mathcal{A}^{(2)}$. Cependant, la perte $\mathcal{L}_T$ attribuée à $\mathcal{P}^2$ est plus élevée, car sa distance par rapport aux autres ancres est plus faible.

Le deuxième terme, $\mathcal{L}_A$, vise à rapprocher les instances de leur ancre en minimisant la distance euclidienne (notée $\|\cdot\|_2$) entre le vecteur de logits prédit, $\boldsymbol{o}$, et l'ancre de la classe à prédire :

$$\mathcal{L}_A(\boldsymbol{X}, y) = d_y = \|\boldsymbol{o} - \mathcal{A}^{(y)}\|_2.$$

La perte de $\mathcal{L}_A$ est pondérée par un hyperparamètre λ, afin de donner plus ou moins d'importance à cet objectif par rapport à la perte $\mathcal{L}_T$. La fonction de perte totale est donc définie par :

$$\mathcal{L}_{CAC}(\boldsymbol{X}, y) = \mathcal{L}_T(\boldsymbol{X}, y) + \lambda \mathcal{L}_A(\boldsymbol{X}, y).$$

Les expériences menées par Miller et al. (2021) révèlent que CAC n'est pas sensible aux variations des valeurs de α et λ sur les intervalles évalués. Ces valeurs ont été fixées à $\alpha = 10$ et $\lambda = 0.1$. La valeur de λ pourrait donner l'impression que l'objectif d'approcher les instances de leurs ancres a moins d'importance. Cependant, les deux fonctions de perte n'ont pas le même ordre de grandeur, λ permet donc d'équilibrer la priorité des deux termes.

Une fois le réseau entraîné, les ancres sont ajustées pour correspondre à la moyenne des représentations des instances *correctement classées* de chaque classe. Enfin, pour faire une prédiction, un score de rejet γ_c est calculé pour chaque classe c, selon la formule :

$$\gamma_c = d_c \cdot (1 - \text{softmin}(d_c)),$$

où $\text{softmin}(x) = \text{softmax}(-x)$. La classe prédite est déterminée par $\hat{y} = \arg\min \gamma$, si $\min \gamma \leq \theta$, où θ est un seuil fixé. Sinon, la prédiction est rejetée et la classe est considérée comme inconnue.

CAC a été évaluée sur le benchmark de Neal et al. (2018), présenté à la section 2.1.3, et a obtenu les meilleurs résultats par rapport aux méthodes d'OSR basées sur une mesure de distance (Bendale & Boult, 2015; Ge et al., 2017; Yoshihashi et al., 2019). Depuis sa publication, CAC a été dépassée par plusieurs approches utilisant également une distance pendant l'entraînement et le test (Chen et al., 2021; Cevikalp et al., 2023). Sur les six tâches du benchmark, CAC obtient une AUROC moyenne de 87.3%, ce qui est inférieur de près de 6% aux performances des approches plus récentes, notamment ARPL+CS obtenant en moyenne 93.0% d'AUROC. Les résultats de CAC comparés à ceux des deux meilleures approches présentées dans la section 2.2.2 sont présentés dans le tableau 2.3.

En conclusion, Miller et al. (2021) ont démontré que fixer des ancres pour chaque classe s'avère plus efficace que de les apprendre, à l'instar de *Center Loss*, du moins quand *Center Loss* est appliqué dans le cadre de la classification d'objets. L'expérience inverse consistant à évaluer CAC sur des tâches de reconnaissance faciale n'a pas été réalisée. Comparativement à *Center Loss*, Miller et al. (2021) notent que la fixation des ancres contribue

Méthode	MNIST	SVHN	CIFAR-10	CIFAR+10	CIFAR+50	TinyImageNet
ConOSR (Xu et al., 2023)	**99.7**	**99.1**	94.2	98.1	97.3	80.9
DCHS (Cevikalp et al., 2023)	99.6	94.5	**94.7**	**99.2**	**98.5**	**83.8**
CAC (Miller et al., 2021)	99.1	94.1	80.1	87.7	87.0	76.0

TABLEAU 2.3: **Performances de CAC sur le benchmark de Neal et al. (2018).** Score AUROC (%) mesuré pour la tâche de détection des classes inconnues, moyenné sur cinq séparations de classes connues et inconnues. Les meilleures performances sont indiquées en gras.

à un apprentissage plus stable, plus rapide, et plus robuste face à un grand nombre de classes. Néanmoins, l'approche plus récente DCHS (Cevikalp et al., 2023) utilise des centres appris et obtient de meilleurs résultats, mais des données *inconnues connues* ont été utilisées ce qui n'est pas le cas pour CAC.

Toutefois, Miller et al. (2021) indiquent que CAC ne permet pas d'apprendre des représentations sémantiques. Ils soulèvent l'hypothèse qu'apprendre une représentation plus sémantique permettrait d'améliorer les performances en OSR. Toutes les ancres étant situées à égale distance, la distance entre des instances de classes sémantiquement proches sera similaire à la distance entre des instances de classes sémantiquement éloignées. En réponse à cette lacune, nous introduisons DIFAIR (*DIFferentiAted Images Representations*) (chapitre 3), une méthode visant à apprendre des représentations plus flexibles, pour améliorer la sémantique de l'espace de représentation.

2.2.4 Récapitulatif des méthodes et positionnement

Chaque méthode abordée dans les sections 2.2.1, 2.2.2 et 2.2.3 présente des spécificités. Afin d'avoir une vision plus globale et facilier la comparaison, le tableau 2.4 récapitule les caractéristiques de chaque méthode.

Nous les comparons selon les caractéristiques suivantes : l'utilisation de centres ou d'ancres, l'utilisation d'une mesure de distance lors de l'entraînement et lors de l'inférence, l'utilisation de données *inconnues connues* pendant l'entraînement, le découpage de la tâche d'OSR en classification et détection des données inconnues, l'utilisation de *contrastive learning*, l'utilisation d'une approche de reconstruction d'image, l'utilisation de la théorie des valeurs extrêmes, et l'utilisation de l'entropie croisée.

Méthodes	**OpenMax**	**G-OpenMax**	**OSRCI**	**CROSR**	**C2AE**	**OpenHybrid**	**ARPL+CS**	**MLS**	**ConOSR**	**DCHS**	**CAC**
Centres, Ancres, Prototypes	✓ [a]	✓ [a]		✓ [a]			✓ [b]			✓ appris	✓ fixés
Distance utilisée lors de l'entraînement							✓			✓	✓
Distance utilisée lors du test	✓	✓		✓	✓ [c]		✓			✓	✓
Données inconnues additionnelles		✓	✓				✓		✓	✓ [d]	
Modèle de classification + Module de détection				✓	✓	✓					
Contrastive learning									✓		
Reconstruction				✓ [e]	✓						
Théorie des valeurs extrêmes	✓	✓		✓	✓						
Entropie croisée (utilisée au moins une fois)	✓	✓	✓	✓	✓	✓		✓	✓		

TABLEAU 2.4: **Récapitulatif des caractéristiques des méthodes d'OSR présentées.**

a. Centres calculés à partir des représentations une fois le modèle entraîné.
b. *Reciprocal points* : les données connues sont éloignées de ces points et les données inconnues en sont proches.
c. Distance utilisée pour mesurer l'erreur de reconstruction.
d. Données réelles, non générées.
e. La reconstruction n'est pas directement utilisée, seulement les représentations intermédiaires aidant à la reconstruction.

La plupart des approches présentées estiment que pour les tâches d'OSR il n'est pas approprié d'utiliser des modèles entraînés uniquement à discriminer les classes connues grâce à l'entropie croisée, car les attributs extraits seraient spécifiques aux classes connues et ne permettraient donc pas de modéliser correctement les classes inconnues. Il faudrait par exemple utiliser des méthodes de reconstruction (c'est-à-dire un objectif non discriminant), pour extraire des attributs plus généraux, et donc plus aptes à modéliser les données inconnues et mieux les distinguer des données connues. Les résultats de Vaze et al. (2022) montrent néanmoins qu'il est possible d'obtenir de très bonnes performances en OSR à partir d'une approche purement discriminative, en entraînant un modèle avec l'entropie croisée.

Dans l'approche que nous proposons, présentée dans le chapitre 3, notre objectif est de permettre la discrimination des classes connues, mais sans faire appel à l'entropie croisée. De plus, nous choisissons aussi de ne pas modéliser explicitement les données inconnues. En effet, à l'exception de DCHS qui est entraîné avec des données inconnues non générées, les autres approches nécessitent d'entraîner plusieurs modèles (auto-encodeur, GAN) alors que nous recherchons une parcimonie en termes de complexité et de ressources de calcul, à l'instar des approches discriminantes.

De manière similaire à DCHS nous focalisons l'optimisation du modèle sur la *représentation latente* et non pas sur les sorties du modèle, c'est la raison pour laquelle nous n'utilisons pas l'entropie croisée comme fonction de coût du modèle. Nous proposons d'entraîner un réseau de neurones à générer une représentation spécifique permettant la classification et la détection de classes inconnues : pour cela, des ancres sont fixées dans l'espace de représentation pour répartir les classes, sur le même principe que CAC.

Pour contraindre l'apprentissage de cette représentation, une mesure de distance est utilisée comme fonction de coût. Cette distance est aussi utilisée pour déterminer la classe prédite et détecter les classes inconnues. Dans la continuité des approches de pointes, nous n'utilisons pas la théorie des valeurs extrêmes, car nous n'avons pas de probabilités à recalibrer et, de plus, les prédictions du modèle sont directement basées sur la distance aux ancres.

Un intérêt particulier est porté sur l'apprentissage de représentations sémantiques, suivant l'hypothèse de Miller et al. (2021) selon laquelle cela améliorerait les résultats en OSR, et en accord avec les performances obtenues par Cevikalp et al. (2023) avec leur approche DCHS. Miller et al. (2021) indiquent que fixer les ancres ne permet pas d'obtenir une représentation sémantique, c'est pourquoi nous voulons rendre plus flexible la repré-

sentation en permettant aux instances d'être représentées dans une hypersphère autour de leur ancre, et non pas directement sur celle-ci. L'objectif de cette flexibilité est de permettre à des instances de classes sémantiquement proches d'être représentées à proximité dans l'espace de représentation.

Comparativement aux approches présentées, notre méthode possède une caractéristique qui n'avait pas encore été proposée : en effet, les contraintes que nous appliquons à la représentation latente ont également pour objectif de la rendre plus interprétable, en associant chaque dimension de la représentation à une classe.

Chapitre 3

DIFAIR : DIFferentiAted Image Representations

Résumé

Ce chapitre présente les concepts et hypothèses derrière DIFAIR (DIFferentiAted Images Representations), une méthode d'apprentissage de représentation que nous proposons pour aborder les défis liés à l'OSR, que nous avons précédemment introduits. Dans un premier temps, la méthode est présentée et son application en OSR est discutée. Ensuite, nous détaillons deux particularités de la représentation apprise, à savoir sa sémantique et son interprétabilité. Trois méthodes de visualisation et d'analyse des représentations sont décrites, avant de discuter des avantages de DIFAIR par rapport aux méthodes existantes.

Sommaire

Les modèles d'intelligence artificielle réalisent des prédictions dans l'ensemble des classes qu'ils connaissent, et ce, indépendamment du fait qu'une entrée donnée peut ne pas appartenir à une de ces classes. Cette caractéristique s'avère peu adaptée à l'utilisation de ces modèles dans des environnements réels qui évoluent constamment, et où la présence de classes inconnues est inévitable. Le domaine de l'*Open-Set Recognition* (OSR) aborde cette problématique avec un double objectif : permettre aux modèles de détecter les situations inconnues afin de pouvoir rejeter les prédictions qui seraient erronées, et tout en étant capable de réaliser des prédictions correctes face à des données connues. Ce chapitre se concentre sur l'application de l'OSR aux réseaux de neurones profonds utilisés pour la classification d'images. Comme nous l'avons vu dans le chapitre 2, les travaux existants indiquent qu'une approche d'entraînement discriminative utilisant l'entropie croisée et entraînée uniquement à partir des classes connues, ne permet pas de détecter efficacement les classes inconnues. Une alternative serait d'apprendre à représenter sémantiquement les données dans l'espace de représentation latent du réseau, tout en imposant une séparation suffisante des classes dans ce même espace.

Afin de répondre à la problématique de l'OSR, tout en nous attachant à l'idée de faire construire une représentation sémantique par un modèle, nous proposons une nouvelle approche nommée DIFAIR (*DIFferentiAted Images Representations*). Notre objectif est de *contraindre la représentation latente* apprise par un réseau de neurones convolutif pour la classification d'images, afin que celle-ci permette à la fois :

— La détection des données inconnues et la classification des données connues.

— L'obtention d'une représentation reflétant, au moins dans une certaine mesure, la sémantique des données.

— L'obtention d'une représentation interprétable, pour permettre à l'utilisateur du modèle, à partir de la représentation, de déterminer si l'image d'entrée est connue par le réseau et le cas échéant la classe prédite par le modèle.

Nous souhaitons donc contribuer à l'OSR en proposant une approche apportant une combinaison de sémantique et d'interprétabilité afin de mieux comprendre pour quelles raisons une prédiction est rejetée ou non. Cela permettrait aussi de mieux comprendre le fonctionnement du modèle et aiderait à identifier les limites de l'apprentissage.

3.1 Présentation de DIFAIR

L'hypothèse de la « *Familiarity Hypothesis* » (hypothèse de familiarité), soulevée par Dietterich & Guyer (2022), stipule que, en OSR, les réseaux de neurones n'apprennent pas réellement à détecter la présence d'une classe inconnue dans une image, mais détectent plutôt l'*absence* de classes connues, par exemple, en observant des activations plus faibles des attributs extraits (Chen et al., 2021; Vaze et al., 2022). De ce point de vue, la détection de classes inconnues peut donc être interprétée comme la détection de l'absence de toute classe connue dans l'image. Cela peut créer des difficultés ; par exemple, si une image contient une classe connue et une classe inconnue, cette détection des classes inconnues basée sur la présence des classes connues ne permettra pas de détecter la classe inconnue. La solution proposée par Dietterich & Guyer (2022) dans ce scénario est d'appliquer dans un premier temps des méthodes de détection d'objet, puis d'effectuer une prédiction sur chacun des objets détectés. Suivant cette hypothèse, nous limitons nos travaux à la détection de classes inconnues dans des images où aucune classe connue n'est présente. Pour simplifier la compréhension, nous conservons tout de même la formulation « détection de classe inconnue », bien que cela désigne en réalité la détection de l'*absence de classe connue*.

3.1.1 Description de l'approche

L'optimisation avec entropie croisée et softmax contraint *indirectement* la représentation apprise par un réseau de neurones : la dernière couche du réseau va apprendre à déterminer des frontières de décisions entre les classes dans l'espace de représentation latent. Basée sur l'hypothèse de familiarité, DIFAIR vise à **optimiser directement la représentation apprise par un réseau de neurones pour contraindre les attributs décrivant une classe à s'activer uniquement en présence de cette classe et pas des autres** (des exceptions seront présentées par la suite). Dans une certaine mesure, ce point de vue est partagé par Xu et al. (2023) (section 2.2.2) qui ont pour objectif d'extraire des attributs représentatifs des classes connues grâce au *contrastive learning*, afin que ces attributs ne s'activent pas en présence de classes inconnues.

Pour optimiser un réseau de neurones avec cette contrainte, il faut donc savoir au préalable à quelle classe correspondent les attributs extraits. **C'est pourquoi DIFAIR associe chaque dimension de la représentation à une classe, par opposition aux approches utilisant l'entropie croisée qui n'imposent aucune contrainte à la représentation.**

De plus, nous faisons le choix d'entraîner DIFAIR uniquement sur des données connues (sans utiliser de données *inconnues connues*), afin de déterminer s'il est possible d'extraire des attributs suffisamment représentatifs, à partir des données connues, pour permettre la détection des données inconnues (ce que Vaze et al. (2022) semblent montrer en obtenant des résultats proches de l'état de l'art sans données supplémentaires (section 2.2.2)).

DIFAIR, en optimisant directement la représentation, est une approche applicable à n'importe quel réseau de neurones convolutif. Les seules modifications d'architecture nécessaires sont la suppression des couches situées après l'extracteur d'attributs, et l'ajout d'une couche de convolution supplémentaire, contenant autant de filtres que d'attributs à extraire pour le bon fonctionnement de DIFAIR. Une fonction d'activation est appliquée à la sortie de cette couche de convolution (le choix de cette fonction est discuté dans le chapitre 4). Enfin, du *global average pooling* (Lin et al., 2014) est appliqué pour obtenir la représentation $\boldsymbol{z}$ de l'image en entrée, correspondant à la représentation à apprendre. Chaque dimension de cette représentation correspond à un attribut extrait par le réseau. Une comparaison des différences d'architecture entre un modèle classique utilisant l'entropie croisée et un modèle utilisant DIFAIR est illustrée dans la figure 3.1.

Avec DIFAIR, un attribut extrait est associé à une ou plusieurs caractéristiques d'une image, et a une valeur positive et élevée [1] si les caractéristiques sont bien présentes dans l'image, et une valeur proche de 0 autrement. Plutôt que d'utiliser une couche entièrement connectée combinant les attributs extraits pour effectuer une prédiction, la représentation apprise par DIFAIR contraint les attributs extraits à être représentatifs d'une classe en particulier. Cela permet donc de déterminer la classe d'une image directement en examinant la représentation, ce qui n'est pas possible à partir des représentations apprises par entropie croisée. Nous considérons donc que DIFAIR apprend des représentations que nous nommons *différenciées*.

La représentation d'une image avec DIFAIR est un vecteur de taille $\mathcal{N} \cdot K$, où $\mathcal{N}$ est le nombre de dimensions allouées à chaque classe [2] et K le nombre de classes. Notre hypothèse est qu'en réservant plusieurs dimensions pour représenter une classe, les attributs extraits sur ces dimensions pourraient être représentatifs de *caractéristiques distinctes* de la classe. Ainsi,

1. Un attribut est considéré comme ayant une valeur élevée lorsqu'il est proche de sa valeur cible (section 3.1.2).

2. Dans les travaux que nous présentons, chaque classe est représentée par $\mathcal{N}$ dimensions ; de futurs travaux pourraient étudier l'impact de l'allocation d'un nombre différent de dimensions à chaque classe.

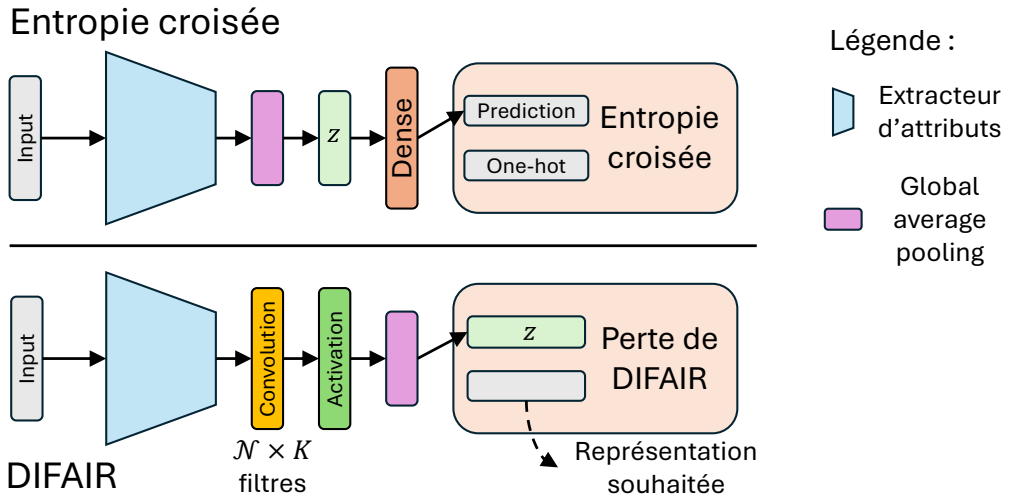


FIGURE 3.1: **Comparaison des architectures d'un réseau de neurones classique et de DIFAIR.** En haut, l'architecture d'un réseau de neurones classique utilisant l'entropie croisée pour optimiser la prédiction réalisée par le réseau. En bas, l'architecture modifiée pour appliquer DIFAIR, qui optimise directement la représentation $\boldsymbol{z}$. Le point commun entre ces deux architectures est l'extracteur d'attributs.

ces attributs pourraient correspondre à une *décomposition sémantique* de la classe à reconnaître. Des détails supplémentaires sur ce point sont présentés dans les sections 3.2 et 3.3.

Pour illustrer notre propos, considérons une représentation $\boldsymbol{z} \in \mathbb{R}^{\mathcal{N} \cdot K}$ d'une image obtenue grâce à DIFAIR, l'ensemble $\mathbb{I}_c$ contient les indices des dimensions de $\boldsymbol{z}$ associées à la classe $c \in \{1, 2, \ldots, K\}$:

$$\mathbb{I}_c = \{(c-1) \cdot \mathcal{N} + 1, (c-1) \cdot \mathcal{N} + 2, \ldots, (c-1) \cdot \mathcal{N} + \mathcal{N}\}.$$

La figure 3.2 illustre de façon schématique un exemple de représentations types que nous voudrions obtenir avec DIFAIR avec un modèle entraîné à reconnaître des chats et des zèbres. La classe inconnue « tigre » partage des caractéristiques avec les deux classes connues et pourrait donc activer des attributs de chacune des classes.

3.1.2 Apprentissage

Ancres. Nous proposons d'utiliser des *ancres* pour contraindre la représentation apprise par le réseau, à l'instar de CAC (Miller et al., 2021). Dans le cas de DIFAIR, les ancres sont placées dans l'espace de représentation la-

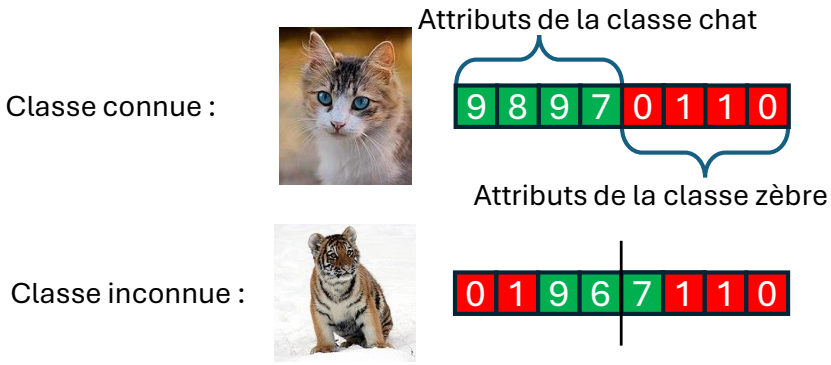


Figure 3.2: **Schéma des attributs extraits par DIFAIR.** Considérons qu'un modèle a été entraîné à reconnaître des chats et des zèbres en utilisant DIFAIR. Les attributs fortement activés sont en vert et indiquent que des caractéristiques qui leur sont associées ont été détectées dans l'image. Les attributs faiblement activés sont en rouge et indiquent que les caractéristiques qui leur sont associées n'ont pas été détectées dans l'image.
Source des photographies : Wikimedia Commons.

tent plutôt que dans l'espace des logits, ce qui permet à DIFAIR de pouvoir représenter les classes sur plus de dimensions. Les ancres sont des vecteurs de taille $\mathcal{N} \cdot K$ qui servent de référence pour chaque classe, signifiant que les instances d'une classe doivent être représentées à proximité de l'ancre de leur classe. Une ancre $\mathcal{A}^{(c)}$ représentant la classe c a pour coordonnées :

$$\mathcal{A}_i^{(c)} = \begin{cases} \alpha & \text{si } i \in \mathbb{I}_c, \\ 0 & \text{sinon,} \end{cases}$$

où α est un hyperparamètre permettant de contrôler la distance entre les ancres, et $\mathbb{I}_c$ est l'ensemble des indices des dimensions associées à la classe c. La valeur α est la même pour toutes les ancres : il s'agit de la valeur « cible » de chaque attribut. Lorsqu'un attribut est activé avec une valeur proche de α, on peut considérer qu'il a une valeur *élevée*, sous entendant que les caractéristiques associées à l'attribut ont été détectées dans l'image d'entrée. Autoriser des valeurs α différentes pour chaque classe aurait pour conséquence de rapprocher certaines ancres et de les éloigner d'autres, ce qui créerait des problèmes concernant la sémantique que nous voulons permettre dans la représentation (section 3.2).

Pour déterminer la distance entre la représentation d'une instance et les ancres, nous proposons d'utiliser la distance euclidienne, définie entre deux vecteurs $\boldsymbol{x}$ et $\boldsymbol{y}$ de taille n par :

$$\text{Eucl}(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}.$$

Ainsi configurées, et en utilisant la distance euclidienne, toutes les ancres sont équidistantes, comme c'est le cas pour CAC.

Selon Miller et al. (2021), des ancres équidistantes ne permettent pas d'obtenir une représentation exprimant une sémantique, c'est pourquoi nous proposons d'autoriser la représentation des instances dans une hypersphère de rayon r autour des ancres. Concrètement, cela revient à ne pas contraindre le placement des instances sur les ancres, ce que fait CAC. Ainsi, la distance entre la représentation d'une instance et l'ancre de sa classe doit être inférieure à r pour que l'instance soit considérée comme appartenant à la classe durant l'apprentissage. La flexibilité dans la représentation apportée par l'utilisation d'hypersphères autour des ancres permet d'obtenir une représentation plus sémantique, comme le détaille la section 3.2.

Fonction de perte. La fonction de perte proposée pour optimiser la représentation, conformément aux points précédemment soulevés, est définie comme suit :

$$\mathcal{L}(\boldsymbol{X}, y) = \max(\text{Eucl}(\boldsymbol{z}, \mathcal{A}^{(y)}) - r, 0),$$

où $\text{Eucl}(\boldsymbol{z}, \mathcal{A}^{(y)})$ est la distance entre la représentation $\boldsymbol{z}$ de l'image $\boldsymbol{X}$ et l'ancre de la classe y. Cette fonction retourne une valeur de 0 lorsque l'instance est dans l'hypersphère de rayon r autour de l'ancre de sa classe, et renvoie la distance par rapport à l'hypersphère sinon. La figure 3.3 illustre le comportement de cette fonction de perte.

Rappelons que l'optimisation avec la fonction d'entropie croisée influe indirectement sur la représentation apprise : les attributs extraits ne sont pas contraint à valoir 0 lorsqu'une classe est absente de l'image en entrée et il est seulement nécessaire que les attributs de la vraie classe soient plus activés que les autres attributs afin que le logit de la vraie classe soit le plus élevé. En utilisant la distance euclidienne comme fonction de perte dans DIFAIR, nous évitons cette caractéristique de l'apprentissage avec entropie croisée. S'il n'y avait pas d'hypersphère autour des ancres, les attributs des autres classes devraient valoir 0 pour que la valeur de la perte soit basse.

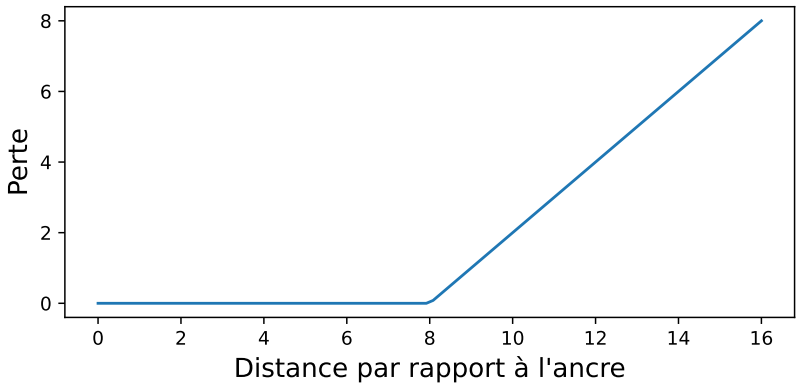


Figure 3.3: **Comportement de la fonction de perte de DIFAIR.** Pour la représentation d'une instance quelconque, sa perte correspond à sa distance par rapport à son hypersphère (pas par rapport à son ancre). Il n'y a pas de pénalisation si l'instance est dans l'hypersphère.

Cependant, avec l'hypersphère, les attributs des autres classes peuvent s'activer sans pour autant pénaliser l'apprentissage, cette activation se réalise donc dans une certaine limite, ce qui diffère de l'optimisation avec l'entropie croisée.

La fonction de perte proposée répond ainsi aux objectifs fixés :

— Pour qu'une instance soit proche de son ancre, il faut que seules les dimensions associées à la classe soient fortement activées. S'il existe une relation sémantique entre les classes, les activations d'autres dimensions sont possibles grâce à l'hypersphère (section 3.2).

— Il n'y a pas de pénalisation lorsque les instances sont représentées à l'intérieur de l'hypersphère, ce qui permet d'obtenir une représentation plus sémantique en laissant le réseau représenter librement les instances de la classe dans l'hypersphère (section 3.2).

— Avec des hypersphères dont le rayon est inférieur à la moitié de la distance entre deux ancres, les hypersphères ne se chevauchent pas, ce qui permet d'obtenir une représentation où les instances des classes sont correctement séparées les unes des autres, et donc de pouvoir discriminer les classes connues.

Comme nous le verrons par la suite (chapitre 4), DIFAIR n'exhibe pourtant pas le comportement attendu, et des améliorations seront proposées pour tenter de répondre aux problèmes rencontrés.

3.1.3 Utilisation de DIFAIR pour l'OSR

Postulons que DIFAIR permet de représenter les données autour d'ancres bien séparées dans l'espace de représentation, et d'extraire des attributs distincts représentant les classes, qui s'activent lorsque certaines caractéristiques sont effectivement présentes dans l'image d'entrée. Dans ce cas, la représentation apprise par DIFAIR peut être utilisée pour détecter les classes inconnues. En effet, les classes inconnues peuvent être détectées dans deux des trois scénarios suivants :

— Si la classe inconnue ne partage aucune caractéristique avec les classes connues, aucun attribut dans la représentation ne s'activera, donc aucune classe connue n'est présente dans l'image.

— Si la classe inconnue partage des caractéristiques avec les classes connues, des attributs pourront s'activer (potentiellement plus faiblement), mais en théorie tous les attributs d'une classe ne devraient pas s'activer. Il est possible que des attributs de classes différentes s'activent en cas de similarité avec plusieurs classes (comme le montre la figure 3.2). Dans ce cas, le réseau ne reconnaît pas une classe connue avec certitude, ce qui suggère qu'il ne s'agit probablement pas d'une classe connue.

— L'exception concerne le cas où la classe inconnue active tous les attributs d'une classe connue sans en activer d'autres. Soit il s'agit d'une erreur d'étiquetage (et la donnée appartient en réalité à une classe connue), soit les attributs extraits ne sont pas suffisamment spécifiques à la classe connue et sont activés indifféremment par des données de classe connue ou inconnue.

Au vu de ces considérations, les principes de DIFAIR sont adaptés pour l'OSR. Nous proposons d'utiliser la distance euclidienne dans l'espace de représentation pour automatiser les tâches d'OSR : dans la représentation d'une nouvelle instance, plus les attributs d'une seule classe sont activés, plus l'instance est représentée à proximité de l'ancre de cette classe. Dans la représentation d'une classe inconnue, si les dimensions sont moins activées (caractéristiques peu reconnues) ou bien que des dimensions sont activées pour *plusieurs classes* (partage de caractéristiques avec plusieurs classes connues), l'image inconnue serait représentée éloignée des ancres des classes connues. C'est pourquoi afin de déterminer la classe prédite par le réseau pour une instance $\boldsymbol{X}$, nous mesurons la distance entre la représentation $\boldsymbol{z}$ de cette instance et toutes les ancres : la classe prédite $\hat{y}$ est la classe de l'ancre la plus proche. En fixant un seuil sur la distance à l'ancre la plus proche,

il est possible de rejeter une prédiction pour signifier que l'image contient possiblement une classe inconnue, car une représentation trop éloignée de toutes les ancres ne correspond probablement pas à une classe connue.

Définissons $\boldsymbol{d}$ comme le vecteur des distances entre la représentation $\boldsymbol{z}$ et toutes les ancres, où $\boldsymbol{d} = (\text{Eucl}(\boldsymbol{z}, \mathcal{A}^{(1)}), \text{Eucl}(\boldsymbol{z}, \mathcal{A}^{(2)}), \ldots, \text{Eucl}(\boldsymbol{z}, \mathcal{A}^{(K)})$. En utilisant DIFAIR, le score d'OSR choisi est $S(y \in \mathbb{C}|\boldsymbol{X}) = \min \boldsymbol{d}$. Ainsi, la prédiction réalisée par le modèle est définie comme suit :

$$\hat{y} = \begin{cases} -1 & \text{si } S(y \in \mathbb{C}|\boldsymbol{X}) > \epsilon, \\ \arg\min \boldsymbol{d} & \text{sinon,} \end{cases}$$

où ϵ est le seuil de rejet, et en considérant qu'une donnée identifiée inconnue est prédite comme la classe -1.

3.2 Sémantique des représentations

Dans un espace de représentation, nous considérons que la similarité sémantique entre deux images est exprimée par la proximité entre les représentations des deux instances. Par exemple, dans un espace de représentation sémantique, la distance entre une instance de chat et une instance de chien sera plus faible que la distance entre une instance de chat et une instance de véhicule.

Dans l'approche CAC, les ancres sont équidistantes et les instances sont optimisées pour être regroupées au plus proche des ancres : si le minimum global pour ce problème d'optimisation est atteint, toutes les instances d'une classe seront représentées exactement sur leur ancre dans l'espace latent. Par conséquent, cette représentation ne permet pas d'exprimer de degré de proximité d'une instance à une ou plusieurs classes, on ne peut donc pas considérer qu'il existe une sémantique dans la représentation.

En revanche, si l'optimisation permet de représenter les instances dans des hypersphères autour des ancres, alors, au minimum global de cette optimisation, toutes les instances seront représentées à l'intérieur de l'hypersphère, sans nécessairement être regroupées uniquement sur l'ancre. Dans cette situation, il est possible (mais pas certain, car il n'y a pas de contrainte explicite) qu'un point représentant une instance de chat soit situé dans son hypersphère du côté de l'ancre de la classe chien, exprimant donc une proximité avec la classe chien pour cette instance. Nous faisons l'hypothèse qu'il s'agira d'un comportement émergent de l'apprentissage, lié à l'usage des hy-

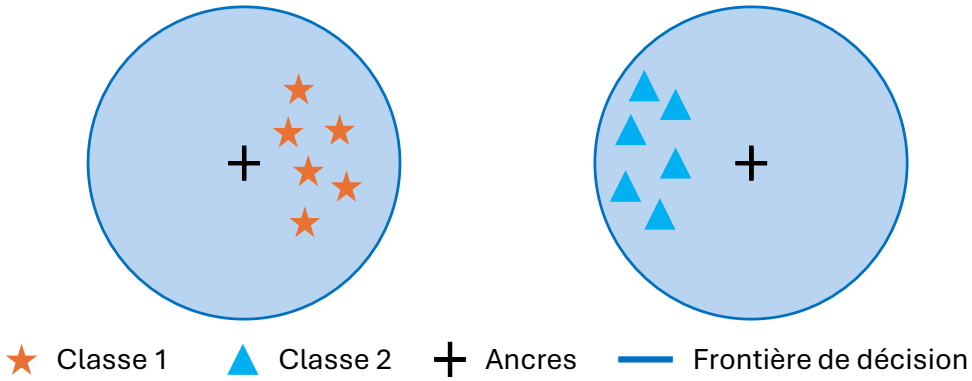


FIGURE 3.4: **Schéma de la sémantique des représentations permise par DIFAIR.** Si les classes A et B sont des classes sémantiquement proches, les instances de la classe A pourront être représentées dans leur hypersphère et proches de l'hypersphère de la classe B. Inversement pour les instances de la classe B.

persphères, la sémantique étant aussi un comportement émergent dans les modèles entraînés avec entropie croisée. La figure 3.4 illustre cette situation pour deux classes.

Pour que cette proximité soit exprimée, et donc que la représentation possède une sémantique, les attributs de plusieurs classes différentes doivent pouvoir s'activer *lorsque ces classes partagent des caractéristiques*. L'objectif initial étant d'activer les attributs d'une classe uniquement en présence de celle-ci, l'optimisation doit être flexible par rapport à l'activation des attributs des autres classes. L'utilisation de l'hypersphère et de la distance euclidienne permettent cela.

En effet, prenons l'exemple de $K = 2$ classes sémantiquement proches, les chats et les chiens. Considérons que DIFAIR est paramétré par $\mathcal{N} = 3$ et $\alpha = 10$. Les ancres $\mathcal{A}$ de ces classes sont situées aux coordonnées :

$$\mathcal{A}^{\text{chat}} = (10, 10, 10, 0, 0, 0) \quad \text{et} \quad \mathcal{A}^{\text{chien}} = (0, 0, 0, 10, 10, 10).$$

La distance euclidienne entre ces deux ancres est calculée comme suit : $\sqrt{2 * \mathcal{N} * 10^2} \approx 24.5$. Soit $\boldsymbol{z}$ la représentation d'une instance quelconque. La distance euclidienne entre $\boldsymbol{z}$ et $\mathcal{A}^{\text{chat}}$ peut s'écrire :

$$\text{Eucl}(\boldsymbol{z}, \mathcal{A}^{\text{chat}}) = \sqrt{\sum_{i=1}^{\mathcal{N}} (10 - z_i)^2 + \sum_{i=\mathcal{N}+1}^{\mathcal{N} \cdot K} (0 - z_i)^2}.$$

Considérons maintenant que $\boldsymbol{z}$ soit la représentation d'une image de chat, notons que les dimensions associées à la classe chat sont toutes activées avec la valeur de 10 (indiquant que la classe est bien reconnue) et les autres dimensions sont à 0 pour simplifier les calculs. La représentation $\boldsymbol{z}$ est alors :

$$z_i = \begin{cases} 10 & \text{si } i \in \{1, \ldots, \mathcal{N}\}, \\ 0 & \text{si } i \in \{\mathcal{N}, \ldots, \mathcal{N} \cdot 2\}. \end{cases}$$

Puisque $\boldsymbol{z}$ est situé exactement sur l'ancre,

$$\text{Eucl}(\boldsymbol{z}, \mathcal{A}^{\text{chat}}) = 0 \quad \text{et} \quad \text{Eucl}(\boldsymbol{z}, \mathcal{A}^{\text{chien}}) \approx 24.5.$$

Soit, $\mathbb{I}_{\text{chien}} = \{\mathcal{N} + 1, \ldots, 2 \cdot \mathcal{N}\}$ les indices des dimensions de la représentation associées à la classe chien. S'il existe $j \in \mathbb{I}_{\text{chien}}$ tel que $z_j \neq 0$, alors la distance entre $\boldsymbol{z}$ et l'ancre de la classe chat s'écrit :

$$\text{Eucl}(\boldsymbol{z}, \mathcal{A}^{\text{chat}}) = \sqrt{\sum_j z_j^2}.$$

Tant que $\text{Eucl}(\boldsymbol{z}, \mathcal{A}^{\text{chat}}) < r$, avec r le rayon de l'hypersphère, les z_j peuvent être non nuls sans pour autant être pénalisants dans la fonction de perte, car l'instance $\boldsymbol{z}$ reste représentée dans l'hypersphère de la classe chat.

Par exemple, si $z_{\mathcal{N}+1} = 8$, alors $\text{Eucl}(\boldsymbol{z}, \mathcal{A}^{\text{chat}}) = 8$ et $\text{Eucl}(\boldsymbol{z}, \mathcal{A}^{\text{chien}}) \approx 22.4$. Si $r = 8$, alors $\boldsymbol{z}$ reste dans l'hypersphère associée à la classe chat et respecte les contraintes appliquées à la représentation, tout en contenant une information sur la sémantique entre la classe chat et chien, avec une dimension de la classe chien activée [3].

Grâce à l'allocation d'hypersphères autour des ancres, dans lesquelles les instances peuvent être représentées, une instance peut rester représentée dans l'hypersphère de la classe correcte même avec des valeurs activées pour d'autres classes. Pour ces raisons, nous supposons que la représentation apprise par DIFAIR peut exprimer une sémantique améliorant l'interprétabilité et la richesse des représentations.

3.3 Recherche d'interprétabilité

Plusieurs types de représentations sont possibles pour représenter un ensemble de classes, dont trois représentations particulières sont illustrées dans la figure 3.5. Selon la représentation utilisée, l'interprétation de celle-ci

3. Une valeur élevée d'un attribut de la classe chien éloigne beaucoup l'instance de l'ancre de la classe chat, sans pour autant la rapprocher significativement de la classe chien : la distance euclidienne peut donc s'avérer problématique.

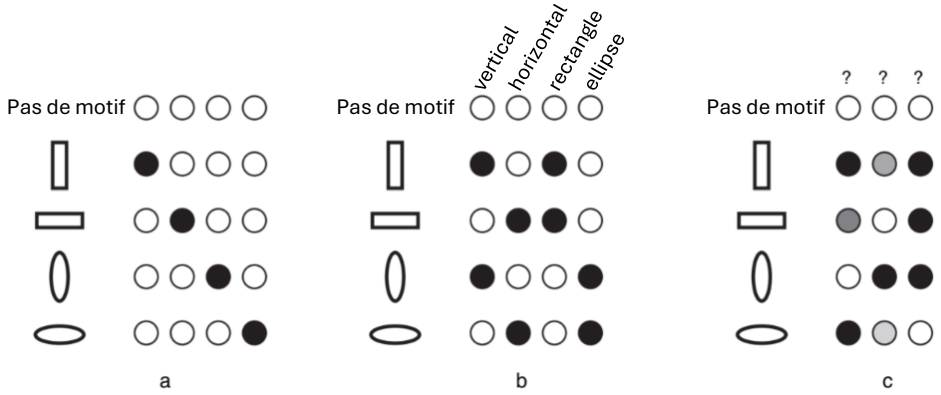


FIGURE 3.5: **Différentes possibilités de représentations.** Chaque ligne correspond à la représentation de la classe indiquée à gauche. Les dimensions activées sont indiquées par un fond coloré. (a) Représentation non-distribuée. (b) Représentation distribuée et basée sur les attributs. (c) Représentation distribuée.
Source : figure issue de Tosh & Ruxton (2010)

peut être plus ou moins aisée. Considérons l'exemple proposé par Tosh & Ruxton (2010), où quatre classes doivent être représentées : un rectangle vertical, un rectangle horizontal, une ellipse verticale et une ellipse horizontale.

La figure 3.5a montre une représentation *non-distribuée* des attributs, où chaque dimension est représentative d'une classe dans sa globalité. Une dimension par classe est donc requise dans cette représentation. La dimension représentant la classe est activée tandis que les autres ont une valeur proche de 0.

La figure 3.5b présente une représentation *distribuée et basée sur les attributs*, où chaque dimension est associée à une caractéristique des classes. Les classes sont donc représentées par une combinaison d'attributs *identifiés* (associés à une caractéristique), ici : vertical, horizontal, rectangle, ellipse. Une telle représentation est donc facilement interprétable, car il suffit de déterminer à quelle caractéristique correspond l'attribut activé.

Enfin, la figure 3.5c montre une autre forme de représentation *distribuée*, où chaque classe est représentée par une combinaison d'attributs *non identifiés*. Chaque attribut peut être plus ou moins activé par plusieurs classes. Dans cette modalidé de représentation, il n'est pas nécessaire de définir autant de dimensions que de classes. L'interprétation d'une telle représen-

tation semble d'autant plus complexe que des informations sémantiques de différentes classes sont combinées chacune dans des dimensions de la représentation.

Les réseaux de neurones entraînés avec la fonction de perte d'entropie croisée sont notamment réputés pour apprendre des représentations distribuées (Hinton, 1984), c'est-à-dire que les attributs extraits peuvent être représentatifs de plusieurs caractéristiques de classes différentes. Par exemple, dans un réseau de neurones entraîné avec entropie croisée, un attribut pourrait être actif pendant le traitement d'une donnée appartenant à la classe chien, et ce même attribut (éventuellement avec un niveau d'activation différent) pourrait également être actif pendant le traitement d'une donnée appartenant à la classe chat. De plus, les représentations dans ces modèles sont souvent distribuées sur des centaines de dimensions, ce qui complexifie davantage la tâche d'interprétation.

L'objectif principal de DIFAIR est qu'une dimension de la représentation latente $\boldsymbol{z}$ du réseau de neurones ait une *valeur élevée* uniquement en présence de caractéristiques connues dans une image ; si ces caractéristiques sont absentes, la valeur de la dimension devrait être proche de 0. De plus, les dimensions de $\boldsymbol{z}$ sont chacune associées à une classe. Chaque classe est donc représentée par $\mathcal{N}$ dimensions qui doivent s'activer en présence de cette classe ; chacune de ces $\mathcal{N}$ dimensions devrait représenter une caractéristique *distincte* de la classe. Puisque l'objectif est d'activer chaque attribut pour une classe donnée, notre méthode devrait extraire des attributs indépendants pour chaque classe, excepté lorsque des caractéristiques sont partagées par des classes. Nous notons que, bien que notre préférence soit d'extraire une seule caractéristique reconnue par dimension, cela n'est pas un objectif explicitement optimisé. Par conséquent, le réseau peut techniquement apprendre à détecter un *groupe de caractéristiques* dans une image et les exprimer sur une unique dimension. Ainsi, une classe serait décrite par plusieurs groupes de caractéristiques, chacun associé à une dimension.

Les contraintes appliquées à la représentation apprise par un réseau de neurones en utilisant DIFAIR rendent celle-ci plus interprétable, en proposant un niveau de détail sémantique proche de la représentation basée sur les attributs. En effet, utiliser DIFAIR avec $\mathcal{N} = 1$ revient à chercher à obtenir une représentation non-distribuée, ce qui permet uniquement d'identifier la classe, sans donner d'information sur une quelconque proximité sémantique entre les classes. Or lorsque $\mathcal{N} \geq 2$, les attributs extraits pour chaque classe peuvent être de plus bas niveau, c'est-à-dire qu'ils ne vont pas nécessairement représenter la classe dans son ensemble, mais des composants

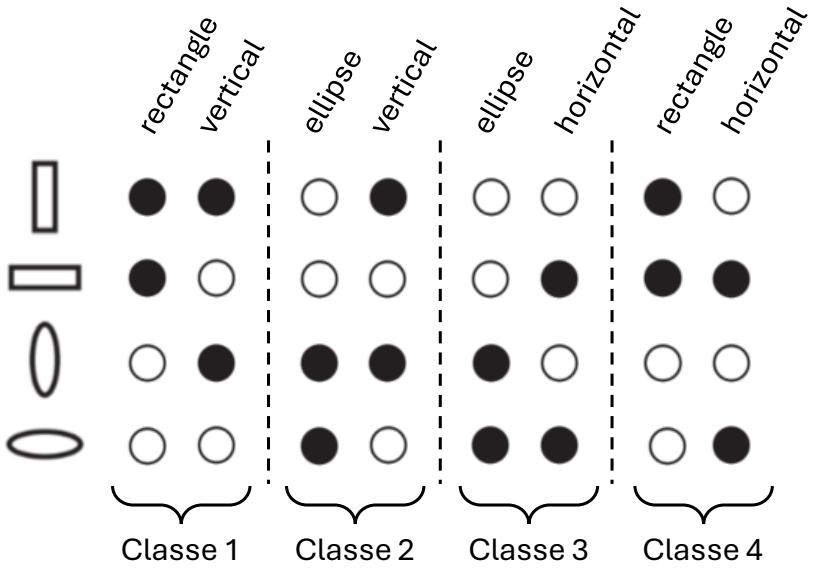


FIGURE 3.6: **Représentation attendue avec DIFAIR.** Chaque ligne correspond à la représentation de la classe indiquée à la gauche. Les dimensions activées sont indiquées par un fond coloré. Les interprétations des colonnes sont ajoutées pour la compréhension de l'exemple, mais en pratique les caractéristiques extraites sur chaque dimension ne sont pas directement identifiées. Avec DIFAIR, l'objectif est que les attributs communs à plusieurs classes soient dupliqués sur différentes dimensions, car chaque classe est identifiée uniquement par ses dimensions. Cette duplication apporte l'information sur la relation sémantique entre les classes.

sémantiques de cette classe. De plus, la représentation apprise par DIFAIR n'est pas distribuée comme peut l'être celle apprise par un réseau de neurones entraîné avec entropie croisée : avec DIFAIR, les attributs extraits doivent être distincts au sein d'une même classe et indépendants entre les classes. Ainsi, DIFAIR permet d'obtenir des représentations *différenciées*, où les classes sont représentées par des attributs prédéfinis, actifs en présence de caractéristiques spécifiques aux classes associées et où les attributs partagés par plusieurs classes sont dupliqués. La figure 3.6 illustre le type de représentation recherché avec DIFAIR, en reprenant l'exemple précédent où quatre classes sont représentées [4].

4. Ici encore, on peut noter que l'utilisation de la distance euclidienne pourrait être inappropriée sur une représentation non-distribuée telle qu'obtenue avec DIFAIR, là où cette distance est plus pertinente lorsque les représentations sont distribuées.

Par conséquent, l'utilisation d'ancres avec $\mathcal{N} \geq 2$ dimensions associées à chaque classe facilite l'apprentissage de représentations plus interprétables. DIFAIR garantit à minima que, pour une image donnée, tous les attributs détectés (toutes les dimensions activées) peuvent être associés à une classe.

Pour faciliter l'interprétation de la représentation, les attributs qui expriment des caractéristiques communes à plusieurs classes sont dupliqués, comme illustré dans la figure 3.6. Bien que cette duplication ne soit pas explicitement contrainte, elle est autorisée par l'utilisation des hypersphères. La duplication des attributs communs est préférée à une combinaison de ceux-ci. Cette duplication permet d'indiquer que les classes partagent des caractéristiques et partagent ainsi possiblement une certaine similarité sémantique. Notons que si DIFAIR construisait une représentation « parfaite », au sens où seuls les attributs de la classe et les attributs communs aux autres classes sont activés (le reste des valeurs est à 0), la représentation pourrait être comprimée pour obtenir une représentation basée sur les attributs. Cette piste n'a cependant pas été explorée dans le cadre de cette thèse.

3.4 Analyse et discussions des représentations

Afin de mieux interpréter les résultats obtenus par DIFAIR, nous décrivons différentes méthodes de visualisations et d'analyse des représentations apprises. Dans un premier temps, la méthode de réduction de dimensions t-SNE est présentée pour visualiser les représentations extraites par DIFAIR dans un espace en deux dimensions. Ensuite, nous introduisons les diagrammes de Hinton, permettant de visualiser les activations de la représentation, et nous décrivons leur usage pour interpréter la représentation. Enfin, nous présentons l'utilisation d'un dendrogramme, un type de diagramme permettant de visualiser les relations hiérarchiques, et dans notre cas sémantiques, entre les classes dans l'espace de représentation.

3.4.1 t-SNE

La visualisation de points dans un espace est limitée à deux ou trois dimensions. Or, les représentations apprises par les réseaux de neurones profonds pour la classification d'images ont généralement beaucoup plus de dimensions, afin d'apprendre une représentation riche et précise des données. L'espace de représentation appris par DIFAIR a $\mathcal{N} \cdot K$ dimensions,

avec $\mathcal{N}$ le nombre de dimensions par classe et K le nombre de classes. Ceci implique, sauf si l'on étudie un problème avec peu de classes et en allouant peu de dimensions par classe, que l'espace de représentation de DIFAIR est aussi de haute dimensionnalité.

Pour visualiser ces représentations, il est donc essentiel de réduire la dimension de l'espace de représentation. Le défi consiste à transformer un espace en n dimensions en un espace de deux ou trois dimensions, tout en préservant certaines propriétés des données (telles que la proximité entre les points dans l'espace initial). Nous utilisons la technique *t-distributed Stochastic Neighbor Embedding* (t-SNE) (Van der Maaten & Hinton, 2008), une méthode non-linéaire de réduction de dimensions qui permet de capturer les relations complexes entre les données. Cette méthode a aussi été utilisée en OSR par plusieurs auteurs (Yoshihashi et al., 2019; Zhang et al., 2020; Cevikalp et al., 2023) pour visualiser les représentations apprises par leurs modèles.

Visualiser les représentations apprises a plusieurs avantages. Cela permet dans un premier temps de vérifier que les données d'une même classes sont bien regroupées dans l'espace de représentation. Ensuite, si des groupes d'instances appartenant à des classes sémantiquement proches sont représentés à proximité avec t-SNE, cela signifie que la représentation à visualiser exprime une proximité sémantique entre les classes. Dans le cas de l'OSR, la visualisation permet également de localiser dans l'espace les instances inconnues par rapport aux classes connues.

La méthode t-SNE, proposée par Van der Maaten & Hinton (2008), s'appuie sur la méthode SNE (Hinton & Roweis, 2002). L'objectif est de réduire la dimension de l'espace de représentation tout en préservant les proximités entre les point : deux points proches (respectivement éloignés) dans l'espace de représentation initial doivent aussi être proches (respectivement éloignés) dans l'espace de représentation réduit. Pour parvenir à ce résultat, la première étape consiste à estimer une distribution de probabilités dans l'espace en haute dimension sur chaque paire de point. Une haute probabilité est assignée à une paire de point proches, tandis qu'une faible probabilité est assignée à une paire de points éloignés. La proximité est mesurée en utilisant la distance euclidienne. Puis, une distribution similaire est créée à partir des points dans l'espace de représentation réduit : ces points peuvent être initialisés aléatoirement ou en utilisant une autre méthode de réduction de dimensions comme la *Principal Component Analysis* (PCA) (Pearson, 1901). L'objectif est de minimiser la divergence de Kullback-Leibler (KL-divergence) (Kullback & Leibler, 1951) entre ces deux

distributions, la KL-divergence permettant de mesurer la différence entre deux distributions de probabilités. La minimisation est réalisée en utilisant une descente de gradient stochastique. Lorsque l'algorithme converge, les points dans l'espace de représentation réduit sont positionnés selon la même distribution de proximité que dans l'espace de représentation initial, permettant de conserver les proximités entre les points.

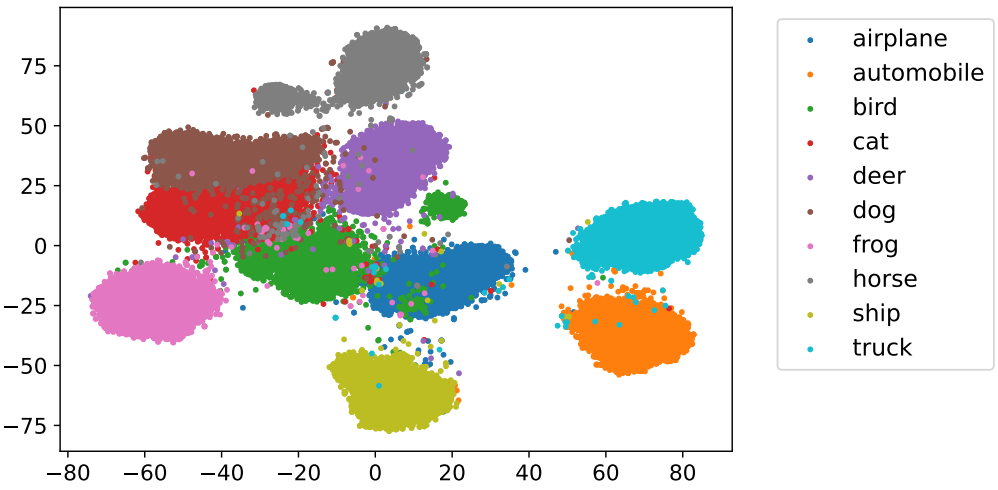


Figure 3.7: **Exemple de visualisation t-SNE d'un espace de représentation appris par un modèle entraîné avec entropie croisée.** Ce modèle a été entraîné sur le jeu de données CIFAR-10. L'espace de représentation latent possède 128 dimensions. Uniquement les données correctement classées ont été utilisées pour cette visualisation.

La figure 3.7 montre un exemple de visualisation de représentations apprises par un modèle entraîné avec entropie croisée sur le jeu de données CIFAR-10 (contenant quatre classes de véhicules et six d'animaux). Cette visualisation révèle que les points d'une même classe sont regroupés, indiquant que le modèle a appris à bien séparer les classes. De plus, les classes sémantiquement similaires sont proches dans l'espace de représentation, les classes sémantiquement différentes peuvent notamment être séparés linéairement dans cet espace réduit et la sémantique des données est préservée dans cet espace. Des résultats avec des représentations apprises par DIFAIR sont présentés dans le chapitre 5.

3.4.2 Diagrammes de Hinton et interprétation

Les diagrammes de Hinton (Hinton & Shallice, 1991; Bremner et al., 1994) sont une méthode de visualisation des activations des neurones dans un réseau de neurones. L'idée est de représenter graphiquement les valeurs des activations, plutôt qu'observer une matrice contenant ces valeurs. Pour cela, chaque activation est représentée par un carré de couleur dont la nuance exprime la valeur de l'activation. En disposant ces carrés côte à côte, le diagramme obtenu permet de visualiser efficacement les neurones qui s'activent ensemble, ainsi que les degrés d'activations de chacun des neurones.

Grâce aux diagrammes de Hinton, pour une image donnée, nous pouvons donc visualiser la représentation extraite par les modèles que nous entraînons avec DIFAIR, et examiner si celle-ci exhibe les activations attendues.

La figure 3.8 montre la visualisation de la représentation d'une image apprise par un modèle entraîné avec entropie croisée. Il est évident qu'à partir de la représentation d'une seule image, une quelconque interprétation des attributs est impossible. L'analyse de plusieurs représentations, ou le calcul d'une représentation moyenne pour plusieurs instances, peut mettre en évidence l'activation régulière de certains attributs pour des classes particulières. Plus l'espace de représentation est de haute dimensionnalité, plus la recherche d'association entre attributs et classes devient complexe.

Par contraste, la représentation obtenue par un modèle entraîné avec DIFAIR, comme illustré dans la figure 3.9a, est immédiatement interprétable puisque chaque attribut est explicitement associé à une classe. Chaque ligne correspond aux attributs associés à une classe. Dans le cas où le nombre de classes serait trop grand, un tri peut être effectué afin de montrer uniquement les classes qui sont le plus activées, comme l'illustre la figure 3.9b. La représentation est donc facilement interprétable et n'est pas limitée par le nombre de classes, car les activations de chaque classe sont bien séparées et peuvent être interprétées tour à tour.

3.4.3 Dendrogrammes

Un dendrogramme est un type de diagramme permettant d'illustrer des relations de proximité ou de similarité entre des objets. Il s'agit d'un arbre dont les feuilles correspondent aux objets, tandis que les nœuds symbolisent un regroupement des objets les plus similaires. Une particularité de cet arbre est que les nœuds sont disposés de manière à refléter la distance entre

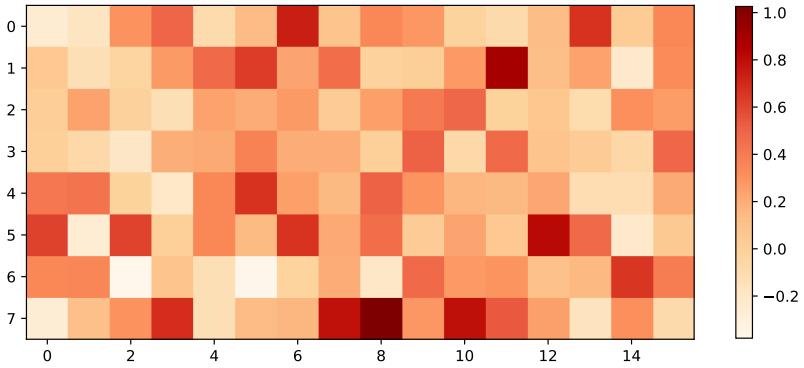
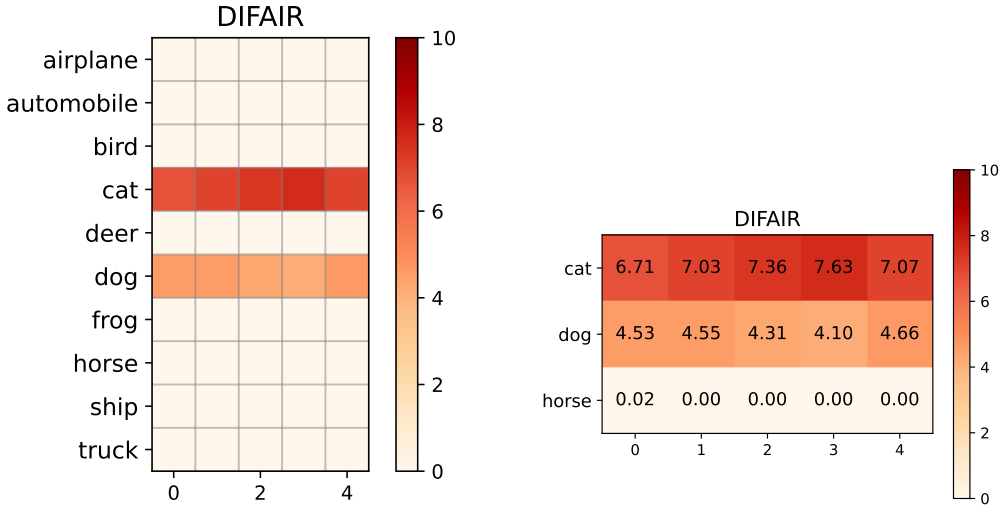


Figure 3.8: **Exemple de visualisation de représentations apprises par un modèle entraîné avec entropie croisée.** Ce modèle a été entraîné sur le jeu de données CIFAR-10. L'espace de représentation latent possède 128 dimensions, redimensionné en une matrice de 16×8 pour la visualisation. Les axes permettent de localiser les attributs : par exemple, l'attribut ligne 1, colonne 11 est fortement activé (par rapport aux autres).



(a) Représentation entière.

(b) Représentation réduite.

Figure 3.9: **Exemple de visualisation de représentations apprises par un modèle entraîné avec DIFAIR.** Ce modèle a été entraîné sur le jeu de données CIFAR-10. Les paramètres utilisés pour DIFAIR sont $\mathcal{N} = 5$, $\alpha = 10$ et $r = 18.97$. (a) La représentation complète montre les attributs associés à chaque classe. (b) La représentation réduite montre uniquement les attributs des 3 classes les plus activées. Dans cette situation, les valeurs peuvent aussi être indiquées s'il y a peu d'attributs à visualiser.

les objets et groupes d'objets. Un dendrogramme est souvent utilisé pour visualiser le résultat d'une classification hiérarchique, laquelle consiste à créer des groupes d'objets en fonction de leur similarité deux à deux.

Dans notre étude, nous utilisons un dendrogramme comme complément à t-SNE pour visualiser les relations sémantiques entre les classes dans l'espace de représentation. Pour cela, nous commençons par calculer la représentation moyenne de chaque classe (le centre des représentations de chaque classe) dans l'espace de représentation étudié et à partir des données correctement classées, de façon à d'éviter d'introduire un bruit lié à des instances qui ne seraient pas représentées au bon endroit. À partir de ces centres, une matrice de distances est calculée, en utilisant la distance euclidienne [5]. Cette matrice permet d'effectuer une classification hiérarchique des centres, qui est ensuite visualisée sous forme de dendrogramme, comme illustré dans la figure 3.10. Ce dendrogramme, réalisé à partir de la représentation d'un modèle entraîné avec entropie croisée sur le jeu de données CIFAR-10 montre que le modèle a appris à représenter à proximité les classes *automobile* et *truck*, *cat* et *dog*, ainsi que *airplane* et *bird*. Les autres relations exposées semblent moins cohérente avec la sémantique que nous pourrions imaginer entre les classes, par exemple, *deer* est plus proche de *ship* et *airplane* que de *horse* et des autres animaux.

Cette visualisation permet de mettre en évidence des relations de proximité entre les classes, que nous interprétons comme l'expression de relations sémantiques. Cependant, chaque classe étant représentée par un seul point, le dendrogramme résultant n'est pas parfaitement indicateur de la sémantique qui peut exister dans la représentation. L'utilisation conjointe de différentes méthodes de visualisation permet de mieux comprendre les représentations apprises.

3.5 Récapitulatif et avantages

Pour récapituler, l'approche DIFAIR que nous proposons optimise directement la représentation d'un réseau de neurones et a pour objectif l'extraction d'attributs distincts pour chaque classe. Chaque attribut extrait est associé à une classe et s'active uniquement en présence de caractéristiques spécifiques dans l'image d'entrée. Les classes inconnues n'étant pas censées présenter toutes les caractéristiques des classes connues, peu d'attributs (ou des attributs de classes différentes) devraient s'activer, ce qui

5. Cevikalp et al. (2023) utilisent un filtre gaussien pour mesurer la similarité entre les centres appris, mais nous utilisons simplement la distance euclidienne pour le moment.

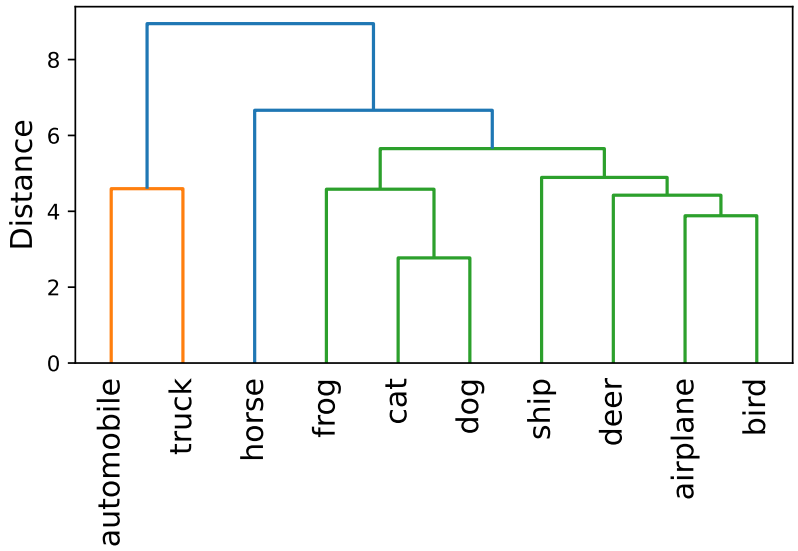


Figure 3.10: **Exemple de dendrogramme réalisé à partir de la représentation d'un modèle entraîné avec entropie croisée.** Ce modèle a été entraîné sur le jeu de données CIFAR-10. Certaines des relations exposées par ce dendrogramme peuvent être retrouvées dans la représentation t-SNE de ce même espace de représentation (figure 3.7).

permet de différencier les classes inconnues des classes connues. Pour obtenir cette représentation, des ancres sont fixées dans l'espace de représentation et une fonction de perte utilisant la distance euclidienne a pour objectif de pénaliser les instances qui sont éloignées de leurs ancres. Pour capturer un certain degré de sémantique des données, les instances sont autorisées à être représentées dans des hypersphères autour des ancres.

Les caractéristiques de la représentation que nous souhaitons obtenir avec DIFAIR permettent de répondre aux problématiques de l'OSR en utilisant la distance euclidienne au moment de l'inférence. Les classes inconnues doivent être représentées éloignées de toutes les ancres, car elles ne partagent pas toutes les caractéristiques des classes connues. Une instance est donc considérée comme appartenant à une classe inconnue quand elle est éloignée de toutes les ancres d'une distance supérieure à un seuil ϵ. Dans le cas où l'instance n'est pas inconnue, la classe prédite est la classe de l'ancre la plus proche.

Gestion des classes inconnues. DIFAIR se distingue de la plupart des approches existantes en ne fermant pas la voie à la capture d'informations sur les données inconnues. Ce point de vue est partagé par Cevikalp et al. (2023), qui disent ne pas considérer les données inconnues connues comme une unique classe à représenter en un seul point dans l'espace de représentation. Par opposition, Chen et al. (2021), avec leur approche ARPL+CS, tentent de regrouper la représentation des classes inconnues à proximité des *reciprocal points*. Dans DIFAIR, l'objectif est d'activer les attributs quand des caractéristiques associées sont présentes dans l'image. Or, les classes inconnues peuvent présenter certaines caractéristiques des classes connues, cela veut donc dire que les classes inconnues ne seront pas représentées uniquement à l'origine, comme ce serait le cas si aucune activation n'avait lieu. Au contraire, elles pourraient être représentées à proximité de classes connues, ce qui renforce l'aspect sémantique de la représentation. Bien que cela excède le cadre du présent travail, nous pensons que de telles informations peuvent être utilisées pour étiqueter à gros grain ces données de classe indéterminée.

Cette considération des classes inconnues peut compliquer la séparation des classes connues des classes inconnues, car ces dernières peuvent être représentées dans une certaine proximité des ancres selon les attributs activés. Cependant, nous considérons qu'il est important que cette spécificité soit prise en compte : car cela pourrait permettre d'obtenir des informations sur les caractéristiques des classes inconnues. Par exemple, si des instances de classes inconnues forment des groupes dans l'espace de représentation, il est possible d'inférer à partir d'un certain seuil d'instances, qu'une nouvelle classe doit être apprise par le modèle.

Risque de classification en *open-space*. L'approche DIFAIR permet de limiter le risque de classification en *open-space* en acceptant les prédictions uniquement dans K hypersphères différentes. Notons que lors des tests, ces hypersphères ne correspondent pas forcément à celles utilisées lors de l'apprentissage, car leur rayon est paramétré par ϵ. Par opposition, une approche utilisant l'entropie croisée, évaluée avec le MLS comme proposé par Vaze et al. (2022), ne limite pas ce risque. S'il arrivait qu'une classe inconnue active des attributs de manière anormalement élevée, voire davantage que pour une classe connue, la prédiction qui en découlera peut indiquer une classe avec une probabilité suffisamment élevée pour ne pas être détectée comme inconnue, alors qu'il pourrait s'agir d'une anomalie.

Dans le cas de DIFAIR, si un attribut est fortement activé de manière anormale, cela n'implique pas forcément une classification, car cette particularité peut placer l'instance à l'extérieur d'une hypersphère.

Avec DIFAIR, les erreurs de détection des classes inconnues sont donc réduites, car les classes inconnues ne sont pas censées être représentées à l'intérieur des hypersphères. Cependant, il peut arriver que des classes inconnues soient représentées dans les hypersphères, selon le seuil ϵ choisi : cette situation est décrite plus en détails dans le chapitre 5.

La méthode DIFAIR a été testée sur le benchmark d'OSR proposé par Neal et al. (2018) (chapitre 2). Comme nous le verrons, l'interprétation des représentations apprises par DIFAIR montre certaines limites de notre approche mais exhibe des résultats prometteurs. Un processus itératif d'amélioration du modèle en adéquation avec nos objectifs, ainsi qu'une recherche d'hyperparamètres rigoureuse a permis d'obtenir ces résultats. Ceux-ci sont comparés à ceux obtenus par des approches de l'état de l'art présentées dans le chapitre 2.

Chapitre 4

Analyse conceptuelle du modèle DIFAIR

Résumé

Ce chapitre présente les résultats obtenus avec DIFAIR en utilisant une fonction de perte basée sur la distance euclidienne entre les instances et les ancres. Les premiers résultats obtenus ont conduit à la proposition d'un terme de perte pour améliorer l'extraction de caractéristiques distinctes sur les dimensions d'une même classe. L'analyse des représentations apprises a joué un rôle important, révélant les avantages de DIFAIR et les limites à aborder. À la suite de cela, une recherche d'hyperparamètres a été effectuée dans le but d'améliorer les performances de DIFAIR et comprendre l'influence de chacun des paramètres sur les performances.

Sommaire

Pour répondre aux problématiques de l'*Open-Set Recognition* (OSR) nous avons présenté dans le chapitre 3 une méthode d'apprentissage de représentations différenciées nommée *DIFferentiAted Images Representations* (DIFAIR). L'objectif est d'obtenir des représentations où chaque attribut est associé à une classe et s'active uniquement en présence de cette classe, excepté si l'attribut représente une ou plusieurs caractéristiques partagées par d'autres classes. Pour ce faire, chaque classe est représentée par une *ancre* dans l'espace de représentation et les instances de cette classe doivent être représentées dans une hypersphère autour de cette ancre.

4.1 Fonction de perte euclidienne

La fonction de perte utilisée pour entraîner DIFAIR est basée sur la distance euclidienne entre les représentations des instances $\boldsymbol{X}$ et les ancres $\mathcal{A}^{\mathrm{c}}$ de chaque classe c. La spécificité de notre méthode est d'autoriser les instances à être représentées dans une hypersphère de rayon r autour des ancres. La fonction de perte est définie comme suit :

$$\mathcal{L}_{\text{out}}(\boldsymbol{X}, y) = \max(\text{Eucl}(\boldsymbol{z}, \mathcal{A}^{(y)}) - r, 0), \tag{4.1}$$

où $\boldsymbol{z}$ est la représentation de l'instance $\boldsymbol{X}$ et $\mathcal{A}^{(y)}$ est l'ancre de la classe y. La représentation d'une instance *en dehors* de l'hypersphère associée à sa classe est donc pénalisée. Si l'instance est à l'intérieur de l'hypersphère, la perte vaut 0, l'objectif étant de laisser le modèle représenter librement (sans contrainte) les instances dans l'hypersphère.

Nous présentons dans cette section les expériences réalisées avec la fonction de perte $\mathcal{L}_{\text{out}}$ et les résultats obtenus. Enfin, ces résultats sont analysés afin de comprendre les limites de cette fonction et de proposer des solutions pour y remédier.

4.1.1 Notations et expériences

Notations. Commençons par formaliser un réseau de neurones avec les notations suivantes :

$$C(\boldsymbol{X}) = \Theta \circ \phi(\boldsymbol{X}),$$

où ϕ désigne l'extracteur d'attributs et Θ est la tête de classification, la notation $\circ$ indique la composition des fonctions. Notons $\boldsymbol{z} = \phi(\boldsymbol{X})$ la représentation de l'instance $\boldsymbol{X}$ apprise par l'extracteur d'attributs et $\boldsymbol{o} = \Theta(\boldsymbol{z})$ les *logits* prédits par le réseau, c'est-à-dire qu'aucune fonction d'activation n'est appliquée à $\boldsymbol{o}$. La classe prédite par le réseau est alors $\hat{y} = \arg\max \boldsymbol{o}$, ce qui est équivalent à $\hat{y} = \arg\max \text{softmax}(\boldsymbol{o})$.

L'extracteur d'attributs utilisé pour ces expériences est l'encodeur proposé par Neal et al. (2018), qui est classiquement utilisé pour évaluer les approches en OSR. L'architecture de cet encodeur, noté par ϕ_{OSR}, est détaillée dans la section 2.1.4.

Pour évaluer les performances en OSR, nous avons d'abord comparé DIFAIR à un réseau *baseline* entraîné avec la fonction de perte d'entropie croisée, il s'agit de l'expérience de contrôle. Les architectures de ces deux modèles sont présentées dans la figure 3.1 (section 3.1.1). À partir des notations introduites, considérons le réseau *baseline* :

$$C_{\text{EC}}(\boldsymbol{X}) = \boldsymbol{o}_{\text{EC}} = \Theta_{\text{EC}} \circ \phi_{\text{OSR}}(\boldsymbol{X}),$$

où Θ_{EC} désigne la tête de classification du réseau *baseline* et l'indice EC signifie entropie croisée. Le réseau utilisé pour entraîner DIFAIR est noté :

$$C_{\text{DIFAIR}}(\boldsymbol{X}) = \boldsymbol{z}_{\text{D}} = \phi_{\text{DIFAIR}} \circ \phi_{\text{OSR}}(\boldsymbol{X}),$$

où ϕ_{DIFAIR} est l'ensemble des couches ajoutées à l'extracteur d'attributs pour obtenir la représentation différenciée $\boldsymbol{z}_{\text{D}}$. Dans cette première version de DIFAIR, nous avons fait le choix de ne pas appliquer de fonction d'activation à la sortie de la dernière couche de convolution (c'est-à-dire dans ϕ_{DIFAIR}), afin de laisser une liberté maximale à l'extracteur d'attributs pour placer les instances dans l'espace de représentation.

Scores d'OSR. À partir de ces deux modèles, nous extrayons plusieurs scores d'OSR. Vaze et al. (2022) ont proposé le *Maximum Logit Score* pour améliorer les performances par rapport au score initial, la MSP. Nous proposons d'adapter la terminologie du MLS en proposant le *Maximum Output Score* (MOS), consistant à utiliser la valeur maximale de la *sortie* d'un réseau de neurones comme score d'OSR, qu'il s'agisse de logits comme pour la *baseline* ou d'une représentation, comme c'est le cas pour DIFAIR. Le score d'OSR correspondant à la méthode MOS est donc $S(y \in \mathbb{C}|\boldsymbol{X}) = \max \boldsymbol{s}$, avec $\boldsymbol{s}$ la sortie du réseau.

La représentation $\boldsymbol{z}_{\text{EC}} = \phi_{\text{OSR}}(\boldsymbol{X})$ est extraite du modèle C_{EC} afin de la comparer à la représentation apprise par DIFAIR. Un score d'OSR similaire à celui utilisé par DIFAIR est proposé à partir de cette représentation. Pour une instance donnée, la distance euclidienne est mesurée entre sa représentation et les centres $\boldsymbol{m}^{(c)}$ de chaque classe c. Le centre de la classe c est la représentation moyenne des données étiquetées c *correctement classées*, cela afin de ne pas introduire de bruit dans le cas où la donnée serait mal représentée. Cette expérience est nommée *Cross-Entropy Learned Representation* (CELR).

Les scores extraits pour ces deux modèles sont donc :

— **MSP** : $S(y \in \mathbb{C}|\boldsymbol{X}) = \max \text{softmax}(\boldsymbol{o}_{\text{EC}})$.

— **MLS** : $S(y \in \mathbb{C}|\boldsymbol{X}) = \max \boldsymbol{o}_{\text{EC}}$.

— **CELR** : $S(y \in \mathbb{C}|\boldsymbol{X}) = \min \boldsymbol{d}_{\text{EC}}$, avec $\boldsymbol{d}_{\text{EC}}$ le vecteur de distances entre la représentation $\boldsymbol{z}_{\text{EC}}$ de l'instance et les centres de chaque classe.

— **DIFAIR** : $S(y \in \mathbb{C}|\boldsymbol{X}) = \min \boldsymbol{d}$, avec $\boldsymbol{d}$ le vecteur de distances entre la représentation de l'instance $\boldsymbol{z}_{\text{D}}$ et toutes les ancres.

— **DIFAIR(MOS)** : $S(y \in \mathbb{C}|\boldsymbol{X}) = \max \boldsymbol{z}_{\text{D}}$.

Évaluation. L'évaluation de chacun de ces scores d'OSR est réalisée sur le benchmark proposé par Neal et al. (2018), décrit dans la section 2.1.3 du chapitre 2. Ce benchmark contient six tâches de détection de classes inconnues. La performance de détection des classes inconnues est mesurée avec l'AUROC. L'Exactitude en *Closed-Set* (ECS) est reportée pour les deux modèles (C_{EC} et C_{DIFAIR}) et en utilisant CELR (la classe du centre le plus proche est assignée à chaque instance). Les résultats présentés sont la moyenne des résultats obtenus sur 5 séparations aléatoires de classes connues/inconnues pour chaque tâche. Pour garantir une comparaison équitable des performances de nos modèles avec d'autres études, nous avons utilisé les mêmes séparations aléatoires que Vaze et al. (2022).

Entraînement. L'entraînement du réseau *baseline* ainsi que du modèle utilisant DIFAIR est réalisé suivant le protocole amélioré proposé par Vaze et al. (2022). Ce protocole implique un entraînement pendant 600 époques, l'utilisation de l'augmentation de données *RandAugment* (Cubuk et al., 2020) et la planification du taux d'apprentissage au cours de l'entraînement. Le taux d'apprentissage initial utilisé est de 0.1 pour toutes les tâches sauf pour TinyImageNet où il vaut 0.01 (valeurs utilisées par Vaze et al. (2022)). Le type de planification utilisé est nommé *Cosine Decay Restart* (Loshchilov & Hutter, 2016) : le taux d'apprentissage diminue de sa valeur initiale vers une valeur faible prédéterminée en suivant une fonction cosinus et est remis à sa valeur initiale toutes les 200 époques. L'effet du « redémarrage » de l'apprentissage avec un taux d'apprentissage à nouveau élevé est équivalent à commencer un apprentissage avec des poids qui sont déjà correctement initialisés, ce qui permet d'améliorer la recherche du minimum de la fonction de perte (Loshchilov & Hutter, 2016). DIFAIR est paramétré avec $\alpha = 10$ et $\mathcal{N} = 5$. Le rayon r est défini comme $r = 0.4 \cdot \sqrt{2\mathcal{N}\alpha^2}$, ce qui permet d'allouer 40% de l'espace (en ligne droite) entre deux ancres à chaque hypersphère,

laissant 20% de l'espace entre les ancres pour représenter les données inconnues. Le facteur de 40% a été choisi de manière empirique et de façon à respecter la contrainte selon laquelle deux hypersphères ne doivent pas se chevaucher, puisque le réseau est uniquement contraint à représenter les instances dans les hypersphères.

4.1.2 Résultats en OSR

Résultats. Le tableau 4.1 présente les résultats obtenus en OSR sur le benchmark de Neal et al. (2018). Nous avons eu des difficultés à reproduire les résultats obtenus par Vaze et al. (2022) avec le MLS, nos performances sont inférieures d'au moins 4% en AUROC sur CIFAR+10, CIFAR+50 et de 15% sur TinyImageNet. Nous spéculons que cela pourrait être lié aux différences d'implémentation entre TensorFlow (Martín Abadi et al., 2015), la bibliothèque d'apprentissage profond utilisée pour nos travaux, et PyTorch (Paszke et al., 2019), la bibliothèque utilisée par Vaze et al. (2022). Les recherches et expériences effectuées pour comprendre et répondre aux différences d'implémentation existantes n'ont pas permis de réduire ces écarts de performances. Nous remarquons tout de même qu'en considérant uniquement nos résultats, l'utilisation du MLS permet effectivement d'améliorer les performances par rapport à la MSP, comme l'ont affirmé Vaze et al. (2022).

Les scores d'AUROC obtenus par DIFAIR sont inférieurs d'au moins 16.3% par rapport à ceux obtenus par la méthode MLS de Vaze et al. (2022) sur les jeux de données d'images naturelles (soit toutes les tâches sauf MNIST et SVHN). En utilisant le MOS, les performances de DIFAIR sont améliorées mais restent inférieures à celles de la méthode MLS.

Appliquer le MOS aux représentations de DIFAIR est néanmoins plus adapté à évaluer DIFAIR en OSR par rapport à l'utilisation de la distance aux ancres comme score, démontrant une amélioration significative de l'AUROC. Une AUROC supérieure de 4.1% sur TinyImageNet et d'au moins 8.7% sur les autres jeux de données d'images naturelles est observée. Les performances sur les jeux de données MNIST et SVHN augmentent plus faiblement, mais ces jeux de données étant plus simples, les performances de base de tout modèles sont déjà élevées.

Les résultats obtenus en utilisant la représentation apprise par le modèle entraîné avec entropie croisée (CELR) sont notablement plus faibles que les autres méthodes, à l'exception des résultats obtenus sur le jeu de données MNIST.

Méthode	MNIST	SVHN	CIFAR10	CIFAR+10	CIFAR+50	TinyImageNet
MSP	99.4	95.6	89.0	90.5	89.8	66.3
MLS	**99.6**	96.9	92.0	91.8	92.1	67.9
MLS (Vaze et al., 2022)	99.3	**97.1**	**93.6**	**97.9**	**96.5**	**83.0**
CELR	99.2	87.0	57.2	66.0	69.3	53.7
DIFAIR	98.9	93.5	77.3	77.4	77.4	65.2
DIFAIR(MOS)	99.5	94.8	86.5	86.8	86.1	69.3

TABLEAU 4.1: **AUROC sur le benchmark de détection de classes inconnues de Neal et al. (2018)**. Scores d'AUROC pour la tâche de détection des classes inconnues, moyennés sur cinq séparations de classes connues/inconnues. Les résultats non cités proviennent de nos expériences. Les scores les plus élevés sont indiqués en gras.

Modèle	MNIST	SVHN	CIFAR10	CIFAR+10	CIFAR+50	TinyImageNet
C_{EC}	99.8±0.1	97.7±0.2	96.1±1.2	96.1±0.2	96.1±0.1	61.4±3.1
ϕ_{OSR} (CELR)	99.8±0.1	96.8±0.2	90.0±2.5	92.7±0.9	93.1±0.2	60.5±3.1
C_{DIFAIR}	99.8±0.1	98.0±0.2	95.8±1.2	96.2±0.3	96.2±0.1	66.8±2.3

TABLEAU 4.2: **ECS des modèles sur le benchmark de Neal et al. (2018)**. La moyenne de l'ECS (%) et l'écart-type sont reportés sur les cinq séparations de classes connues/inconnues.

L'ECS pour chaque modèle est présentée dans le tableau 4.2. Le modèle entraîné avec DIFAIR obtient une exactitude similaire à celle du modèle entraîné avec entropie croisée, excepté sur le jeu de données TinyImageNet où DIFAIR obtient une exactitude supérieure de 4.7%. L'exactitude mesurée à partir de la représentation du modèle entraîné avec entropie croisée (CELR) est inférieure par rapport aux autres modèles, sauf sur le jeu de données MNIST où les résultats sont équivalents.

Discussion. Les résultats obtenus avec CELR montrent que les représentations apprises par le réseau entraîné avec entropie croisée ne sont pas adaptées pour l'OSR basé sur les distances. Les performances plus faibles en ECS signifient que l'apprentissage avec entropie croisée ne crée pas de groupe d'instances assez compact dans l'espace de représentation pour utiliser une mesure de distance par rapport au centre (représentation moyenne) de ce groupe en tant que critère de classification. La très faible AUROC obtenue sur la détection des classes inconnues indique que des instances de classes inconnues sont sûrement représentées à proximité des centres des classes. En revanche, la tête de classification Θ_{EC} est capable d'extraire

des informations utiles de l'espace de représentation d'un modèle entraîné avec entropie croisée, puisque le score MLS permet d'obtenir les meilleurs résultats en AUROC, proches de l'état de l'art (section 2.2.2).

Dans la représentation apprise par DIFAIR, les attributs ne s'activent pas ou peu en présence de classes inconnues, car la valeur de l'attribut le plus activé (MOS) forme un bon score d'OSR par rapport à l'utilisation de la distance aux ancres. Cependant, un tel score ne permet pas de limiter le risque de classification en *open-space*, comme expliqué dans la section 3.5 du chapitre 3. Le fait que l'ECS obtenue par DIFAIR soit équivalente à celle obtenue par le modèle entraîné avec entropie croisée révèle que les représentations apprises par DIFAIR sont suffisamment distinctes pour permettre une classification correcte en utilisant la distance euclidienne par rapport aux ancres.

Cependant, les performances inférieures obtenues en AUROC avec le score initial de DIFAIR, basé sur la distance euclidienne, montrent que des données inconnues sont représentées dans un même rayon autour des ancres que des données connues. Cela peut être lié au fait que la fonction de perte ne pénalise pas la représentation des données connues lorsque celles-ci sont dans l'hypersphère, impliquant que cette flexibilité dans la représentation ne permet pas d'extraire des attributs assez représentatifs des classes connues. En effet, du moment que l'instance est représentée dans l'hypersphère, il n'y a plus de contraintes sur la représentation. Une autre cause possible de cette proximité entre instances connues et inconnues peut provenir du fait que nous ne voulons pas limiter les activations des attributs en présence de classes inconnues, celles-ci pouvant partager des caractéristiques avec des classes connues, elles peuvent donc être représentées à une certaine proximité des ancres.

Malgré des résultats inférieurs, l'activation d'attributs face à des données inconnues est une spécificité que nous souhaitons conserver dans notre méthode, car elle peut permettre d'extraire des informations sémantiques concernant les données inconnues. Ce dernier point est discuté plus amplement dans la section 3.5.

Pour aller plus loin que l'observation des métriques, nous proposons de visualiser des représentations apprises par DIFAIR afin de vérifier si elles présentent les caractéristiques souhaitées et tenter de mieux comprendre les résultats obtenus.

4.1.3 Analyse des représentations

Cette première version de DIFAIR a été entraînée avec la fonction de perte $\mathcal{L}_{\text{out}}$ (équation 4.1), et l'architecture utilisée n'applique pas de fonction d'activation à la sortie de la dernière couche de convolution, afin de laisser une liberté maximale à l'extracteur d'attributs pour placer les instances dans l'espace de représentation. L'analyse des représentations apprises par cette version de DIFAIR a directement exposé un problème : en effet, comme le montre la figure 4.1, les attributs extraits associés à une même classe ont des valeurs très proches, voire égales. Or, l'objectif de DIFAIR étant d'extraire des attributs distincts pour une classe, le fait que tous les attributs aient la même valeur suggère qu'ils ne le sont pas. Les expériences présentées dans la suite de cette section visent à vérifier cette hypothèse.

Dans la situation où les attributs d'une même classe sont dupliqués, une explication possible est que les poids associés à chacun de ces attributs ont convergé vers des valeurs similaires, et donc que la même information a été extraite de la couche précédente. La figure 4.2 illustre une simplification de ce problème dans le cadre d'un réseau de neurones entièrement connecté et montre les poids en jeu ainsi que les conséquences d'une convergence de ces poids. Dans cette situation, le problème de duplication de l'information extraite survient lorsque les poids reliant un neurone de la couche $L-1$

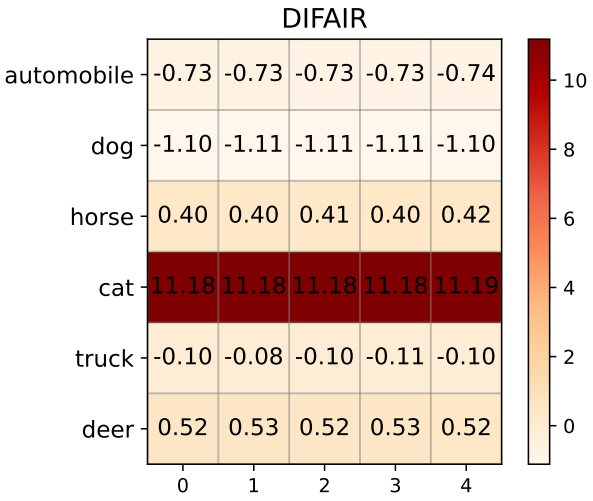


Figure 4.1: **Représentation d'une image de chat obtenue avec DIFAIR**. Le modèle utilisé pour obtenir cette représentation a été entraîné sur la séparation numéro 2 du jeu de données CIFAR-10 lors de l'expérience présentée en section 4.1.2.

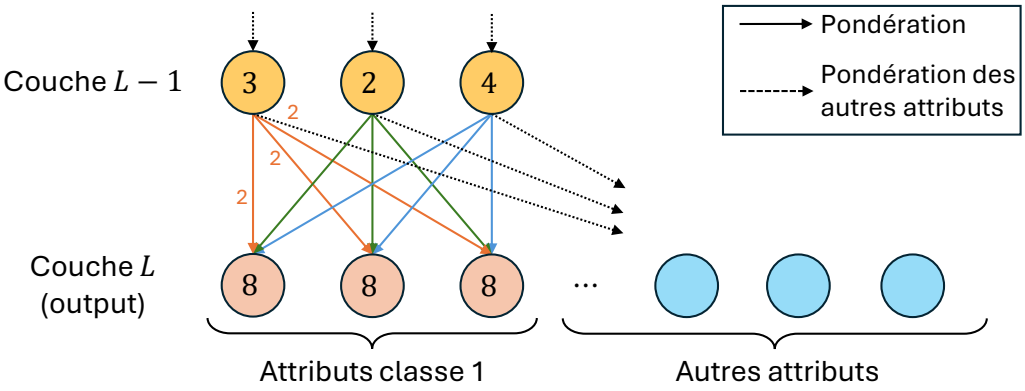


FIGURE 4.2: **Schéma des poids de la dernière couche d'un réseau entièrement connecté**. Considérons un exemple simplifié avec trois neurones sur la couche $L-1$. Les valeurs dans les neurones représentent leur valeur d'activation. Si trois attributs de la couche L associés à une même classe s'activent avec une même valeur, alors une possibilité est que la même information ait été extraite des neurones de la couche $L-1$. Par exemple, si les poids orange ont tous la valeur 2, les poids verts la valeur 0.5 et les poids bleus la valeur 0.25, alors tous les attributs de la classe 1 seront activés avec la valeur 8, et l'information extraite de la couche $L-1$ est dupliquée sur chacun de ces attributs.

à tous les neurones d'une même classe ont convergé vers des valeurs similaires, et ce, pour tous les neurones de la couche $L-1$. Dans notre cas, l'architecture du réseau de neurones est plus complexe qu'une architecture entièrement connectée, mais le problème est similaire : chaque poids est en fait un filtre (ensemble de poids) appliqué sur l'ensemble des cartes d'activation de la couche $L-1$. Par exemple, si deux filtres sont identiques, les cartes d'activation résultant de l'application de ces deux filtres seront aussi identiques. Ainsi, après avoir appliqué une fonction d'activation et une opération de *global average pooling* à ces deux cartes d'activation, les deux attributs obtenus en sortie seront également identiques.

La couche de convolution ajoutée après l'extracteur d'attributs dans l'approche DIFAIR possède $\mathcal{N} \cdot K$ filtres, dans le but d'extraire autant d'attributs, où $\mathcal{N}$ le nombre de dimensions allouées à chaque classe et K est le nombre de classes. Commençons par introduire $\mathbb{I}$, l'ensemble d'indices des filtres de cette couche :

$$\mathbb{I} = \{1, \ldots, \mathcal{N} \cdot K\}.$$

Nous définissons ensuite $\mathbb{I}_c$, sous-ensemble de $\mathbb{I}$, contenant les indices des filtres associés à une classe $c \in \{1, \ldots, K\}$:

$$\mathbb{I}_c = \{(c-1) \cdot \mathcal{N} + 1, \ldots, (c-1) \cdot \mathcal{N} + \mathcal{N}\},$$

avec :

$$\bigcup_{c=1}^{K} \mathbb{I}_c = \mathbb{I} \quad \text{et} \quad \bigcap_{c=1}^{K} \mathbb{I}_c = \emptyset.$$

À partir de ces indices, notons $\mathbb{W}$ l'ensemble des filtres de la dernière couche de convolution :

$$\mathbb{W} = \{\boldsymbol{W}^{(i)},\ i \in \mathbb{I}\},$$

avec $\boldsymbol{W}^{(i)}$ une matrice à trois dimensions, et définissons l'ensemble $\mathbb{W}_c$, sous-ensemble de $\mathbb{W}$ contenant les filtres associés à une classe c :

$$\mathbb{W}_c = \{\boldsymbol{W}^{(i)},\ i \in \mathbb{I}_c\}.$$

Pour simplifier les calculs, nous considérons dans la suite que les filtres $\boldsymbol{W}^{(i)}$ sont représentés sous une forme aplatie, en un vecteur de taille n_f. Les éléments de ce vecteur sont notés $W_j^{(i)}$, avec $j \in \{1, \ldots, n_f\}$.

Un ensemble de filtres $\mathbb{W}_c$ est considéré comme convergent si les poids de même indice de chaque filtre de cet ensemble convergent : dans cette situation les filtres tendent à devenir identiques. L'ensemble $\mathbb{S}_{j,c}$ des poids d'indice j des filtres de la classe c est noté :

$$\mathbb{S}_{j,c} = \{W_j^{(i)},\ i \in \mathbb{I}_c\}. \tag{4.2}$$

Afin de vérifier si ces poids convergent vers une même valeur au cours de l'apprentissage, nous observons l'écart-type de l'ensemble $\mathbb{S}_{j,c}$. En cas de convergence, cet écart-type diminuera au cours du temps et sera faible, signe d'une faible dispersion des poids. La métrique globale pour évaluer si les filtres associés à une classe d'indice c convergent vers des valeurs similaires est la moyenne des écarts-types de tous les ensembles de poids $\mathbb{S}_{j,c}$ associés à cette classe, notée :

$$\mathrm{CONV}(c) = \frac{1}{n_f} \sum_{j=1}^{n_f} \mathrm{std}(\mathbb{S}_{j,c}), \tag{4.3}$$

avec $\mathrm{std}(\mathbb{S}_{j,c})$ l'écart-type de l'ensemble de poids $\mathbb{S}_{j,c}$. Cette métrique est calculée indépendamment pour chaque classe c afin de pouvoir observer si la convergence des poids est spécifique à certaines classes ou non.

La figure 4.3 montre l'évolution de cette métrique pour un modèle entraîné avec DIFAIR sur la séparation 2 de CIFAR-10. Les coudes dans la courbe sont liés au taux d'apprentissage variable (tel que décrit page 106). L'écart-type moyen est faible et diminue pour chaque classe, ce qui veut dire que les valeurs des filtres associés à une même classe deviennent de plus en plus similaires. Plus précisément, la figure 4.4 montre l'évolution du premier poids $W_1^{(}i)$ pour les filtres associés à deux classes différentes. Ces exemples montrent que les poids tendent vers une même valeur au cours de l'apprentissage. L'utilisation de la métrique avec l'écart-type permet cependant d'avoir une vision plus globale de la convergence des poids. Ces exemples confirment l'hypothèse que les informations extraites par la couche de convolution ajoutée sont dupliquées pour les attributs de chaque classe. Ce n'est pas le résultat souhaité en utilisant l'approche DIFAIR, où les attributs extraits devraient être distincts.

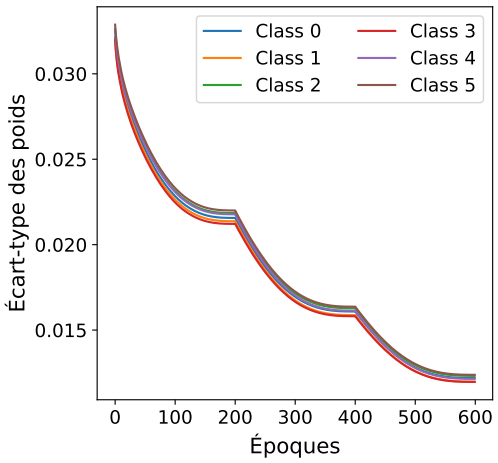
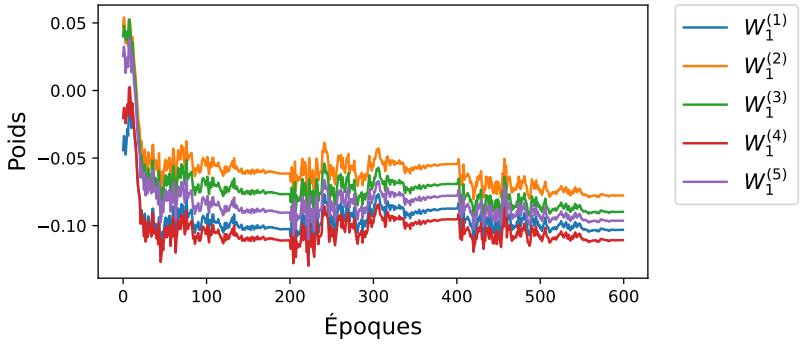
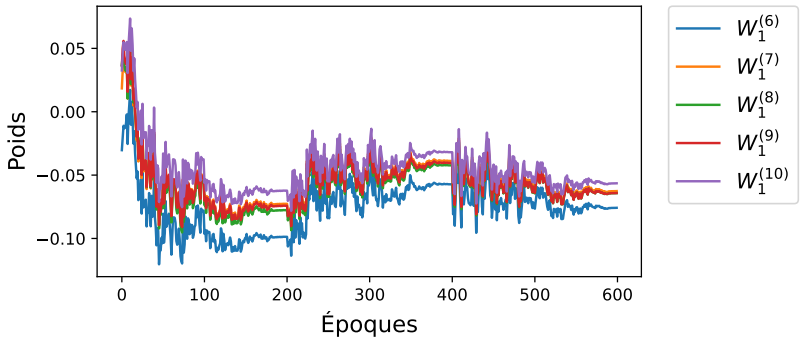


FIGURE 4.3: **Évolution de la moyenne des écarts-types des groupes de poids de la dernière couche de convolution**. La métrique définie dans l'équation 4.3 est évaluée pour chaque classe au fil de l'apprentissage.

Ce comportement a probablement émergé car il permet de satisfaire plus facilement l'objectif de DIFAIR qui est d'activer avec une valeur élevée les attributs associés à une même classe. La convergence des poids permet de toujours avoir les attributs de la classe activés en même temps et avec la même valeur. L'émergence de ce comportement illustre la difficulté de formaliser des objectifs d'apprentissage.



(a)



(b)

Figure 4.4: **Évolution des poids associés à une même classe**. (a) Évolution des poids de l'ensemble $\mathbb{S}_{1,1}$ (premiers poids de chaque filtre associé à la classe 1). (b) Évolution des poids de l'ensemble $\mathbb{S}_{1,2}$ (premiers poids de chaque filtre associé à la classe 2).

Puisque l'objectif initial de DIFAIR est d'extraire des attributs distincts pour chaque classe, nous proposons de modifier la fonction de perte afin de pénaliser le cas où les poids associés à une même classe convergent. Nous détaillons cette modification dans la section suivante.

4.2 Prévenir la convergence des poids

Dans cette section, deux fonctions de perte contraignant le comportement des poids de la dernière couche sont proposées. La première est basée sur l'écart-type des poids associés à une même classe, la seconde sur la corrélation entre ces poids.

4.2.1 Écart-type

Puisque nous avons mesuré la convergence des poids avec l'écart-type, la première idée pour empêcher cette convergence est de pénaliser l'apprentissage lorsque cet écart-type est faible. Cela signifie que si le modèle apprend une solution où les poids associés à une même classe deviennent trop proches, la valeur de perte sera augmentée et le modèle devra trouver une autre solution pour minimiser la fonction de perte.

Définissons dans un premier temps la valeur CONV que nous voulons pénaliser. Il s'agit de l'écart-type moyen des poids toutes classes confondues, la formule diffère donc légèrement de celle de l'équation 4.3 :

$$\mathrm{CONV} = \frac{1}{K \cdot n_f} \sum_{c=1}^{K} \sum_{j=1}^{n_f} \mathrm{std}(\mathbb{S}_{j,c}). \tag{4.4}$$

Nous proposons deux fonctions de perte pour pénaliser la diminution de la valeur CONV, la première est exponentielle et la seconde linéaire. La fonction de perte exponentielle $\mathcal{L}_{\exp}$ est la suivante :

$$\mathcal{L}_{\exp}(\mathrm{CONV}) = \alpha \cdot \exp\left(-\beta \cdot \mathrm{CONV}\right), \tag{4.5}$$

avec

$$\beta = \frac{1}{\tau} \cdot \log\left(\frac{\alpha}{\epsilon}\right).$$

La valeur β est fixée de telle sorte que la fonction de perte vaille un ϵ très faible lorsque $\mathrm{CONV} = \tau$, car la fonction exponentielle ne peut pas être nulle. Cette fonction est donc paramétrée par (α, τ, ϵ).

La fonction de perte linéaire $\mathcal{L}_{\mathrm{lin}}$ est définie comme suit :

$$\mathcal{L}_{\mathrm{lin}}(\mathrm{CONV}) = -\gamma \cdot \mathrm{CONV} + \gamma \cdot \tau. \tag{4.6}$$

Cette fonction est paramétrée par (γ, τ), où γ représente la pente de la fonction et τ est la valeur pour laquelle la perte vaut 0.

Le tracé de ces fonctions est présenté dans la figure 4.5 avec différents paramètres pour τ, la valeur d'écart-type pour laquelle la perte vaut 0 ou s'en approche. La fonction exponentielle pénalise plus fortement les valeurs faibles de CONV que la fonction linéaire, mais elle s'approche de 0 plus rapidement, donc la pénalisation devient moins forte pour des valeurs de CONV plus élevées. La fonction linéaire permet une pénalisation plus prononcée jusqu'à la valeur d'écart-type souhaitée.

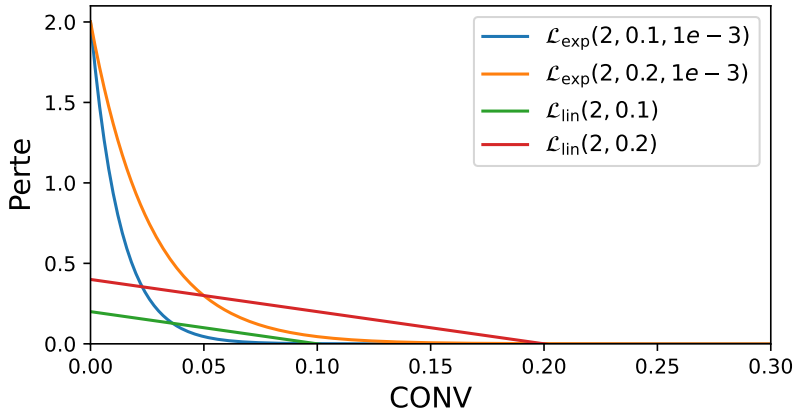


Figure 4.5: **Fonctions de perte pénalisant l'écart-type**. Les fonctions exponentielles et linéaires sont représentées avec des paramètres différents pour τ, la valeur pour laquelle la perte vaut 0 ou s'en approche, autrement dit l'écart-type souhaité.

4.2.2 Corrélation

L'évolution des poids au fil des époques, illustrée précédemment dans la figure 4.4, indique que les poids convergents semblent aussi être linéairement corrélés.

La corrélation linéaire entre deux variables aléatoires X et Y est exprimée par le coefficient de corrélation de Pearson, défini comme :

$$\text{Cor}(X, Y) = \frac{\text{Cov}(X, Y)}{\sigma_X \cdot \sigma_Y},$$

avec $\text{Cov}(X, Y)$ la covariance de X et Y, et σ_X et σ_Y les écarts-types de X et Y respectivement. Ce coefficient est compris entre -1 et 1, une valeur de 1 indiquant une corrélation linéaire parfaite, c'est-à-dire qu'il existe une relation affine telle que $Y = aX + b$ entre les deux variables. La valeur -1 indique une corrélation linéaire négative parfaite, et 0 indique l'absence de corrélation linéaire. Par soucis de simplification, nous utiliserons le terme « corrélation » pour évoquer la corrélation linéaire dans la suite de cette section.

Dans le cas où une corrélation existe entre deux filtres associés à une même classe, cela signifie que les deux filtres extraient la même information, mais à une transformation affine près. Tout comme lorsque les poids

des filtres convergent, la corrélation des filtres ne permet pas d'obtenir des attributs distincts, car il s'agit seulement de la même information extraite avec une transformation linéaire.

Bien que les hypothèses que nous essayons d'examiner avec DIFAIR soient différentes de l'apprentissage d'un modèle avec l'entropie croisée, il est tout de même intéressant d'observer comment les informations sont extraites par un tel modèle afin de comprendre les différences avec notre méthode et estimer comment devraient être distribués les poids de la dernière couche de convolution pour DIFAIR.

La matrice de corrélation entre chacun des filtres de la dernière couche de convolution est présentée dans la figure 4.6a pour un modèle entraîné avec entropie croisée et dans la figure 4.6b pour un modèle entraîné avec DIFAIR. Ces résultats montrent que les filtres de DIFAIR associés à une même classe sont fortement corrélés, avec une corrélation très proche de 1. À l'inverse, pour le modèle entraîné avec entropie croisée, possédant 128 filtres (98 de plus que DIFAIR) aucune corrélation aussi élevée n'est observée. Afin de comparer plus facilement la corrélation des filtres du modèle entraîné avec entropie croisée par rapport au modèle entraîné avec DIFAIR, la figure 4.6c présente la distribution des coefficients de corrélation pour chacun des modèles. Les coefficients de corrélation sont distribués autour de 0 pour le modèle entraîné avec entropie croisée, ce qui peut s'expliquer par le fait que le modèle apprend à extraire des informations différentes pour chaque classe à reconnaître, et donc que les filtres ne sont pas corrélés. À l'inverse, pour le modèle entraîné avec DIFAIR, les coefficients de corrélation sont soit proches de 1 soit entre -0.25 et -0.05. Ce qui veut dire qu'il semble exister une relation, même très faible, entre les filtres associés à des classes différentes.

Pour éviter l'extraction d'informations redondantes, nous proposons une pénalisation de la corrélation entre les filtres $\mathbb{W}_c$ pour chacune des classes c. Cela signifie que la corrélation entre des filtres de différentes classes est autorisée, ce qui s'aligne sur l'idée de DIFAIR que certaines caractéristiques peuvent être partagées entre plusieurs classes. Par conséquent, la pénalisation de la corrélation n'a pas d'impact négatif sur la sémantique de la représentation. La fonction de perte $\mathcal{L}_{\text{correlation}}$ est notée :

$$\mathcal{L}_{\text{correlation}} = \sum_{c=1}^{K} \rho_c, \tag{4.7}$$

avec :

$$\rho_c = \frac{2}{\mathcal{N}(\mathcal{N}-1)} \sum_{\substack{i,j \in \mathbb{I}_c \\ j>i}} \left| \mathrm{Cor}(\boldsymbol{W}^{(i)}, \boldsymbol{W}^{(j)}) \right|,$$

où $|\cdot|$ représente la valeur absolue et ρ_c est la moyenne de la valeur absolue de la corrélation entre chacun des filtres de l'ensemble $\mathbb{W}_c$, associés à la classe c. La somme des ρ_c est pénalisée afin que la corrélation soit faible pour chacune des classes.

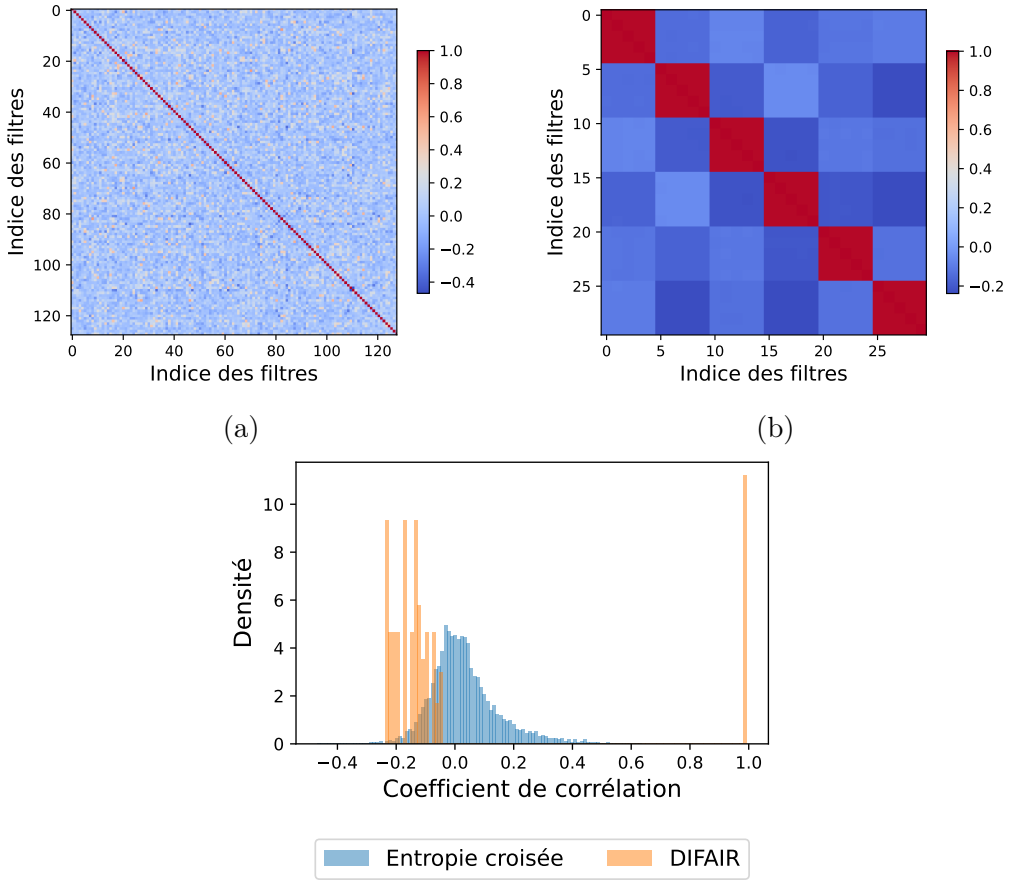


Figure 4.6: **Corrélation des filtres de la dernière couche de convolution**. (a) Matrice de corrélation des filtres pour un modèle entraîné avec entropie croisée. (b) Matrice de corrélation des filtres pour un modèle entraîné avec DIFAIR. (c) Distribution des coefficients de corrélation entre les filtres pour les modèles entraînés avec entropie croisée et DIFAIR.

Cette fonction de perte force la corrélation linéaire des filtres associés à une même classe à être proche de 0, ce qui permet d'éviter l'extraction de la même information par plusieurs filtres et devrait donc permettre d'extraire des attributs plus distincts.

4.3 Affinage du modèle proposé

Nous considérons que les résultats présentés dans la section 4.1 sont obtenus avec une version de DIFAIR que nous nommons DIFAIR v0. Cette version utilise la fonction de perte $\mathcal{L}_{\text{out}}(\boldsymbol{X}, y)$ (équation 4.1), n'applique pas de fonction d'activation à la sortie de la dernière couche de convolution, et ne pénalise pas les filtres de cette-dernière. Les premiers résultats obtenus avec DIFAIR v0 montrent que le modèle est capable, dans une certaine mesure, d'activer les attributs associés à une classe en présence de celle-ci, et de les désactiver en l'absence des classes. En effet, cela est corroboré par le fait que l'AUROC obtenue avec le MOS est plus élevée qu'en utilisant le score basé sur la distance aux ancres. Le problème majeur dans ce cas est que les poids de la dernière couche ont tendance à être corrélés et à converger vers une même valeur, ce qui empêche d'extraire des attributs distincts pour une même classe.

Puisque des modifications de la perte sont nécessaires, nous proposons d'entraîner plusieurs configurations de DIFAIR dans le but de les comparer et, dans le même temps, d'effectuer une recherche d'hyperparamètres afin d'observer l'influence de ceux-ci sur les performances. Pour ces expériences, la même architecture du modèle est utilisée et est entraînée sur la séparation 2 de CIFAR-10, où les classes connues sont : automobile, chien, cheval, chat, camion, cerf et les classes inconnues sont : oiseau, avion, bateau, grenouille. Les performances sont mesurées sur le jeu de données de validation. Chaque configuration est entraînée cinq fois puis les résultats sont moyennés afin d'obtenir des résultats plus significatifs de la performance de la configuration.

Les configurations évaluées considèrent différentes fonctions de perte, en ajoutant les termes de pénalisation des poids présentés dans la section 4.2, et en modifiant la fonction d'activation de la représentation (voir le schéma de l'architecture de DIFAIR présenté dans la figure 3.1, à la section 3.1.1 du chapitre 3). Les hyperparamètres recherchés sont le taux d'apprentissage, le nombre d'attributs $\mathcal{N}$ associé à chaque classe, le paramètre α qui est la valeur d'activation cible sur chaque dimension et le rayon r de l'hypersphère allouée autour des ancres.

Les différents critères d'évaluation des modèles sont l'Exactitude en *Closed-Set* (ECS), l'AUROC pour la détection des classes inconnues, la métrique CONV pour l'écart-type des poids présentée dans l'équation 4.4 ainsi que la corrélation des filtres associés à une même classe, $\mathcal{L}_{\text{correlation}}$, présentée dans l'équation 4.7. Nous observons également le pourcentage de données connues correctement représentées à l'intérieur des hypersphères de rayon r, exprimé par la métrique PH (Pourcentage dans les Hypersphères) notée :

$$\text{PH} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{d \leq r\}},$$

où n est le nombre de données connues et $d = \text{Eucl}(\boldsymbol{z}^{(i)}, \mathcal{A}^{(y^{(i)})})$ la distance entre la représentation $\boldsymbol{z}^{(i)}$ de l'instance $\boldsymbol{X}^{(i)}$ et l'ancre associée à la classe $y^{(i)}$ de cette instance. La fonction indicatrice $\mathbb{1}_{\{d \leq r\}}$ vaut 1 si la distance d est inférieure ou égale à r, c'est-à-dire lorsque l'instance est représentée dans l'hypersphère associée à sa classe, et 0 sinon. Ces métriques sont calculées sur le jeu de données de validation pour ne pas biaiser les résultats.

4.3.1 Recherche de la meilleure configuration

La recherche de la meilleure configuration et des meilleurs hyperparamètres est effectuée séparément, car le nombre de combinaisons pour toutes les variantes possibles est trop important pour être exploré en une seule fois. Commençons par rechercher la meilleure configuration pour appliquer DIFAIR.

Nous proposons d'évaluer une autre fonction de perte, car les modèles entraînés avec la fonction de perte $\mathcal{L}_{\text{out}}(\boldsymbol{X}, y)$ (équation 4.1) présentent des résultats en OSR plus faibles en utilisant le score de DIFAIR qu'en utilisant le MOS (voir le tableau 4.1). Rappelons que cette fonction ne pénalise pas les instances représentées dans l'hypersphère, car l'objectif était de laisser une certaine liberté de représentation aux instances connues. Il semble que cette fonction ne permette pas d'extraire des attributs suffisamment représentatifs des classes connues pour les séparer des classes inconnues. Pour tenter de pallier ce problème, nous proposons d'utiliser une fonction de perte pénalisant aussi la représentation dans l'hypersphère, afin de forcer une instance à être représentée proche de son ancre. Néanmoins, l'idée est de ne pas donner autant d'importance à l'objectif de représenter l'instance proche de l'ancre qu'à celui de la représenter dans l'hypersphère. Nous proposons la fonction de perte $\mathcal{L}_{\text{in/out}}(\boldsymbol{X}, y)$ définie par :

$$\mathcal{L}_{\text{in/out}}(\boldsymbol{X}, y) = \mathcal{L}_{\text{in}}(\boldsymbol{X}, y) + \mathcal{L}_{\text{out}}(\boldsymbol{X}, y), \tag{4.8}$$

avec :

$$\mathcal{L}_{\text{in}}(\boldsymbol{X}, y) = 0.1 \cdot \text{Eucl}(\boldsymbol{z}, \mathcal{A}^{(y)}),$$

où $\boldsymbol{z}$ est la représentation de l'instance $\boldsymbol{X}$. Pour différencier ces fonctions, nous nommons fonction de perte « forte » la fonction $\mathcal{L}_{\text{out}}(\boldsymbol{X}, y)$ (équation 4.1) et fonction de perte « faible » la fonction $\mathcal{L}_{\text{in/out}}(\boldsymbol{X}, y)$. La figure 4.7 montre les valeurs de ces fonctions de perte en fonction de la distance d'une instance à son ancre.

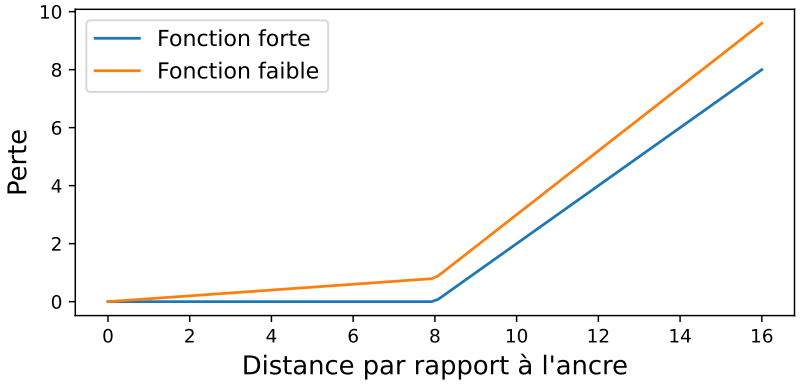


FIGURE 4.7: **Fonctions de perte *forte* et *faible***. Dans un cas où le rayon de l'hypersphère vaut $r = 8$, la fonction de perte *forte* pénalise uniquement la distance d'une instance par rapport à l'hypersphère associée, tandis que la fonction de perte *faible* pénalise aussi la distance entre l'instance et l'ancre, mais avec un coefficient plus faible que la distance par rapport à l'hypersphère.

Les éléments de configuration du modèle que nous souhaitons faire varier sont les suivants :

- **La fonction de perte** (2 possibilités) :
  - $\mathcal{L}_{\text{out}}(\boldsymbol{X}, y)$ (équation 4.1), la fonction de perte *forte*,
  - $\mathcal{L}_{\text{in/out}}(\boldsymbol{X}, y)$ (équation 4.8), la fonction de perte *faible*.
- **La fonction d'activation de la représentation** (3 possibilités) :
  - ReLU,
  - LeakyReLU,
  - aucune activation.
- **La pénalisation de l'écart-type** (5 possibilités) :
  - $\mathcal{L}_{\text{exp}}(\alpha = 2, \tau = 0.1, \epsilon = 1e-3)$,
  - $\mathcal{L}_{\text{exp}}(\alpha = 2, \tau = 0.2, \epsilon = 1e-3)$,

  - $\mathcal{L}_{\text{lin}}(\gamma = 2, \tau = 0.1)$,
  - $\mathcal{L}_{\text{lin}}(\gamma = 2, \tau = 0.2)$,
  - aucune pénalisation sur l'écart-type.
- **La pénalisation de la corrélation** (2 possibilités) :
  - $\mathcal{L}_{\text{correlation}}$ (équation 4.7),
  - aucune pénalisation sur la corrélation.

Au total, cela correspond à $2 \times 3 \times 5 \times 2 = 60$ configurations différentes à tester. Sachant que chaque expérience est répétée 5 fois, cela donne un total de 300 entraînements à effectuer. À raison d'une exécution moyenne de 1 heure par entraînement, cela représente donc 300 heures de calculs, soit environ 2 jours de calculs en utilisant 6 cartes graphiques pour effectuer des entraînements simultanément [1].

Pour les expériences de configuration, nous fixons les hyperparamètres de DIFAIR à $\mathcal{N} = 20$, $\alpha = 2$ et $r = 0.4 \cdot d_{\mathcal{A}} \approx 5.06$, avec $d_{\mathcal{A}}$ la distance entre deux ancres. Le taux d'apprentissage utilisé est de 0.1.

La figure 4.8 montre les performances en ECS, AUROC et PH obtenues avec les deux fonctions de perte. Ces résultats montrent que la fonction de perte faible obtient de meilleures performances sur tous les critères : pénaliser les instances pour qu'elles se rapprochent de leur ancre permet effectivement d'avoir plus de données représentées dans les hypersphères par rapport à la perte forte. Cela permet aussi d'obtenir une meilleure séparation des données puisque l'ECS est plus élevée, bien que l'amélioration paraisse négligeable (la plage de valeurs de l'ECS allant de 92.5 à 93.1 pour les deux fonctions confondues). Les améliorations en AUROC et en PH sont plus significatives. Ces résultats sont différents de l'observation faite par Vaze et al. (2022) qui stipulaient que l'exactitude était corrélée à l'AUROC. En effet, dans notre situation, l'AUROC varie grandement en fonction de la configuration, alors que l'ECS reste stable. Le coefficient de corrélation de Pearson entre les deux métriques vaut 0.52 en considérant l'ensemble des points, 0.16 et -0.2 en considérant indépendamment les fonctions de perte faible et forte, respectivement. Cependant, Vaze et al. (2022) ont réalisé cette observation sur des modèles entraînés avec entropie croisée, il est donc possible que cette caractéristique ne s'étende pas à d'autres approches. Puisque la fonction de perte faible obtient de meilleurs résultats, les configurations étudiées sont filtrées pour ne conserver que celles utilisant cette fonction de perte.

1. Nous avons eu accès au Centre de Calcul de l'Université de Strasbourg (CCUS) pour entraîner nos modèles.

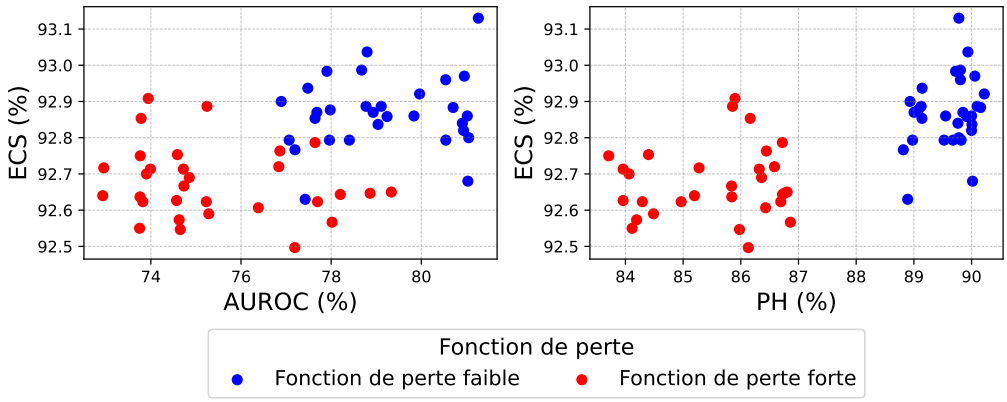


FIGURE 4.8: **Performances en fonction de la fonction de perte**. Les performances sont mesurées avec l'ECS, l'AUROC et le PH.

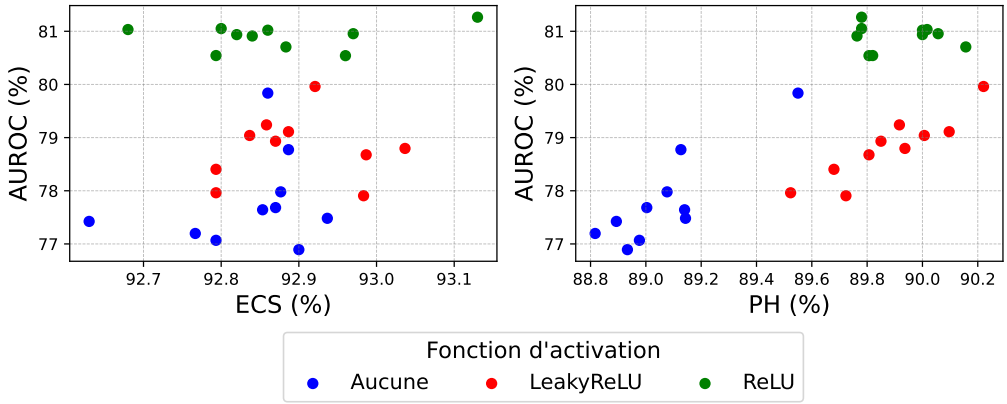


FIGURE 4.9: **Performances en fonction de la fonction d'activation**. Les expériences ont été filtrées pour conserver uniquement celles utilisant la fonction de perte faible. Les performances sont mesurées avec l'ECS, l'AUROC et le PH.

Les performances obtenues en fonction de la fonction d'activation appliquée à la représentation extraite par DIFAIR sont présentées dans la figure 4.9. La fonction d'activation ReLU donne lieu aux meilleures performances en AUROC et en PH, là où l'ECS reste stable peu importe la fonction de perte utilisée. La fonction d'activation LeakyReLU permet tout de même d'obtenir plus de données représentées dans les hypersphères qu'en n'utilisant pas de fonction d'activation, mais ses performances en AUROC sont inférieures à ReLU. Nous supposons que la fonction d'activation ReLU est plus adaptée à l'utilisation de DIFAIR, car des dimensions ayant des va-

leurs négatives pénalisent l'évaluation de la distance par rapport aux ancres. Par exemple, une donnée connue pour laquelle les attributs associés à sa classe sont tous activés et les autres attributs sont à 0 serait proche de son ancre, mais la présence de valeurs négatives sur les autres dimensions, qui pourrait représenter l'absence des caractéristiques d'autres classes, éloignerait l'instance de son ancre. Par conséquent, la fonction d'activation ReLU est retenue pour la suite des expériences et les résultats sont filtrés pour ne conserver que les configurations utilisant cette fonction d'activation.

La fonction de perte et la fonction d'activation étant choisies, il reste à déterminer quelles pénalisations appliquer aux poids de la dernière couche afin d'éviter leur convergence ou leur corrélation. Les figures 4.10 et 4.11 montrent les performances obtenues en utilisant respectivement la pénalisation de l'écart-type et de la corrélation. Lorsque ni l'écart-type ni la corrélation ne sont pénalisés (point jaune non ciblé dans la figure 4.10), la valeur de CONV (équation 4.4) est proche de 0, ce qui signifie que les poids de la dernière couche ont probablement convergé et la corrélation totale des filtres, mesurée par $\mathcal{L}_{\text{correlation}}$ (équation 4.7), est la plus élevée. Les points signalés en rouge correspondent à la configuration utilisant uniquement la pénalisation $\mathcal{L}_{\text{correlation}}$. Même si aucune pénalisation de l'écart-type n'est appliquée, la valeur de CONV est autour de 0.1 (en sachant qu'il s'agit d'une valeur moyenne de 5 exécutions de cette configuration), ce qui démontre qu'il n'y a pas eu de convergence, car la métrique CONV vaut environ 0.035 à l'initialisation des poids. Puisque $\mathcal{L}_{\text{correlation}}$ permet à la fois d'empêcher la convergence et la corrélation, nous choisissons d'utiliser exclusivement cette pénalisation. Cela permet aussi d'éviter d'avoir à déterminer les hyperparamètres supplémentaires liés aux fonctions de pénalisation de l'écart-type, car celles-ci nécessitent de choisir une valeur cible pour l'écart-type moyen des poids, or c'est une valeur qui semble difficile à déterminer a priori.

Sur les figures 4.10 et 4.11, le point encadré et indiqué par une flèche rouge correspond donc à la configuration retenue pour la suite des expériences, à savoir la configuration de DIFAIR avec la fonction de perte faible, la fonction d'activation ReLU et la pénalisation de la corrélation, mais pas de l'écart-type. Cette configuration est utilisée pour la recherche des hyperparamètres.

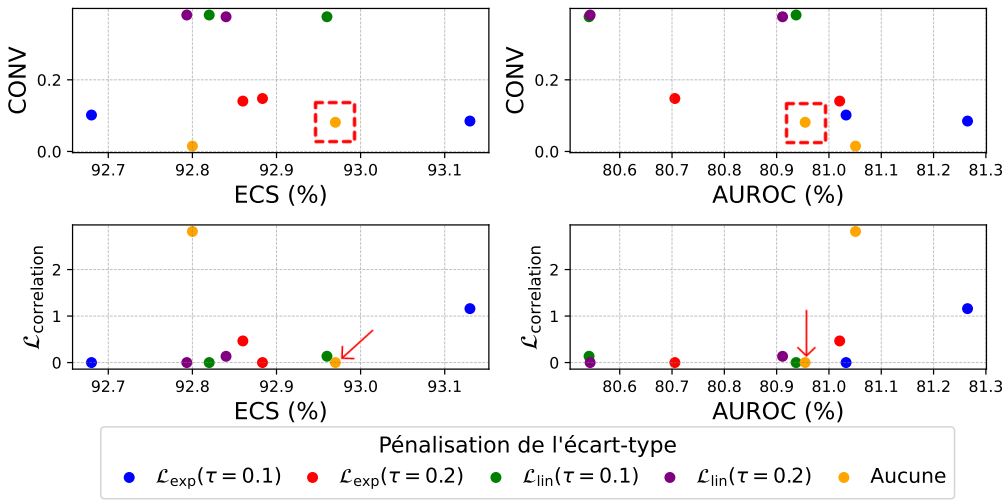


FIGURE 4.10: **Performances en fonction de la pénalisation de l'écart-type**. Les performances sont mesurées avec CONV (équation 4.4), $\mathcal{L}_{\text{correlation}}$ (équation 4.7), l'AUROC et l'ECS. Le point encadré et ciblé en rouge correspond à la configuration retenue pour la suite des expériences.

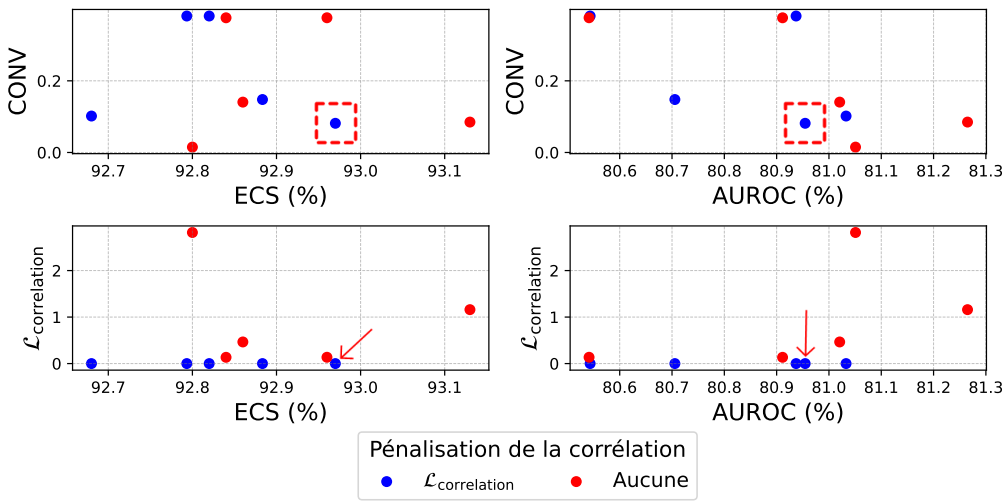


FIGURE 4.11: **Performances en fonction de la pénalisation de la corrélation**. Les performances sont mesurées avec CONV (équation 4.4), $\mathcal{L}_{\text{correlation}}$ (équation 4.7), l'AUROC et l'ECS. Le point encadré et ciblé en rouge correspond à la configuration retenue pour la suite des expériences.

4.3.2 Recherche des hyperparamètres

Après avoir défini la configuration optimale pour DIFAIR, nous souhaitons évaluer l'influence des hyperparamètres sur les performances du modèle et rechercher leurs valeurs optimales. Nous effectuons la recherche d'hyperparamètres sur les jeux de données CIFAR-10 et TinyImageNet, en utilisant la configuration de DIFAIR décrite ci-dessus.

CIFAR-10

Pour évaluer les performances sur CIFAR-10, les hyperparamètres que nous souhaitons faire varier sont :

- **Le taux d'apprentissage** (4 possibilités) : 0.1, 0.2, 0.3, 0.4.
- **Le nombre d'attributs** $\mathcal{N}$ (3 possibilités) : 5, 10, 20.
- **Le paramètre** α (3 possibilités) : 2, 5, 10.
- **Le rayon** r **de l'hypersphère** (5 possibilités) : puisque la distance entre les ancres dépend de $\mathcal{N}$ et α, le rayon r est défini comme une fraction de la distance entre deux ancres. Les valeurs testées sont $0.2 \cdot d_{\mathcal{A}}$, $0.3 \cdot d_{\mathcal{A}}$, $0.4 \cdot d_{\mathcal{A}}$, $0.5 \cdot d_{\mathcal{A}}$, $0.6 \cdot d_{\mathcal{A}}$ avec $d_{\mathcal{A}}$ la distance entre deux ancres.

Cela correspond à $4 \times 3 \times 3 \times 5 = 180$ combinaisons à tester. Ici aussi, chaque expérience est répétée 5 fois pour améliorer la significativité des résultats, ce qui donne un total de 900 entraînements à effectuer. À raison de 1 heure par entraînement, cela représente 900 heures de calculs, soit environ 6 jours de calculs en utilisant 6 cartes graphiques pour effectuer des entraînements simultanément.

La figure 4.12 montre les performances mesurées en fonction de la fraction de la distance entre deux ancres utilisée pour calculer r, le rayon de l'hypersphère allouée autour de chaque ancre. Une observation logique est que, plus le rayon de l'hypersphère est grand, plus il y a de données représentées dans les hypersphères (mesurées par le PH). Ce qui est intéressant est de remarquer que l'AUROC augmente avec le rayon de l'hypersphère. De plus, même avec un rayon $r = 0.6 \cdot d_{\mathcal{A}}$, avec $d_{\mathcal{A}}$ la distance entre deux ancres, l'AUROC augmente par rapport à $r = 0.5 \cdot d_{\mathcal{A}}$. Or, si $r = 0.6 \cdot d_{\mathcal{A}}$, cela signifie que les hypersphères associées à chaque classe se chevauchent, ce qui parait contradictoire avec l'idée de DIFAIR qui est de séparer correctement les classes connues. Pourtant, l'ECS ne diminue que très légèrement en utilisant une fraction de la distance à 0.6. Nous avons dans un premier temps émis l'hypothèse qu'une hypersphère plus grande augmenterait la

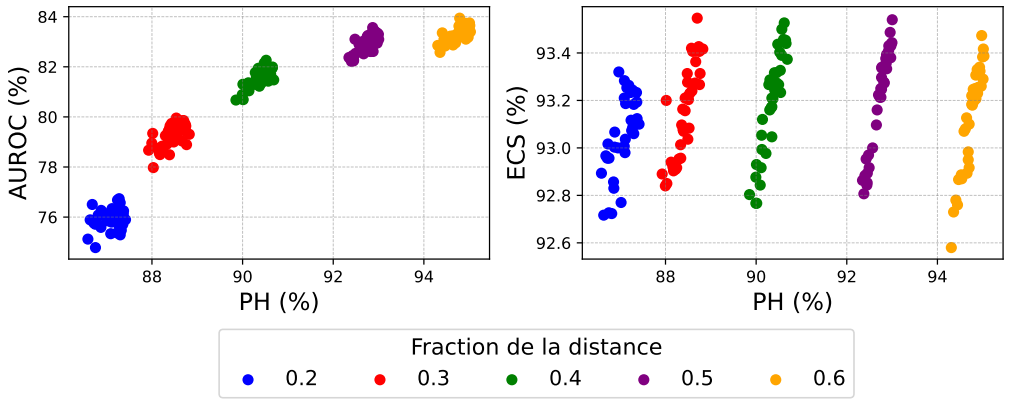


FIGURE 4.12: **Performances en fonction du rayon** r. La valeur recherchée correspond à la fraction de la distance entre deux ancres. Les performances sont mesurées avec l'ECS, l'AUROC et le PH.

flexibilité dans la représentation : cela permettrait ainsi d'extraire des attributs exprimant une meilleure sémantique pour décrire les classes connues. En effet, avec une hypersphère plus grande, l'activation d'attributs d'autres classes est moins pénalisante ; et l'extraction de ces attributs sémantiques pour décrire les classes connues permettrait d'obtenir des attributs moins activés sur les classes inconnues, et ainsi d'améliorer l'AUROC. Comme nous le verrons par la suite (section 5.2.3), cette hypothèse semble en réalité incorrecte. De plus, le fait que la fonction de perte faible soit utilisée permet de tout de même représenter les instances à proximité de leur ancre. Des hypersphères se chevauchant sont donc moins problématiques que si la fonction de perte forte était utilisée et que les instances pouvaient être représentées librement dans les hypersphères. Pour ces raisons, la valeur de $r = 0.6 \cdot d_{\mathcal{A}}$ est retenue pour la suite des expériences.

Comme précédemment, les expériences restantes sont filtrées pour conserver uniquement celles avec la fraction de la distance valant 0.6. Les performances en fonction du taux d'apprentissage sont présentées dans la figure 4.13. Ces graphiques montrent que les taux d'apprentissage 0.3 et 0.4 permettent d'obtenir les meilleurs résultats en ECS, en AUROC et en PH. Les performances sont similaires pour ces deux taux d'apprentissage, c'est pourquoi nous étudions les performances des autres hyperparamètres pour ces deux valeurs. Notons que les améliorations de chacune des métriques sont minimes, avec des plages de valeurs séparées de moins de 1% pour l'ECS et le PH et de 1.5% pour l'AUROC.

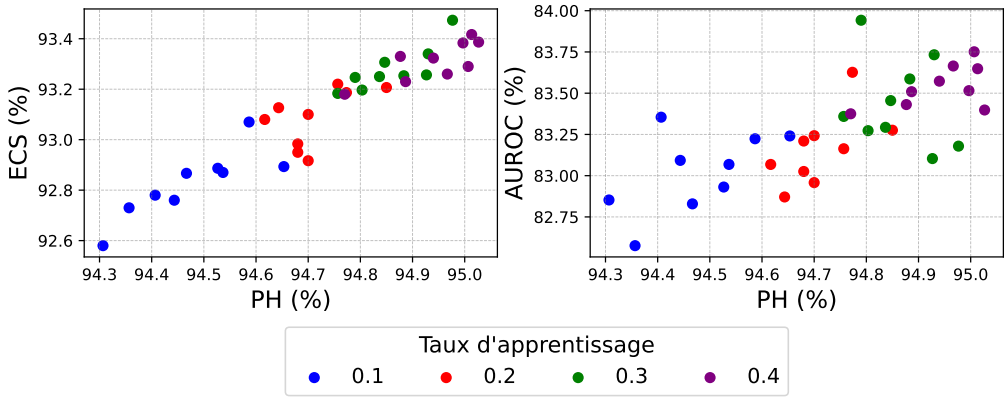


Figure 4.13: **Performances en fonction du taux d'apprentissage**. Les performances sont mesurées avec l'ECS, l'AUROC et le PH.

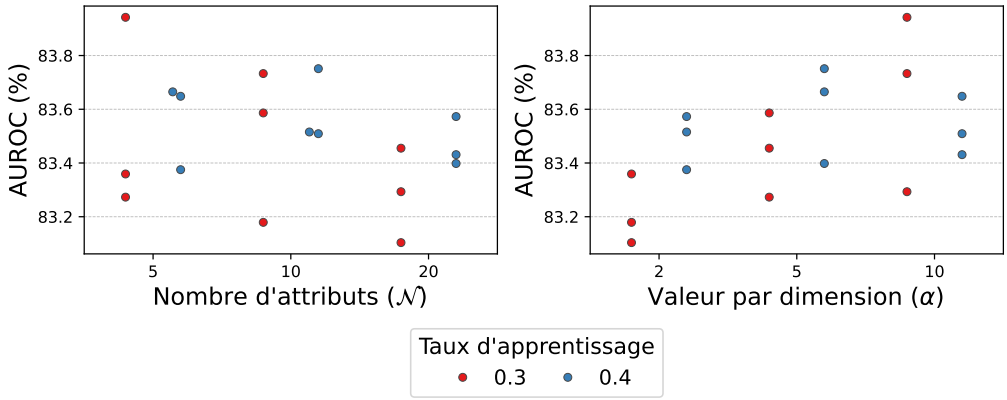


Figure 4.14: **Performances en fonction du nombre d'attributs $\mathcal{N}$ et du paramètre α**. Les performances sont mesurées avec l'AUROC.

La figure 4.14 montre les résultats obtenus en fonction du nombre d'attributs $\mathcal{N}$ et du paramètre α, configurant la valeur d'activation cible sur chaque dimension. Afin de visualiser les différences de distribution de performances selon ces paramètres pour les différents taux d'apprentissage, nous utilisons un graphique en essaim (*swarmplot* en anglais) afin de représenter pour chaque paramètre et chaque taux d'apprentissage les valeurs obtenues. La seule métrique utilisée pour évaluer les performances des deux hyperparamètres est l'AUROC, car l'ECS varie peu sur ces configurations, et le PH a moins d'importance que l'AUROC, qui, elle, indique la capacité à détecter les données inconnues, l'objectif initial de DIFAIR. Il est difficile d'observer une quelconque influence des paramètres à partir de ce

graphique. Ce que nous pouvons en déduire est que la meilleure AUROC a été obtenue avec les paramètres $\mathcal{N} = 5$ et $\alpha = 10$ pour le taux d'apprentissage 0.3. Ces paramètres sont ceux que nous choisissons pour le reste des expériences. Notons que la valeur de α est identique à celle utilisée par Miller et al. (2021) dans leur approche CAC. Pour le taux d'apprentissage 0.4, les meilleures performances sont obtenues avec $\mathcal{N} = 10$ et $\alpha = 5$.

Pour mieux comprendre l'influence de ces paramètres sur les performances, calculons la distance $d_{\mathcal{A}}$ entre les ancres. La figure 4.15 présente l'AUROC obtenue en fonction de cette distance, pour les deux taux d'apprentissage. Deux courbes polynomiales de degré 5 ont été ajustées pour approximer les données. Ces courbes semblent montrer que l'AUROC augmente d'abord conjointement à la distance entre les ancres puis diminue après un certain point. Selon le taux d'apprentissage utilisé, le pic d'AUROC est atteint pour des distances entre les ancres différentes. Lorsque le taux d'apprentissage est plus faible, le pic est atteint pour une distance plus large.

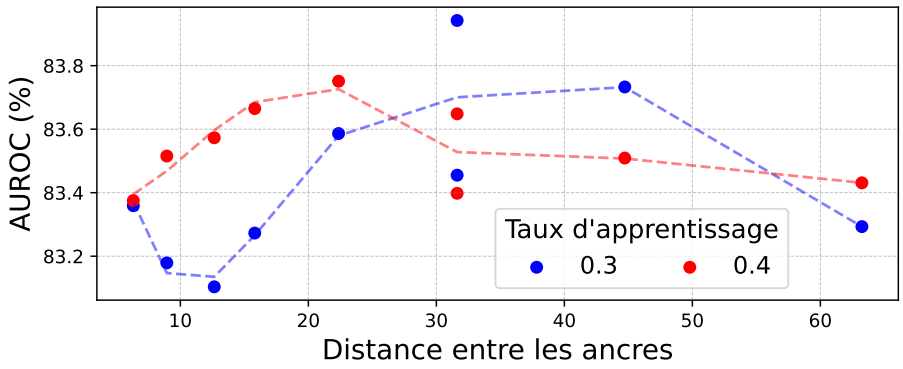


FIGURE 4.15: **Évolution de l'AUROC en fonction de la distance entre les ancres**. Les lignes bleu et rouge représentent une approximation de la courbe suivie par les données à l'aide d'un polynôme de degré 5 pour les taux d'apprentissage de 0.3 et 0.4 respectivement.

Tous les modèles entraînés pour rechercher les hyperparamètres ont une corrélation des poids de la dernière couche très faible : cet objectif est facilement optimisé par le modèle. De même, sur l'ensemble des expériences, les poids ne convergent pas. Cela a été vérifié à l'aide d'un script indiquant que l'écart-type des poids final n'est jamais supérieur à l'écart-type des poids initial, ce qui signifie que la configuration choisie et la pénalisation de la corrélation empêchent efficacement la convergence des poids.

Pour un jeu de données comme CIFAR-10, avec six classes inconnues, les meilleurs hyperparamètres déterminés pour DIFAIR sont donc : $\mathcal{N} = 5$, $\alpha = 10$, $r = 0.6 \cdot d_{\mathcal{A}}$, avec $d_{\mathcal{A}}$ la distance entre deux ancres. Le taux d'apprentissage choisi est de 0.3. Ce sont principalement le taux d'apprentissage et la taille des hypersphères qui influent sur les performances du modèle. Les hyperparamètres $\mathcal{N}$ et α, déterminant la distance entre les ancres, semblent indiquer qu'il faut trouver un équilibre entre des ancres trop proches et trop éloignées. En effet, des ancres trop proches ne permettent probablement pas de bien séparer les classes, tandis qu'en utilisant des ancres trop éloignées, les performances semblent se dégrader.

TinyImageNet

Une recherche d'hyperparamètres a aussi été effectuée sur la séparation 0 de TinyImageNet. Il s'agit d'un jeu de données plus difficile que CIFAR-10, car il contient plus de classes et possède moins d'images par classe pour l'entraînement. L'objectif est de vérifier si, dans une situation où il y a plus de classes connues (vingt contre six pour CIFAR-10), les ancres devraient être configurées différemment, par exemple en les éloignant davantage les unes des autres et en attribuant plus de dimensions à chaque classe. Des plages de valeurs d'hyperparamètres plus restreintes sont testées, au vu des résultats obtenus sur CIFAR-10.

Pour évaluer les performances sur TinyImageNet, les hyperparamètres que nous souhaitons faire varier sont :

— **Le taux d'apprentissage** (3 possibilités) : 0.3, 0.4, 0.5.

— **Le nombre d'attributs** $\mathcal{N}$ (3 possibilités) : 5, 10, 20.

— **Le paramètre** α (3 possibilités) : 2, 5, 10.

— **Le rayon** r **de l'hypersphère** (3 possibilités) : $0.4 \cdot d_{\mathcal{A}}$, $0.5 \cdot d_{\mathcal{A}}$, $0.6 \cdot d_{\mathcal{A}}$ avec $d_{\mathcal{A}}$ la distance entre deux ancres.

Cela correspond à $3 \times 3 \times 3 \times 3 = 81$ combinaisons à tester. Chaque expérience est répétée 5 fois pour améliorer la significativité des résultats, ce qui donne un total de 405 entraînements à effectuer. À raison de 1 heure par entraînement en moyenne, cela représente 405 heures de calculs, soit presque 3 jours de calculs sur 6 cartes graphiques.

Les performances obtenues en fonction de la fraction de la distance entre deux ancres utilisée pour calculer r sont présentées dans la figure 4.16. Comme pour CIFAR-10, la valeur 0.6 permet d'obtenir la

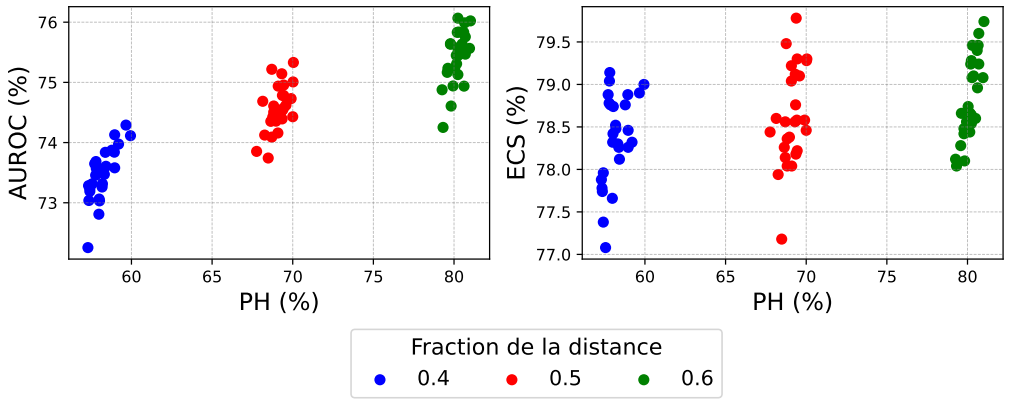


FIGURE 4.16: **Performances en fonction du rayon r sur TinyImageNet**. La valeur recherchée correspond à la fraction de la distance entre deux ancres. Les performances sont mesurées avec l'ECS, l'AUROC et le PH.

meilleure AUROC, le meilleur PH ainsi qu'une ECS légèrement plus élevée qu'avec une fraction de la distance valant 0.4. Nous retenons donc la valeur $r = 0.6 \cdot d_{\mathcal{A}}$ pour la suite des expériences.

Les résultats obtenus en fonction du taux d'apprentissage sont présentés dans la figure 4.17. Globalement, les performances sont meilleures pour le taux d'apprentissage 0.5 : ce taux plus élevé permet d'obtenir une AUROC et une ECS supérieures à la plupart des autres expériences. La variabilité des différentes métriques est supérieure à celle observée sur CIFAR-10, avec une ECS variant de 1.5% et une AUROC de 2.5% entre les différentes configurations. Nous choisissons d'observer les résultats en fonction des autres hyperparamètres pour les taux d'apprentissage 0.4 et 0.5.

La figure 4.18 montre les performances obtenues en fonction du nombre d'attributs $\mathcal{N}$ et du paramètre α. Seule l'AUROC est observée pour ces différentes configurations, l'ECS étant stable. La meilleure AUROC est obtenue avec $\mathcal{N} = 5$ et $\alpha = 5$ pour le taux d'apprentissage 0.5. Cependant, l'observation des représentations apprises par DIFAIR sur TinyImageNet a montré que certaines classes exhibaient des valeurs d'attributs peu élevées en moyenne indiquant que le modèle ne parvenait pas à les représenter correctement autour des ancres. Sachant que α permet d'augmenter la distance entre les ancres, nous choisissons de fixer $\alpha = 10$ sous l'hypothèse que cette valeur permettrait d'obtenir de meilleures performances en OSR, comme c'est le cas pour CIFAR-10, si nous parvenions à mieux optimiser le modèle malgré la difficulté du jeu de données.

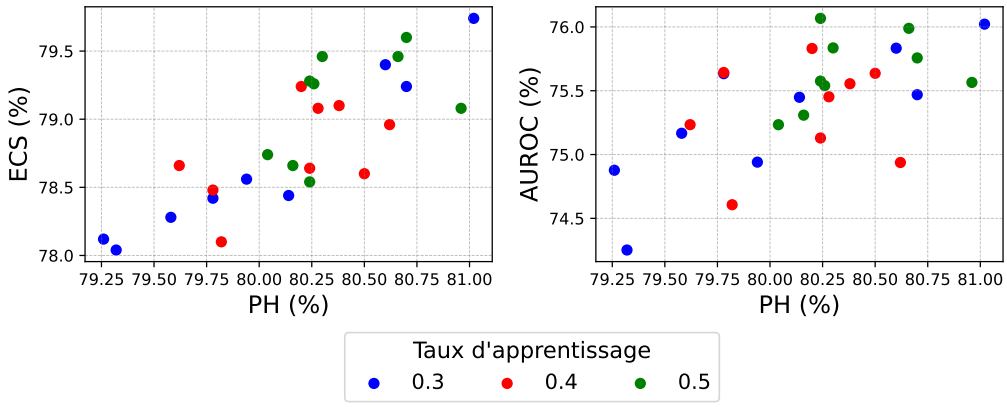


Figure 4.17: **Performances en fonction du taux d'apprentissage sur TinyImageNet**. Les performances sont mesurées avec l'ECS, l'AUROC et le PH.

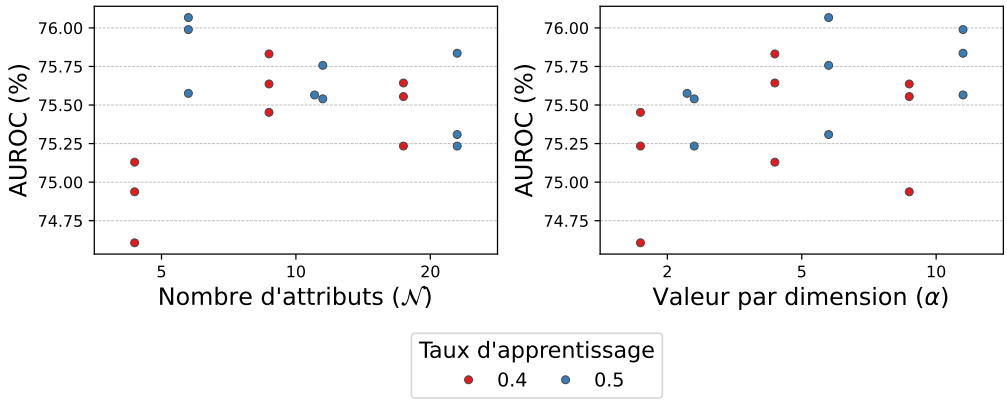


Figure 4.18: **Performances en fonction du nombre d'attributs $\mathcal{N}$ et du paramètre α sur TinyImageNet**. Les performances sont mesurées avec l'AUROC.

Afin d'observer l'influence de la distance entre les ancres $d_{\mathcal{A}}$ sur les performances, la figure 4.19 montre l'AUROC obtenue en fonction de cette distance, pour les deux taux d'apprentissage considérés. Les courbes d'approximation ne semblent pas exhiber un comportement similaire à CIFAR-10. En effet, les performances sont plus élevées avec un taux d'apprentissage à 0.5, mais les performances semblent plus stables en fonction de la distance. Cela peut-être dû au fait que les modèles sont plus difficilement entraînés sur TinyImageNet (la perte diminue moins que sur CIFAR-10), et expliquerait le fait qu'ils convergent vers une performance limite.

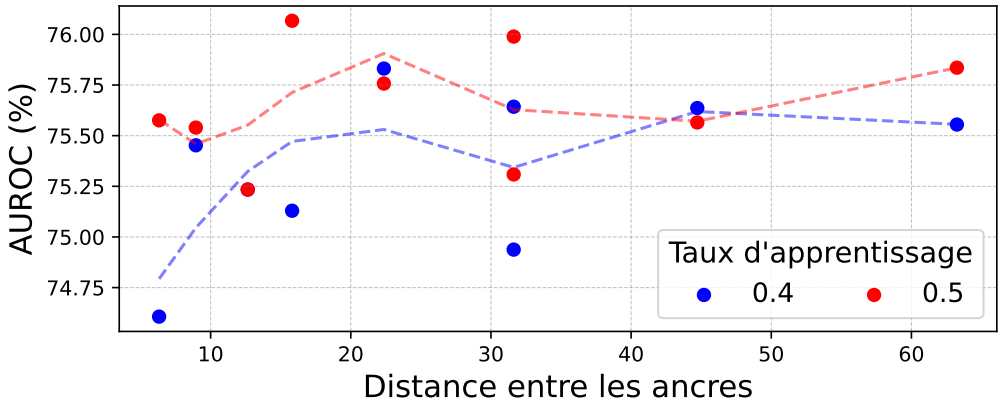


FIGURE 4.19: **Évolution de l'AUROC en fonction de la distance entre les ancres sur TinyImageNet**. Les lignes bleu et rouge représentent une approximation de la courbe suivie par les données à l'aide d'un polynôme de degré 5 pour les taux d'apprentissage de 0.3 et 0.4 respectivement.

Mis à part le taux d'apprentissage, les autres hyperparamètres ne semblent pas avoir besoin d'être modifiés sur TinyImageNet, qui contient plus de classes que CIFAR-10. Dans la version actuelle de DIFAIR, changer le nombre d'attributs pouvant décrire une classe ne semble donc pas avoir de grande influence sur les performances, alors qu'on pourrait penser qu'un plus grand nombre d'attributs serait nécessaire pour mieux distinguer un plus grand nombre de classes. Les meilleurs hyperparamètres déterminés pour DIFAIR sur TinyImageNet sont donc : $\mathcal{N} = 5$, $\alpha = 10$, $r = 0.6 \cdot d_{\mathcal{A}}$, avec $d_{\mathcal{A}}$ la distance entre deux ancres et un taux d'apprentissage de 0.5. De plus, comme pour CIFAR-10, la corrélation des filtres est très faible et les poids ne convergent pas dans l'ensemble des expériences réalisées.

La recherche par grille de la configuration de DIFAIR, ainsi que des hyperparamètres en fonction du nombre de classes connues, a permis de déterminer une configuration et des hyperparamètres optimaux pour obtenir les meilleurs résultats en OSR sur les deux tâches observées. Par la suite, nous ferons référence aux modèles utilisant cette configuration et ces hyperparamètres sous le nom de DIFAIR v1. Nous proposons maintenant d'examiner les capacités de DIFAIR v1 sur le benchmark d'OSR proposé par Neal et al. (2018).

Chapitre 5

Application de DIFAIR en OSR

Résumé

Ce chapitre présente l'évaluation d'une version améliorée de DIFAIR sur le benchmark d'OSR et réalise une comparaison avec les autres méthodes de la littérature. Plus précisément, l'approche que nous évaluons est nommée DIFAIR v1, dont la configuration et les hyperparamètres ont été sélectionnés via une recherche par grille. À nouveau, nous observons que l'analyse approfondie des représentations extraites par DIFAIR joue un rôle crucial dans l'appréciation des avantages et des limites de l'approche.

Sommaire

5.1 Application de DIFAIR v1 au benchmark d'OSR

Après avoir déterminé les meilleures configurations et hyperparamètres pour DIFAIR v1 dans la section 4.3, nous évaluons l'approche sur l'intégralité du benchmark d'OSR proposé par Neal et al. (2018). Les résultats obtenus sont comparés à ceux de DIFAIR v0 (section 4.1), à CAC (Miller et al., 2021) ainsi qu'à plusieurs approches de pointe en OSR, présentées dans la section 2.2.2.

5.1.1 Configuration de DIFAIR v1

La recherche de la meilleure configuration de DIFAIR a révélé que la configuration utilisée pour les premières expériences n'était pas optimale pour obtenir de bonnes performances en OSR. Contrairement à la version DIFAIR v0, présentée dans la section 4.1, la fonction de perte choisie pour entraîner DIFAIR v1 est la fonction de perte faible $\mathcal{L}_{\text{in/out}}$:

$$\mathcal{L}_{\text{in/out}}(\boldsymbol{X}, y) = \mathcal{L}_{\text{in}}(\boldsymbol{X}, y) + \mathcal{L}_{\text{out}}(\boldsymbol{X}, y),$$

avec :

$$\begin{aligned}\mathcal{L}_{\text{in}}(\boldsymbol{X}, y) &= 0.1 \cdot \text{Eucl}(\boldsymbol{z}, \mathcal{A}^{(y)}),\\ \mathcal{L}_{\text{out}}(\boldsymbol{X}, y) &= \max(\text{Eucl}(\boldsymbol{z}, \mathcal{A}^{(y)}) - r, 0),\end{aligned}$$

où $\text{Eucl}(\boldsymbol{z}, \mathcal{A}^{(y)})$ est la distance euclidienne entre la représentation $\boldsymbol{z}$ de l'instance $\boldsymbol{X}$ et l'ancre $\mathcal{A}^{(y)}$ associée à la classe y, et r est le rayon de l'hypersphère autour de chaque ancre. Cette fonction ne permet plus au modèle de représenter librement les instances dans les hypersphères autour des ancres : la fonction de perte $\mathcal{L}_{\text{in}}$ contraint les instances pour être rapprochées de leur ancre, mais cette contrainte est moins forte que celle de représenter les instances dans l'hypersphère. La fonction d'activation ReLU est appliquée à la représentation, et un terme pénalisant la corrélation des filtres de la dernière couche est ajouté, $\mathcal{L}_{\text{correlation}}$ (équation 4.7). La perte totale $\mathcal{L}_{\text{DIFAIR}}$ est donc :

$$\mathcal{L}_{\text{DIFAIR}} = \mathcal{L}_{\text{in/out}} + \lambda \mathcal{L}_{\text{correlation}},$$

avec λ un scalaire permettant équilibrer l'importance des deux termes de la perte. Cependant, l'optimisation de la corrélation des filtres étant facilement réalisée par le modèle, λ est fixé à 1 pour toutes les expériences.

Pour toutes les tâches du benchmark, les hyperparamètres ont pour valeur $\mathcal{N} = 5$, $\alpha = 10$, $r = 0.6 \cdot d_{\mathcal{A}}$, avec $d_{\mathcal{A}}$ la distance entre deux ancres, et un taux d'apprentissage de 0.3 est utilisé pour tous les jeux de données, excepté pour TinyImageNet où le taux utilisé est de 0.5 (les taux d'apprentissage initiaux étaient de 0.1 et 0.01 respectivement). Les mêmes taux d'apprentissage sont utilisés pour l'entraînement du modèle *baseline* utilisant l'entropie croisée.

5.1.2 Expériences

L'architecture (à l'exception de la fonction d'activation de la représentation) et le protocole d'entraînement utilisés pour DIFAIR v1 sont identiques à ceux utilisés pour l'entraînement de DIFAIR v0. Ce protocole, proposé par Vaze et al. (2022), est décrit dans la section 4.1 (page 106).

Nous souhaitons comparer les performances de DIFAIR v1 à plusieurs méthodes d'OSR. Le modèle *baseline* est ré-entraîné en utilisant les taux d'apprentissage nouvellement déterminés : 0.3 pour tous les jeux de données sauf TinyImageNet, où le taux d'apprentissage est de 0.5. Les scores d'OSR évalués sont les mêmes que ceux décrits dans la section 4.1.1 : le *Maximum Softmax Probability* (MSP) et le *Maximum Logit Score* (MLS) (Vaze et al., 2022).

La méthode CAC (Miller et al., 2021), proche de l'approche que nous proposons, est ré-entraînée en utilisant le protocole amélioré de Vaze et al. (2022) et les mêmes séparations de classes que nous utilisons. L'objectif est de comparer équitablement les résultats de CAC avec DIFAIR. Nous n'avons pas effectué de recherche complète d'hyperparamètres pour CAC, mais étant donné que le taux d'apprentissage avait un impact important sur les performances de DIFAIR, nous avons entraîné CAC avec les taux d'apprentissage 0.1 et 0.3. Les résultats sont reportés pour le score d'OSR initial de CAC et en utilisant le *Maximum Output Score* (MOS) que nous proposons.

Pour situer DIFAIR par rapport à l'état de l'art en OSR, les résultats d'approches de pointe en OSR sont reportés. Ces approches sont détaillées dans la section 2.2.2 du chapitre 2. Nous reportons les résultats pour les méthodes :

— DCHS (Cevikalp et al., 2023), une méthode partageant des similarités avec DIFAIR, où les positions des centres représentant chaque classe dans l'espace de représentation sont appris. Des données supplémentaires provenant du jeu de données 80 Million Tiny Images (Torralba

et al., 2008) sont utilisées pendant l'entraînement pour représenter les classes inconnues : cela risque d'exposer le modèle à des classes inconnues qui devraient être détectées au moment du test.

— (ARPL+CS)+ (Vaze et al., 2022), est une version améliorée de la méthode ARPL+CS (Chen et al., 2021) utilisant le protocole d'entraînement de Vaze et al. (2022). L'approche initiale de Chen et al. (2021) utilise des prototypes appris pour représenter des points *réciproques* aux données connues, c'est-à-dire que celles-ci doivent en être éloignées. Des données supplémentaires identifiées comme inconnues sont générées par un GAN pour aider à l'entraînement. La version améliorée de Vaze et al. (2022) augmente les performances de 2.5% en moyenne sur les jeux de données d'images naturelles.

— ConOSR (Xu et al., 2023), utilise du *contrastive learning* supervisé et des exemples inconnus générés à l'aide de la méthode d'augmentation de données *mixup* (Zhang et al., 2018) pour pré-entraîner un encodeur de caractéristiques qui sépare les classes connues et inconnues dans sa représentation, puis un classifieur est entraîné sur cette représentation.

Enfin, DIFAIR v1 est aussi comparée aux résultats obtenus avec DIFAIR v0 (section 4.1). Les résultats sont reportés en utilisant le score d'OSR de DIFAIR basé sur la distance aux ancres et avec le MOS.

5.1.3 Résultats sur le benchmark d'OSR

Les résultats de l'évaluation des approches sur le benchmark d'OSR sont présentés dans le tableau 5.1. Analysons les performances de chacun des modèles.

Baseline. Par rapport aux résultats présentés initialement dans le tableau 4.1, le changement de taux d'apprentissage n'a pas eu d'influence significative sur les performances, excepté sur le jeu de données TinyImageNet où l'AUROC a augmenté de 7%. Le score MLS améliore effectivement les performances par rapport au MSP, notamment sur les jeux de données d'images naturelles CIFAR-10, CIFAR+10 et CIFAR+50 avec une augmentation minimale de 2% de l'AUROC. L'amélioration est plus faible sur TinyImageNet, avec seulement 0.6%. De plus, les performances de notre réseau *baseline* restent inférieures à celles obtenues par Vaze et al. (2022), bien qu'il s'agisse théoriquement de la même expérience, mais utilisant deux bibliothèques d'apprentissage profond différentes. Notre version du réseau *baseline* surpasse cependant les différentes versions de DIFAIR et de CAC que nous avons entraînées.

Méthode	MNIST	SVHN	CIFAR10	CIFAR+10	CIFAR+50	TinyImageNet
MSP	99.32	95.38	89.78	89.60	90.13	73.24
MLS	99.55	96.66	92.55	91.10	92.53	73.87
MLS (Vaze et al., 2022)	99.3	97.1	93.6	97.9	96.5	83.0
CAC (lr=0.1)	99.11	96.58	85.85	86.03	87.28	65.20
CAC (MOS) (lr=0.1)	97.54	96.17	86.38	83.08	84.97	73.23
CAC (lr=0.3)	99.11	96.61	86.43	85.01	86.65	67.65
CAC (MOS) (lr=0.3)	97.92	96.40	86.87	82.08	84.86	76.42
DCHS (Cevikalp et al., 2023)	96.6	94.5	**94.7**	**99.2**	**98.5**	**83.8**
(ARPL+CS)+ (Vaze et al., 2022)	99.2	96.8	93.9	98.1	96.7	82.5
ConOSR (Xu et al., 2023)	**99.7**	**99.1**	94.2	98.1	97.3	80.9
DIFAIR v0	98.9	93.5	77.3	77.4	77.4	65.2
DIFAIR v0 (MOS)	99.5	94.8	86.5	86.8	86.1	69.3
DIFAIR v1	98.35	95.45	85.04	83.86	84.52	73.38
DIFAIR v1 (MOS)	97.81	93.13	82.98	79.74	80.50	74.89

TABLEAU 5.1: **Résultats sur le benchmark d'OSR**. Scores d'AUROC pour la tâche de détection des classes inconnues, moyenné sur cinq séparations de classes connues/inconnues. Les résultats non cités proviennent de nos expériences. Les scores les plus élevés sont indiqués en gras.

CAC. La version de CAC entraînée avec le taux d'apprentissage de 0.3 obtient des performances supérieures à la version entraînée avec le taux de 0.1, sauf sur les jeux de données CIFAR+10 et CIFAR+50. En évaluant l'AUROC avec le MOS au lieu du score d'OSR proposé par Miller et al. (2021), l'AUROC mesurée est supérieure sur les jeux de données CIFAR-10 et TinyImageNet : l'augmentation la plus notable est de plus de 8% sur TinyImageNet, alors qu'elle n'est que de 0.4% sur CIFAR-10. En revanche, sur les autres tâches du benchmark, l'évaluation de l'AUROC avec le MOS exhibe des performances inférieures au score proposé par Miller et al. (2021). Cette diminution des performances est faible pour le jeu de donnée SVHN, mais plus importante sur MNIST, CIFAR+10 et CIFAR+50, avec une diminution d'au moins 1.5% quel que soit le taux d'apprentissage.

DIFAIR. Les performances de DIFAIR v1 sont nettement supérieures à celles de DIFAIR v0 sur les jeux de données d'images naturelles, en utilisant le score d'OSR basé sur la distance aux ancres. En moyenne sur l'ensemble des tâches, DIFAIR v1 permet d'obtenir une AUROC supérieure de 5% par rapport à DIFAIR v0. Par contre, l'évaluation de l'AUROC à l'aide du MOS augmente grandement les performances de DIFAIR v0, ce qui permet à cette version de surpasser DIFAIR v1 sur tous les jeux de données, sauf SVHN et TinyImageNet. Cependant, appliqué sur les représentations de DIFAIR v1, le MOS conduit à une AUROC inférieure de 1% sur MNIST et d'au moins 2% sur les autres jeux de données, par rapport au score basé sur

la distance de DIFAIR v1, à l'exception du jeu de données TinyImageNet où l'AUROC mesurée avec le MOS est supérieure de 1.5%. Les résultats de DIFAIR v1 sont équivalents à ceux de la *baseline* sur TinyImageNet.

Comparé à CAC (pour les deux taux d'apprentissage utilisés), DIFAIR v0 (MOS) obtient des performances supérieures sur les jeux de données CIFAR-10, CIFAR+10 et TinyImageNet. L'AUROC mesurée pour la configuration DIFAIR v1 est cependant inférieure à celle de CAC pour tous les jeux de données, excepté TinyImageNet. Si TinyImageNet est exclu des calculs, en moyenne l'AUROC pour DIFAIR v1 est inférieure de 1.5% à celle de CAC (lr=0.1) et de 1.3% pour CAC (lr=0.3). En incluant TinyImageNet, l'AUROC est en moyenne supérieure de 0.1% pour DIFAIR v1 par rapport à CAC (lr=0.1) et inférieure de 0.15% par rapport à CAC (lr=0.3).

Les résultats de DIFAIR v1 sont inférieurs à l'état de l'art d'au moins 9% sur CIFAR-10, CIFAR+10, CIFAR+50 et de 7% sur TinyImageNet. Le jeu de données MNIST étant plus simple, les performances de DIFAIR v1 sont proches des approches de pointe. DIFAIR v1 permet d'obtenir une AUROC supérieure de 1% par rapport à DCHS sur le jeu de données SVHN.

Une tâche d'OSR consiste aussi à classer correctement les données connues. Le tableau 5.2 reporte l'ECS obtenue par les modèles que nous avons entraînés sur les différentes tâches du benchmark. DIFAIR v1 obtient des performances équivalentes aux autres modèles. De plus, l'augmentation du taux d'apprentissage à 0.5 profite grandement au modèle entraîné sur TinyImageNet, pour DIFAIR et pour le modèle entraîné avec entropie croisée. Sur TinyImageNet, les modifications apportées à DIFAIR v1 permettent d'améliorer l'exactitude de 9% par rapport à DIFAIR v0, pour obtenir la deuxième meilleure exactitude sur ce jeu de données, de 1.7% derrière la version de CAC ré-entraînée.

Modèle	MNIST	SVHN	CIFAR10	CIFAR+10	CIFAR+50	TinyImageNet
C_{EC} (lr=0.1)	99.8±0.1	97.7±0.2	96.1±1.2	96.1±0.2	96.1±0.1	61.4±3.1
C_{EC} (lr=0.3)	99.80±0.05	97.70±0.13	96.28±1.24	96.02±0.15	96.48±0.11	74.66±2.28
CAC (lr=0.1)	99.83±0.05	98.15±0.10	96.39±1.19	96.20±0.28	96.43±0.14	73.20±2.43
CAC (lr=0.3)	99.81±0.08	98.11±0.11	96.42±1.16	96.04±0.08	96.44±0.12	77.68±2.16
C_{DIFAIR} v0	99.8±0.1	98.0±0.2	95.8±1.2	96.2±0.3	96.2±0.1	66.8±2.3
C_{DIFAIR} v1	99.81±0.05	98.04±0.10	96.35±1.16	95.82±0.43	96.50±0.11	75.96±1.85

TABLEAU 5.2: **ECS des modèles sur le benchmark de Neal et al. (2018)**. La moyenne de l'ECS (%) et l'écart-type sont reportés sur les cinq séparations de classes connues/inconnues.

5.1.4 Discussion

Ces résultats montrent que TinyImageNet constitue un cas particulier pour la plupart des modèles. Il s'agit d'un jeu de données plus complexe, avec davantage de classes et moins d'images par classe. Les modèles convergent moins bien durant l'entraînement. Dans ces situations, il semble que l'utilisation du score MOS soit plus pertinente que celle d'un score basé sur la distance. En effet, les scores d'AUROC observés avec le MOS sont nettement supérieurs lorsque les scores basés sur la distance sont faibles. Ce comportement peut être expliqué ainsi : si le modèle n'a pas correctement convergé, les données connues ne sont pas représentées suffisamment proche de leurs ancres et peuvent donc être confondues avec des données inconnues. Cela implique que le score d'OSR basé sur la distance soit faible. Il est intéressant de remarquer que, bien que le modèle n'ait pas bien convergé, les attributs sont tout de même extraits avec une valeur plus faible face à des données inconnues, puisque l'AUROC mesurée avec le MOS est plus élevée.

Nous pouvons, pour le futur, imaginer que le score basé sur la distance et le MOS pourraient être combinés afin de tirer parti des avantages de chacun. En effet, le score basé sur la distance permet de mesurer la capacité du modèle à séparer les classes connues et inconnues dans l'espace de représentation, tandis que le MOS permet de mesurer la capacité du modèle à ne pas extraire d'attributs avec une valeur élevée en présence de données inconnues. De plus, utilisé seul, le score MOS ne permet pas de limiter le risque de classification en *open-space* (section 1.2.2).

Les performances de DIFAIR v1 en termes d'AUROC sont légèrement inférieures à celles de DIFAIR v0 (MOS). Cependant, les représentations obtenues avec DIFAIR v0 ne correspondent pas aux attentes que nous avons avec la méthode DIFAIR, car les attributs sont dupliqués sur les dimensions associées à chaque classe. En effet, un objectif de DIFAIR est également d'extraire des attributs distincts sur chaque dimension de la représentation.

L'ECS des modèles entraînés avec DIFAIR v1 est similaire à celle des autres modèles, ce qui signifie que l'utilisation de DIFAIR n'est pas pénalisant pour la classification des données connues. Entre les versions 0 et 1 de DIFAIR, l'AUROC a pu être améliorée sans impact négatif sur l'ECS.

En comparaison à CAC, les performances de DIFAIR sont proches, mais inférieures. La particularité de CAC est de forcer les instances à être représentées sur les ancres et à être éloignées des autres instances, alors que DIFAIR a pour objectif de construire une représentation sémantique, et donc permet à des instances de différentes classes d'être proches dans l'es-

pace de représentation. Cette flexibilité, rend possiblement la séparation des classes plus difficile, tout en permettant l'activation d'attributs face à des données inconnues. Cela peut expliquer la différence de performances par rapport à CAC.

Comparé aux approches de l'état de l'art, les résultats de DIFAIR v1 sont plus faibles mais, contrairement à ces trois approches, DIFAIR n'utilise pas de données supplémentaires identifiées comme inconnues lors de l'entraînement. L'objectif est d'extraire des attributs distincts à partir des données connues uniquement, attributs suffisamment représentatifs de ces données afin que les attributs ne s'activent pas sur des données inconnues. Cet objectif semble atteignable puisque Vaze et al. (2022) obtiennent des performances proches de l'état de l'art avec leur approche MLS utilisant uniquement les données connues. De plus, un des objectifs de DIFAIR est d'obtenir une représentation exprimant une sémantique pour apporter des informations même sur les données inconnues. C'est aussi le cas de DCHS, mais ça n'est pas un objectif des approches ARPL+CS et ConOSR. En particulier ARPL+CS contraint les données inconnues à être représentées loin des données connues, bien qu'elles pourraient présenter des caractéristiques similaires.

Enfin, l'avantage principal de notre méthode est l'interprétabilité au niveau de la représentation extraite. Cette interprétabilité étant possible grâce à des contraintes spécifiques sur l'association des attributs à des classes, il est possible que la prise en compte de ces contraintes se réalise au prix des performances de détection des classes inconnues. De futurs travaux pourraient se concentrer sur le relâchement de ces contraintes, mais toujours en conservant une association des attributs aux classes pour maintenir l'interprétabilité de la représentation.

La section suivante analyse en détails un modèle entraîné sur TinyImageNet et tire avantage de cette interprétabilité afin d'étudier la représentation apprise par DIFAIR v1 dans le but de vérifier si elle nous permet d'aller vers les objectifs décrits dans le chapitre 3 : les attributs doivent être activés seulement en présence de la classe qui leur est associée ou bien en présence de données sémantiquement proches (connues comme inconnues) ; en dehors de cette situation, les attributs ne doivent pas être activés face à des données inconnues. De manière générale, une donnée inconnue ne devrait pas provoquer l'activation de tous les attributs d'une classe connue.

5.2 Analyse de DIFAIR v1

Les modèles étudiés dans cette section ont été entraînés soit sur la séparation 2 du jeu de données CIFAR-10, soit sur la séparation 0 du jeu de données TinyImageNet. Il est important de noter que, sur TinyImageNet, le modèle n'a pas totalement convergé, certaines classes n'ayant pas été correctement apprises. Nous considérons qu'il est tout de même intéressant d'observer le comportement du modèle entraîné sur TinyImageNet, en montrant que l'analyse des représentations est possible, bien que le jeu de données soit plus complexe que CIFAR-10 et contienne davantage de classes, ce qui implique une plus grande représentation. Dans cette section, nous illustrons le comportement de DIFAIR v1 en analysant majoritairement des exemples spécifiques, ce qui n'en fait pas une évaluation exhaustive. Cependant, les exemples choisis permettent d'illustrer le comportement qui a été observé pour diverses instances, et illustrent notamment les limites de notre approche. Nous essayons toutefois d'être exhaustifs lorsque c'est possible, par exemple en observant des distributions d'activations sur l'ensenble des données.

5.2.1 Pénalisation de la corrélation

Avant de s'intéresser aux représentations apprises par DIFAIR v1, vérifions l'influence de la fonction $\mathcal{L}_{\text{correlation}}$ (équation 4.7) pénalisant la corrélation des filtres de la dernière couche de convolution. La valeur de cette métrique indique que la corrélation linéaire des filtres est nulle. La métrique CONV (équation 4.4) mesurant la convergence des poids grâce à l'écart-type indique que les poids ne convergent pas. Afin d'observer l'influence de cette fonction, nous observons l'évolution des poids et comparons les cartes d'activation obtenues en sortie de la dernière couche avec et sans pénalisation de la corrélation. Les cartes d'activations découlant de l'application des filtres sont observées plutôt que les filtres eux-mêmes, car ceux-ci sont en trois dimensions, ce qui complique leur observation.

La figure 5.1 représente l'évolution des cinq poids de l'ensemble $\mathbb{S}_{1,1}$ (défini dans l'équation 4.2), soit les 5 premiers poids des 5 premiers filtres de la dernière couche de convolution, filtres qui sont associées à la classe $c = 1$. Sans pénalisation de la corrélation, la convergence de ces poids était observée au fil de l'apprentissage, et l'évolution de ces poids semblait corrélée. La figure 5.1 montre que la plupart des poids (excepté deux) ont des

valeurs finales différentes. L'utilisation de la fonction de pénalisation de la corrélation des filtres semble donc avoir un impact positif sur la convergence des poids.

Les cartes d'activation obtenues avec un modèle entraîné sans la fonction de pénalisation de l'écart-type sont toutes fortement similaires, comme le montre la figure 5.2a. En effet, puisque les poids des filtres convergent, les cartes d'activations en résultant sont similaires, ce qui signifie que les filtres sont redondants, et que l'information est dupliquée. En revanche, les cartes d'activation obtenues avec un modèle entraîné avec la fonction de pénalisation de la corrélation des filtres exhibent plus de différences, comme le montre la figure 5.2b. Les zones les plus activées sont plus variées, ce qui semble indiquer que les filtres extraient des attributs différents. La figure 5.2 semble indiquer que, grâce à l'utilisation du terme de pénalisation de la corrélation, l'information extraite par les filtres n'est plus dupliquée, les attributs sont plus distincts.

L'impact de cette pénalisation sur la corrélation de l'ensemble des filtres est présenté dans la figure 5.3. La matrice de corrélation obtenue avec DIFAIR v0 (figure 5.3a) indique que les filtres associés à une même classe sont parfaitement corrélés (coefficient de corrélation de 1), ce qui implique une duplication de l'information extraite. En appliquant la fonction de pénalisation de la corrélation, les coefficients de la matrice de corrélation (figure 5.3b) sont nuls pour les filtres associés à une même classe. Ce sont les seuls filtres pénalisés par cette fonction, car deux filtres de classes différentes sont toujours autorisés à être corrélés, dans le cas où les deux classes auraient un attribut commun. Cependant, la matrice de corrélation ne montre pas de forte corrélation entre des filtres appartenant à des classes différentes. La distribution des coefficients de corrélation entre les filtres est présentée dans la figure 5.3c pour des modèles entraînés avec entropie croisée, DIFAIR v0 et DIFAIR v1. La distribution des coefficients de corrélation obtenue avec DIFAIR v1 est plus proche de la distribution obtenue avec l'entropie croisée, mis à part le fait qu'une grande partie de filtres soit totalement décorrélée, ce qui n'est pas le cas avec le modèle entraîné avec l'entropie croisée.

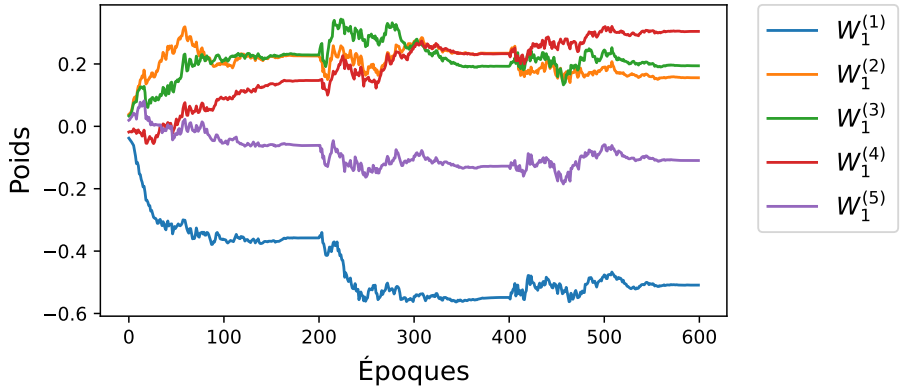
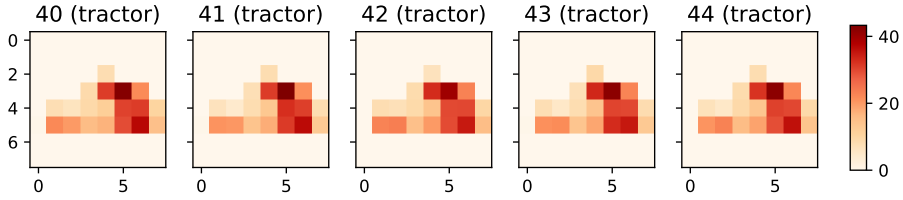
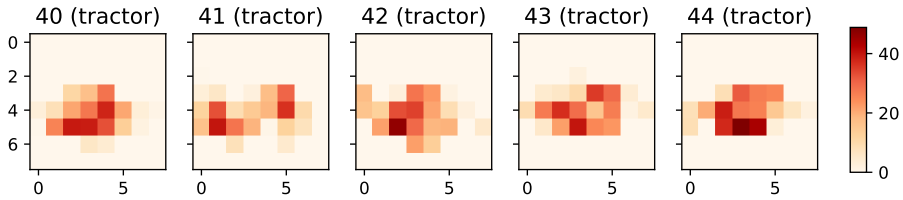


FIGURE 5.1: **Évolution de poids associés à une même classe (DIFAIR v1)**. Les poids étudiés appartiennent à l'ensemble $\mathbb{S}_{1,1}$ (premiers poids de chaque filtre associé à la classe 1).

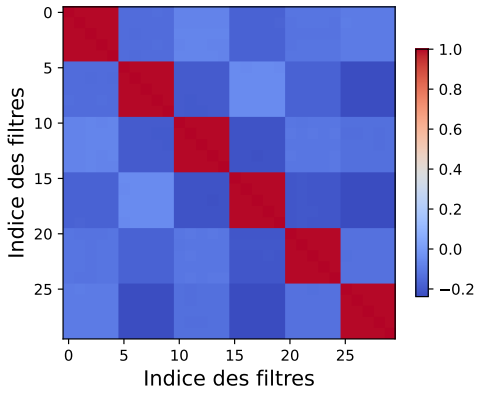


(a) Cartes d'activation obtenues sans pénalisation de la corrélation.

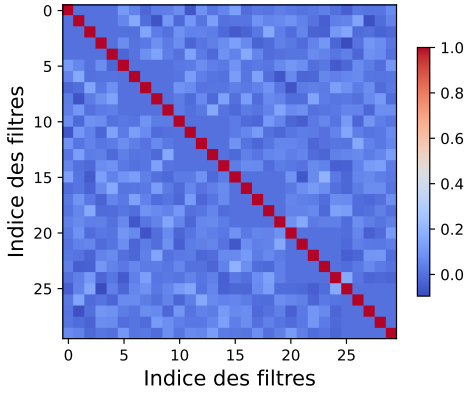


(b) Cartes d'activation obtenues avec la pénalisation de la corrélation.

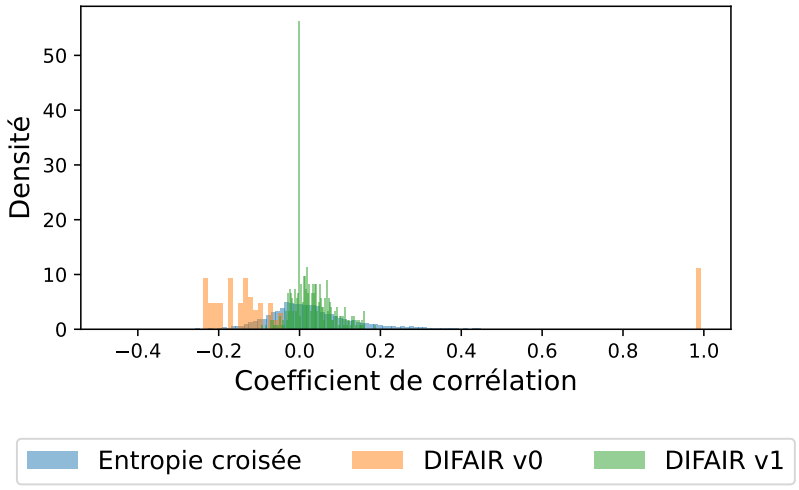
FIGURE 5.2: **Comparaison des cartes d'activation pour les modèles entraînés avec et sans la pénalisation de la corrélation**. Une image de tracteur est donnée en entrée du modèle. Les cartes d'activation associées à cette classe sont affichées. Au-dessus de chaque carte d'activation est indiqué son indice et la classe à laquelle elle est associée. Les graduations des axes servent de coordonnées pour repérer les activations.



(a) Sans pénalisation.



(b) Avec pénalisation.



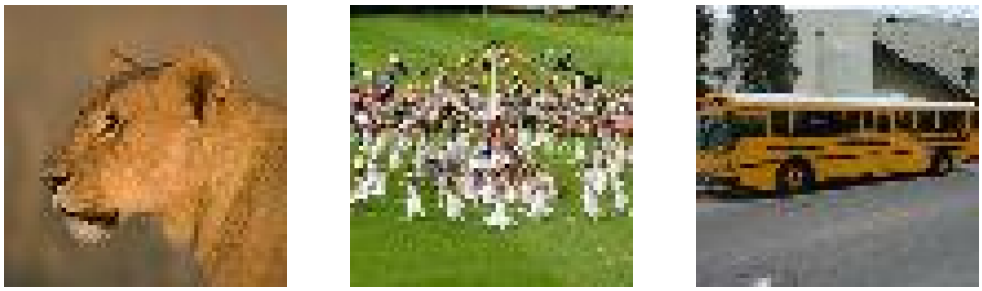
(c)

Figure 5.3: **Corrélation des filtres de la dernière couche de convolution en fonction de la pénalisation de la corrélation**. (a) Matrice de corrélation des filtres pour un modèle entraîné sans pénalisation de la corrélation (DIFAIR v0). (b) Matrice de corrélation des filtres pour un modèle entraîné avec pénalisation de la corrélation (DIFAIR v1). (c) Distribution des coefficients de corrélation entre les filtres pour les modèles entraînés avec entropie croisée, DIFAIR v0 et DIFAIR v1.

5.2.2 Analyse des représentations

Puisque les filtres ne convergent pas et ne sont pas corrélés, DIFAIR v1 devrait extraire des attributs plus distincts que DIFAIR v0 pour représenter chaque classe. Nous étudions les représentations obtenues par DIFAIR v1 sur plusieurs images provenant du jeu de données TinyImageNet. Les images étudiées sont présentées dans la figure 5.4, elles ont été sélectionnées pour présenter différents types de comportements notables que nous avons pu observer dans les représentations analysées. La distinction entre classes connues et inconnues dépend de la séparation du jeu de données utilisée, ici, il s'agit de la séparation 0 de TinyImageNet.

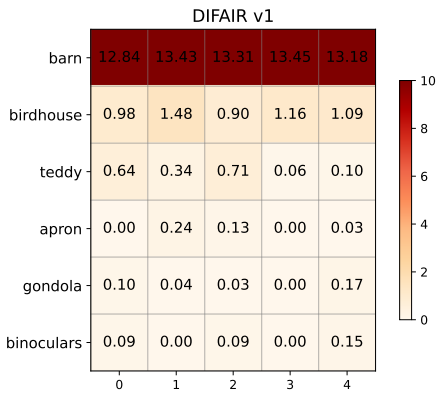
(a) *barn* (connue). (b) *rocking chair*. (c) *lion*.



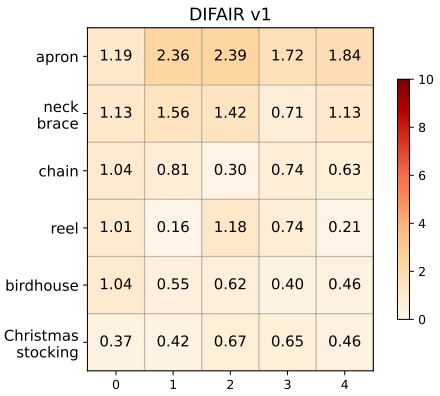
(d) *lion*. (e) *maypole*. (f) *school bus*.

FIGURE 5.4: **Images de classes connues et inconnues étudiées.** Les classes sont *barn* (ferme), *lion* (lion), *rocking chair* (chaise à bascule), *maypole* (arbre de mai) et *school bus* (bus scolaire). Toutes les images correspondent à des classes inconnues sauf la (a), *barn* étant une classe connue.

DIFAIR a pour objectif d'activer des attributs uniquement lorsque des caractéristiques correspondantes sont présentes dans l'image d'entrée, et que les attributs de classes absentes de l'image ne s'activent pas. Définissons, pour cette analyse, qu'un attribut est activé lorsque sa valeur vaut au moins 1, autrement il peut s'agir de bruit.



(a) Représentation de l'image de la classe *barn* (figure 5.4a). (Classe connue)



(b) Représentation de l'image de la classe *rocking chair* (figure 5.4b). (Classe inconnue)

Figure 5.5: **Représentations d'images obtenues avec DIFAIR v1**. Les représentations montrent les *six classes les plus activées* (au lieu de vingt pour TinyImageNet), déterminées par la somme des attributs associés à chaque classe.

Les représentations obtenues pour une image connue et une image inconnue sont présentées dans la figure 5.5. Les attributs activés pour l'image de la classe *barn* (figure 5.5a) sont ceux correspondant à cette classe. Ils sont activés avec une valeur très élevée, autour de 13, ce qui dépasse la valeur attendue de 10. Cependant, l'hypersphère allouée autour de l'ancre permet aux instances d'avoir de telles activations sans que cela soit très pénalisant. La valeur très élevée peut signifier que le réseau est certain de sa prédiction, les caractéristiques représentées par ces attributs ont possiblement été très bien distinguées. Pourtant, même si les attributs sont fortement activés, que le réseau semble exprimer une forte confiance, la classe prédite n'est pas toujours correcte, comme nous le verrons dans la suite. Sur cette représentation, quelques attributs de la classe *birdhouse* (cabane à oiseaux) sont aussi activés, mais avec des valeurs beaucoup plus faibles, autour de 1. Cela peut être dû à une similarité entre ces deux classes (ferme et cabane à oiseaux). En effet, dans ces images zoommées, sans la notion d'échelle, ces classes peuvent paraître assez similaires. D'autres attributs sont un peu plus activés que les autres, par exemple pour la classe *teddy* (ours en peluche) mais comme leur valeur est inférieure à 1 nous considérerons qu'il s'agit de bruit, même si cela a une influence sur la distance par rapport à l'ancre de la vraie classe. Le reste des attributs est désactivé, ce qui correspond au comportement attendu. Les 20 classes ne sont pas montrées, mais puisque

les classes sont triées en fonction de la somme de leurs attributs, les classes suivantes ont forcément des attributs proches de 0. Nous pouvons déduire de cette représentation que le modèle est capable de désactiver les attributs des classes qui ne sont pas présentes dans l'image, du moins *lorsque l'on considère une classe connue*. Un point important à noter est que tous les attributs de la classe *barn* sont activés avec des valeurs assez proches, il n'y a pas d'attribut qui semble avoir été moins détecté que les autres.

En considérant les classes inconnues, plusieurs scénarios d'activations sont possibles. La première possibilité est que le réseau ne détecte pas de caractéristiques connues dans l'image, et donc que les attributs ne soient pas activés, ou alors très faiblement. La représentation de l'image de *rocking chair* (figure 5.5b), une classe inconnue, montre cette situation. Les attributs activés avec une valeur supérieure à 1 sont ceux de plusieurs classes différentes et la valeur maximale d'activation est 2.39. Dans ce cas, le réseau n'a pas réussi à détecter des attributs de classes connues, ce qui est le comportement attendu. Cette instance est donc représentée loin de toutes les ancres et peut facilement être détectée comme correspondant à une classe inconnue.

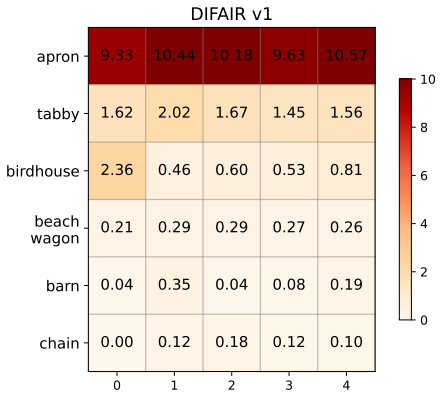
La deuxième possibilité face à une classe inconnue est que tous les attributs d'une classe s'activent avec des valeurs plus élevées, comme le montrent les représentations de plusieurs images inconnues présentes dans la figure 5.6. La plupart du temps, ces activations sont inattendues, car les classes inconnues ne devraient pas présenter toutes les caractéristiques d'une classe connue, et encore moins de plusieurs classes connues.

Par exemple, la représentation de l'image de *maypole* (figure 5.6a) montre que les attributs de la classe *apron* (tablier) sont très fortement activés, cette instance est considérée comme correspondant à une classe connue par le modèle. Une analyse visuelle de la représentation permettrait de dire qu'il y a un problème puisque des attributs de plusieurs classes différentes, qui n'ont pas vraiment de lien sémantique sont activés :*apron* (tablier), *tabby* (chat) ou encore *birdhouse* (cabane à oiseau). Cependant, en utilisant la distance euclidienne, cette instance est représentée assez proche de l'ancre de la classe *apron*.

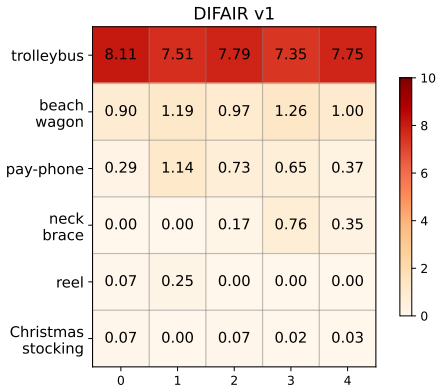
Une situation acceptable où des attributs s'activeraient face à une image inconnue serait si l'image contient vraiment des caractéristiques d'une classe connue. C'est possiblement le cas lorsque la classe inconnue est sémantiquement proche d'une classe connue. Par exemple, la représentation de l'image de *school bus* (figure 5.6b) présente des attributs de la classe *trolleybus* ac-

tivés avec une valeur autour de 7.5. Là aussi, tous les attributs de la classe *trolleybus* sont activés avec des valeurs assez proches, mais ici cela peut être justifié par la forte similarité entre les deux classes.

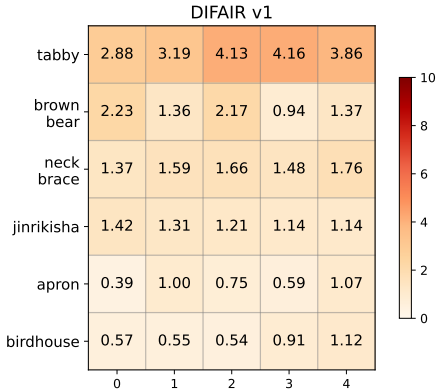
Il arrive tout de même que face à des classes inconnues sémantiquement proches de classes connues, le modèle ne soit pas capable de détecter cette similarité, selon le type d'image de la classe (gros plan, objet positionné



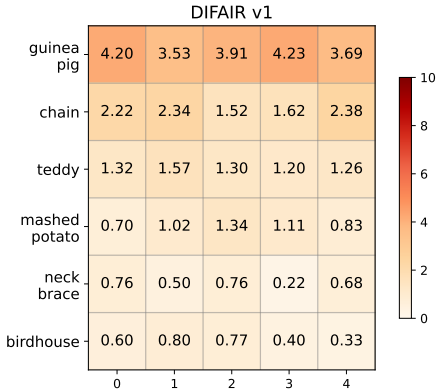
(a) Représentation de l'image de la classe *maypole* (figure 5.4e). (Classe inconnue)



(b) Représentation de l'image de la classe *school bus* (figure 5.4f). (Classe inconnue)



(c) Représentation de l'image de la classe *lion* (figure 5.4c). (Classe inconnue)



(d) Représentation de l'image de la classe *lion* (figure 5.4d). (Classe inconnue)

Figure 5.6: **Représentations d'images inconnues obtenues avec DIFAIR v1**. Les représentations montrent les *six classes les plus activées* (au lieu de vingt pour TinyImageNet), déterminées par la somme des attributs associés à chaque classe.

différemment, *etc.*). Par exemple, le modèle détecte bien la similarité avec un chat pour l'image de *lion* (figure 5.4c), représentant un lion allongé. En effet, dans la représentation en figure 5.6c, les attributs de la classe *tabby* (chat) sont activés ainsi que certains attributs de la classe *brown bear* (ours brun) mais plus faiblement. Globalement, les valeurs d'activation sont faibles et la donnée reste considérée comme inconnue. Par contre, pour l'image de *lion* (figure 5.4d), représentant un gros plan sur la tête du lion, aucun attribut de la classe *tabby* n'est activé dans la représentation en figure 5.6d. Les attributs activés sont ceux de la classe *guinea pig* (cochon d'Inde). Le modèle n'a potentiellement pas appris à représenter les chats sous cet angle. La deuxième classe la plus activée est *chain* (chaîne), ce qui à première vue n'a aucun rapport avec l'image. Il y a donc des problèmes au niveau de l'extraction des attributs dans ce modèle.

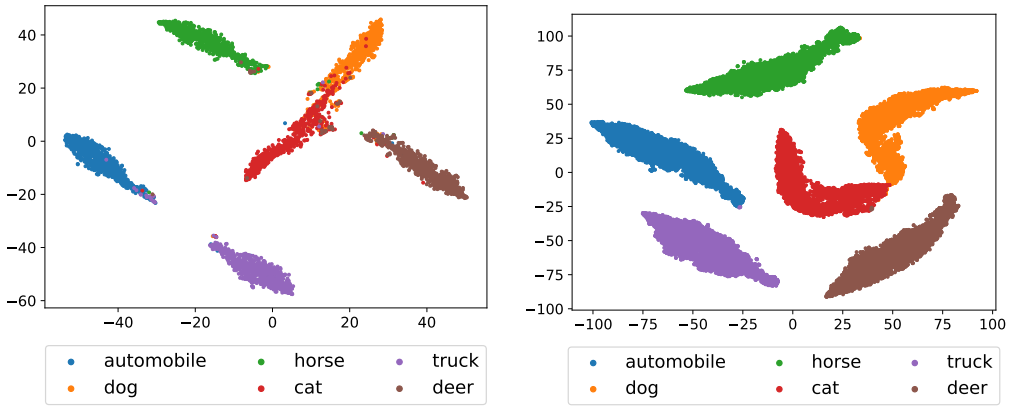
5.2.3 Sémantique dans la représentation

Malgré les problèmes apparents de reconnaissance des caractéristiques, qui s'expriment à travers l'analyse des représentations des classes connues et inconnues, nous souhaitons vérifier si le réseau a été capable d'apprendre une représentation sémantique des données. Pour cela, nous étudions la séparation 2 du jeu de données CIFAR-10, car le nombre réduit de classes permet de mieux vérifier si deux classes sémantiquement proches sont représentées à proximité l'une de l'autre.

Nous utilisons la méthode de réduction de dimensions t-SNE pour visualiser l'espace de représentations obtenu par le modèle entraîné sur cette séparation. La figure 5.7 montre les projections dans un espace en deux dimensions des représentations apprises par DIFAIR v1. La figure 5.7a montre le résultat de l'application de t-SNE sur la totalité des représentations obtenues à partir des images d'entraînement. Cette figure semble montrer qu'il y a un lien entre les couples de classes sémantiquement proches, les groupes d'instances des classes *automobile* et *truck* (voiture et camion) sont alignés, tout comme les classes *deer* et *horse* (cerf et cheval). En plus d'être alignés, les groupes d'instances des classes *cat* et *dog* (chat et chien) sont en contact. Cette représentation pourrait laisser penser qu'il y a bien un lien entre les classes sémantiquement proches, cependant, le piège est que cette représentation est obtenue en utilisant les représentations des images d'entraînement, pour lesquelles le modèle a parfois fait des prédictions incorrectes. Cette figure peut donc être trompeuse, dans le sens où certaines données mal classées peuvent se situer entre les deux ancres des classes sémantiquement proches, d'où l'alignement des groupes d'instances. Ce qui

pourrait être interprété comme une forte proximité entre les classes chat et chien correspond en réalité à la difficulté du modèle à distinguer ces deux classes.

La figure 5.7b montre le résultat de l'application de t-SNE uniquement sur les représentations des images d'entraînement pour lesquelles le modèle a fait une prédiction correcte. Le résultat est très différent, les groupes d'instances des classes *cat* et *dog* sont toujours en contact, le modèle ayant des difficultés à départager ces classes. Cependant, les paires de classes sémantiquement proches ne sont plus alignées et la proximité n'est pas forcément exprimée, par exemple les classes *deer* et *horse* sont séparées, ce qui semble indiquer que le modèle n'apprend pas de relations sémantiques entre ces classes. Pour confirmer cette hypothèse, nous étudions la distribution des activations de chaque attribut face à une classe.



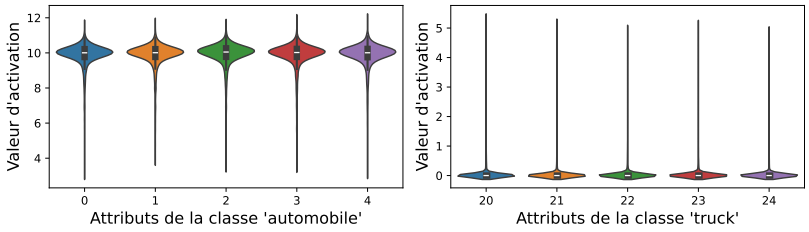
(a) (b)

Figure 5.7: **Visualisation t-SNE des représentations apprises par DIFAIR v1**. Les représentations sont obtenues à partir des images d'entraînement de la séparation 2 de CIFAR-10. (a) t-SNE appliqué sur l'ensemble des représentations. (b) t-SNE appliqué uniquement sur les représentations des images correctement classées.

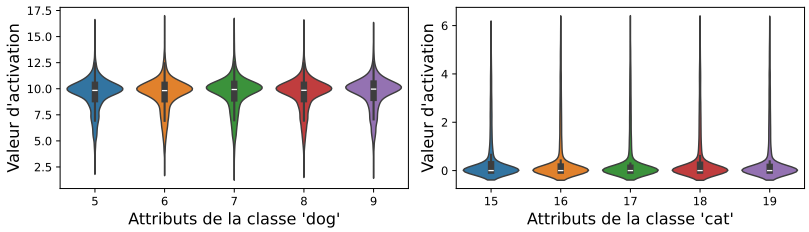
Afin de vérifier si les représentations contiennent des activations conjointes d'attributs, exprimant une proximité sémantique entre les classes, nous étudions la distribution des activations des attributs des classes sémantiquement proches. Pour éviter le biais décrit précédemment, où l'activation d'attributs partagés ne correspondrait qu'à une difficulté de classification, seules les représentations d'instances correctement classées sont considérées dans cette analyse. La figure 5.8a représente la distribution des activations des attributs pour les classes *automobile* et *truck* (voiture et camion), lorsque

la vraie classe est *automobile*. Les diagrammes en violon représentant les distributions indiquent que les attributs de la classe *automobile* semblent s'activer de manière similaire, autour de la valeur 10. Il n'y a pas d'attribut sous-représenté ou sur-représenté. En observant les activations des attributs de la classe *truck*, sémantiquement proche, nous remarquons que les attributs de cette classe sont majoritairement désactivés, avec une valeur autour de 0. Cela indique que le modèle n'apprend pas à partager d'attributs entre ces classes, et donc au vu des caractéristiques de DIFAIR, il n'y a pas de sémantique exprimée dans cette représentation.

La figure 5.8b montre la distribution des activations des attributs pour les classes *cat* et *dog* (chat et chien), lorsque la vraie classe est *dog*. Bien que les distributions soient plus allongées que pour les classes *automobile* et *truck*, indiquant une plus grande variabilité dans les activations, les attributs de la classe *dog* s'activent aussi majoritairement autour de la valeur 10 et ceux de la classe *cat* sont très proches de 0. Ces distributions plus étalées représentent sûrement la difficulté du modèle à distinguer ces deux classes, mais n'indique pas non plus de sémantique dans la représentation.



(a) Distribution des activations pour la classe *automobile*.



(b) Distribution des activations pour la classe *dog*.

FIGURE 5.8: **Distribution des activations des attributs pour des classes sémantiquement proches**. Les distributions sont obtenues à partir des représentations des images correctement classées de la séparation 2 de CIFAR-10.

Dans la section 5.2.2, la représentation de l'image connue de la classe *barn* montrait que des attributs de la classe *birdhouse* étaient activés avec des valeurs faibles. Cela aurait pu indiquer que le modèle a appris qu'il y a une proximité entre ces classes. Cependant, en étudiant la distribution des activations des attributs pour une classe sur le jeu de données CIFAR-10 (figure 5.8), nous n'observons pas d'activation conjointe d'attributs des classes sémantiquement proches. Sachant que sur TinyImageNet la convergence du modèle est incomplète [1], cela peut expliquer que, face à une classe quelconque, des attributs d'autres classes soient tout de même activés, et parfois pour des classes qui n'ont pas de lien sémantique avec la vraie classe. Il n'y a donc probablement pas non plus de sémantique dans les représentations apprises par DIFAIR v1 sur TinyImageNet.

5.3 Limites de DIFAIR v1

Les résultats obtenus en OSR avec DIFAIR v1, ainsi que l'analyse du modèle, montrent que seuls certains objectifs de DIFAIR sont atteints. En effet, les résultats en OSR (section 5.1) indiquent que les modèles sont capables d'être utilisés correctement pour la tâche de classification d'images. Cela signifie que les données connues sont représentées séparées les unes des autres, et que les attributs associés à la classe y, représentant la vérité terrain, s'activent, tandis que les attributs d'autres classes sont désactivés : de cette façon, l'instance est représentée proche de son ancre. Cependant, l'analyse des représentations (section 5.2.2) démontre plusieurs limites de DIFAIR v1.

Les représentations obtenues partagent généralement un point commun : les attributs d'une classe s'activent tous ensemble, avec des valeurs relativement proches. C'est aussi le cas face à des données inconnues. Cela dénote un comportement non souhaité, car cinq attributs décrivant une classe qui n'était pas présente dans l'image ne devraient pas être activés avec une valeur élevée. Nous émettons l'hypothèse que le modèle apprend tout de même une certaine corrélation ou dépendance entre les attributs d'une même classe. Cela mène à l'activation de tous les attributs d'une classe lorsque le modèle détecte qu'une classe est plus probable que les autres pour une instance donnée, afin de respecter l'objectif d'optimisation de la distance par rapport à l'ancre.

1. Sur TinyImageNet, les performances obtenues sont plus faibles que sur les autres jeux de données : le modèle a des difficultés à séparer les classes.

Les attributs extraits ne sont pas indépendants, bien que les filtres correspondant à ces attributs soient décorrélés et que les attributs représentent probablement des caractéristiques distinctes, d'après les cartes d'activation obtenues. En pratique, si les attributs extraits étaient indépendants, masquer une partie de l'image devrait n'activer qu'une partie des attributs, et non pas tous. Cependant, les résultats obtenus indiquent clairement que les attributs ne sont pas indépendants.

Une explication possible à l'activation de tous les attributs d'une classe est que le modèle apprend à extraire uniquement les attributs présents dans *toutes les images* d'une classe qu'il a vues lors de l'entraînement. En effet, puisque chaque image doit être représentée dans l'hypersphère, le moyen le plus efficace pour y parvenir est d'activer tous les attributs de cette classe et pas des autres. Dans cette logique, le modèle DIFAIR v0 extrayait toujours le même attribut sur les $\mathcal{N}$ dimension. Il est donc plausible que le modèle DIFAIR v1 extraie $\mathcal{N}$ attributs communs à toutes les images d'une classe.

Cela expliquerait pourquoi le modèle active tous les attributs d'une classe lorsqu'il détecte la présence d'une classe et cela expliquerait aussi pourquoi des classes inconnues activent aussi des filtres : les filtres sont trop généralisés par rapport aux données d'entraînement et pas assez spécifiques aux attributs qui définissent réellement la classe. La représentation obtenue pour l'image inconnue de la classe *school bus* (figure 5.6b) montre que tous les attributs de la classe *trolleybus* sont activés. Cela soulève des questions quant à la capacité du modèle à extraire des attributs pertinents pour décrire une classe. La caractéristique qui distingue la classe *school bus* de la classe *trolleybus* est que la première n'est pas relié à des caténaires. Cependant, comme le modèle n'a pas eu dans son jeu d'entraînement des images de ces deux classes, il n'a pas pu apprendre que cette caractéristique était importante pour la classe *trolleybus*, permettant de la distinguer d'autres types de bus. De ce point de vue, l'argument relevé par plusieurs approches en OSR (Ge et al., 2017; Neal et al., 2018) stipulant qu'il ne faut pas seulement modéliser les données connues, mais aussi les données inconnues est compréhensible. Malgré tout, l'idée d'extraire un certain nombre d'attributs distincts pour chaque classe visait à permettre au modèle d'apprendre à représenter de telles caractéristiques qui ne sont pas forcément discriminantes dans la tâche courante.

L'hypothèse que nous soulevons, selon laquelle le modèle extrait des attributs présents dans toutes les images est difficile à vérifier et des recherches supplémentaires sont nécessaires pour vraiment comprendre pourquoi le modèle active tous les attributs d'une classe.

Lorsque le modèle converge correctement, il ne semble pas maintenir de sémantique dans sa représentation. Ce qui pouvait s'apparenter à une sémantique entre les classes de TinyImageNet correspond probablement à une difficulté du modèle à distinguer les classes. L'observation des distributions d'activations sur le jeu de données CIFAR-10, sur lequel le modèle a mieux convergé, montre qu'il n'y a pas d'activations des attributs d'autres classes. Pourtant, un camion a généralement toujours un point commun avec une voiture : des pneus. Par conséquent, soit le réseau ne se sert pas de cette caractéristique pour distinguer les deux classes, soit le réseau a appris grâce au contexte autour du pneu à déterminer s'il s'agit d'un pneu de voiture ou d'un pneu de camion. En théorie, rien n'indique non plus que le modèle soit capable d'extraire un attribut de manière aussi précise. L'idée d'extraire des attributs distincts avec DIFAIR est de permettre au modèle d'extraire des attributs même s'ils sont communs à plusieurs classes, car une fois les attributs discriminants appris, le modèle doit encore apprendre d'autres attributs, distincts des premiers. Pourtant, le fait d'avoir $\mathcal{N} = 20$ dimensions allouées à chaque classe n'améliore pas les résultats, ce qui peut se comprendre, car il est sûrement difficile d'extraire 20 attributs distincts pour décrire une classe. Cependant, dans ce cas, l'extraction de 10 attributs aurait pu présenter de meilleurs résultats. La question se pose de savoir si les attributs sont réellement distincts. Une piste pour répondre à cette interrogation serait d'utiliser des méthodes d'explicabilité (Zhou et al., 2016) afin de déterminer quelles parties de l'image activent les attributs.

Le fait d'autoriser les instances à être représentées dans des hypersphères avait pour objectif de permettre au modèle d'apprendre des caractéristiques communes à plusieurs classes, en les exprimant sur les dimensions de chacune des classes. Cependant, les résultats obtenus montrent qu'il s'agit probablement d'une action trop pénalisante : activer un attribut d'une autre classe éloigne trop l'instance de son ancre. Bien que les attributs extraits semblent distincts, ils ne sont pas assez spécifiques pour décrire la classe au niveau sémantique, mais seulement au niveau discriminant.

Pour conclure, le modèle entraîné avec DIFAIR v1 résout dans une certaine mesure la tâche demandée : placer les instances autour des ancres et par conséquent n'activer que les attributs liés à la vraie classe. Cependant, bien qu'il soit permis au modèle d'activer des attributs d'autres classes grâce à l'allocation d'une hypersphère autour des ancres, ce comportement n'a pas émergé. Il est probable que cette permissivité dans l'activation des autres attributs soit en fait trop pénalisante. Les contraintes appliquées ne sont pas assez permissives par rapport à l'utilisation de l'entropie croisée qui laisse vraiment toutes les libertés au modèle pour extraire une représen-

tation qui permet de résoudre le problème, et d'où une sémantique émerge. Les résultats obtenus avec DIFAIR v1 poussent à réfléchir plus loin à la manière dont les attributs peuvent être extraits et partagés entre les classes, tout en conservant l'interprétabilité apportée. Notamment, il faut trouver un moyen de contraindre le modèle à extraire des attributs qui soient plus indépendants et plus spécifiques à la classe qu'ils décrivent, pas uniquement discriminants par rapport aux autres classes. Ces attributs doivent pouvoir être partagés (dupliqués dans la représentation) entre les classes, sans que cela ne soit pénalisant durant l'apprentissage.

Bien que le modèle entraîné avec la fonction de perte d'entropie croisée présente de meilleurs résultats que DIFAIR, ce type de modèle ne permet ni de réaliser une analyse aussi poussée des représentations, ni de questionner ce à quoi devrait correspondre la représentation d'une image.

Rappelons que l'approche que nous proposons a pour objectif d'activer des attributs avec une certaine valeur α. Étant donné que la valeur d'un attribut de la représentation correspond à la moyenne des valeurs de la carte d'activation associée à l'attribut, cela signifie que nous optimisons le modèle de manière que cette moyenne atteigne une certaine valeur. Cependant, si une carte d'activation présente une valeur très élevée en un point, indiquant la détection d'un motif, et des valeurs très faibles ailleurs, la moyenne de cette carte est faible et l'attribut dans la représentation est désactivé. Néanmoins, cet effet est potentiellement atténué par le fait que les cartes d'activations de la dernière couche ont une taille réduite : 4×4 pour des images de taille 32×32 en entrée, mais tout de même 8×8 pour des images de TinyImageNet.

Au vu des limites relevées grâce à l'analyse des représentations, nous proposons un chapitre prospectif dans lequel nous présentons de nouvelles contraintes d'optimisation pour extraire des attributs plus indépendants et essayer d'obtenir des représentations exprimant des relations sémantiques, le tout en maintenant l'interprétabilité de la représentation extraite.

Chapitre 6

Pistes pour dépasser les limites de DIFAIR v1

Résumé

Ce chapitre propose une ouverture sur une nouvelle approche de DIFAIR en décrivant les travaux préliminaires et les expériences menées pour surmonter les limites observées sur la version DIFAIR v1. Pour cela, deux fonctions de perte ont été proposées pour contraindre la représentation apprise. Les premiers résultats obtenus sur deux jeux de données sont présentés, avant de conclure sur des perspectives de recherche.

Sommaire

L'évaluation de l'approche proposée dans le chapitre 5 a révélé plusieurs défis à surmonter pour atteindre les objectifs fixés par DIFAIR. Ce chapitre, dans une perspective d'ouverture, vise à présenter de nouvelles pistes pour surmonter ces défis. Les résultats exposés sont préliminaires et ne sont pas testés extensivement sur le benchmark d'OSR.

6.1 Cadrage des limites de DIFAIR v1

Deux problèmes principaux ont été observés lors de l'évaluation de DIFAIR v1. En premier lieu, il a été constaté que les représentations obtenues ne contiennent pas d'informations sémantiques sur les classes. En effet, les attributs de classes sémantiquement proches ne s'activent pas en même temps, ce qui empêche ces classes d'être représentées à proximité dans l'espace de représentation. La distance euclidienne ne semble pas adaptée pour permettre ce partage d'attributs entre classes, car l'activation d'attributs d'autres classes éloigne l'instance de son ancre, ce qui cause une pénalisation plus forte. Construire une représentation sémantique des données permettrait d'obtenir une représentation ayant plus de sens du point de vue de l'interprétation humaine, et permettrait notamment d'extraire des informations concernant des données inconnues en fonction de leur proximité avec des données connues. Cela permet aussi au réseau de mieux apprendre à représenter les données, en capturant des relations sémantiques pertinentes. Par exemple, si le modèle doit différencier les classes chat et chien et est contraint à déterminer des attributs qui s'activent exclusivement sur les images de chats et jamais sur la classe chien, la tâche peut s'avérer plus difficile. Au contraire, activer un attribut, même commun à deux classes, facilite la distinction de ces deux classes par rapport à d'autres.

Deuxièmement, dans les représentations extraites par DIFAIR v1, tous les attributs d'une classe s'activent en même temps, parfois même en l'absence de la classe dans l'image, notamment face à des données inconnues. Autrement dit, le réseau ne semble pas apprendre à activer les attributs en présence d'une caractéristique spécifique d'une classe, mais plutôt en fonction de la présence globale de cette classe. L'origine de ce comportement vient probablement du fait que l'optimisation pousse le réseau à activer tous les attributs d'une classe en même temps pour valider l'objectif de placer l'instance dans son hypersphère associée. Un attribut désactivé pour une classe est trop pénalisant pour le réseau lorsque l'évaluation est réalisée par la distance euclidienne aux ancres représentant les classes. Le modèle semble donc apprendre une certaine dépendance entre les attributs. Nous

pensons qu'une représentation où chaque attribut serait extrait indépendamment, c'est-à-dire qu'il pourrait s'activer même si les autres attributs de la classe ne sont pas activés, serait plus bénéfique pour le modèle. En particulier, comme nous ne souhaitons pas qu'une donnée inconnue active l'ensemble des attributs appris par une classe connue, il est important que le modèle puisse activer des attributs indépendamment les uns des autres. C'est aussi le cas pour les classes connues, car une image ne contient pas forcément tous les attributs représentatifs d'une classe : certains attributs peuvent être cachés ou non visibles en fonction de l'orientation de la classe.

En conséquence, l'un des objectifs de cette nouvelle approche est de permettre l'apprentissage d'une représentation sémantique en activant des attributs de plusieurs classes lorsque celles-ci sont sémantiquement proches. De plus, l'optimisation doit permettre à des attributs décrivant une classe d'être désactivés même lorsque cette classe est présente dans l'image : un attribut serait désactivé dans le cas où les caractéristiques auxquelles il est associé ne sont pas visibles ou ne sont pas présentes dans l'image.

Pour répondre à ces défis, nous proposons d'optimiser de nouveaux objectifs visant à mieux prendre en compte ces deux aspects. Le premier objectif consiste à contrôler le nombre d'attributs qui s'activent pour une classe, et le second à contrôler la magnitude (la norme) du vecteur formé par les attributs associés à une même classe.

6.2 Fonction de perte

La fonction de perte proposée est différente de celle de DIFAIR v1. Cependant, le nom DIFAIR est conservé, car il désigne l'objectif de construire une représentation spécifique, indépendamment de la fonction de perte utilisée.

La figure 6.1 illustre l'approche adoptée pour définir cette nouvelle fonction de perte. Cette fonction est définie différemment sur plusieurs sous-intervalles ($n_{\text{intervalles}}$) de $[0, \infty[$. Notons ces intervalles $\mathcal{I}_i$ avec $i \in \{1, \ldots, n_{\text{intervalles}}\}$. Sur chaque sous-intervalle $\mathcal{I}_i$, la fonction de perte est définie par une fonction linéaire f_i. Dans la suite, nous utilisons les notations suivantes, où :

- s_i est la pente de la fonction f_i sur l'intervalle $\mathcal{I}_i$,
- v est la valeur *souhaitée* pour l'objectif optimisé,
- m_b est la marge inférieure (basse) acceptée pour v, définissant l'intervalle $[v - m_b, v[$ de valeurs *acceptables* pour l'objectif,

— l_b est la valeur de la fonction de perte au point $v - m_b$,

— m_h est la marge supérieure (haute) acceptée pour v, définissant l'intervalle $[v, v + m_h[$ de valeurs *acceptables* pour l'objectif,

— l_h est la valeur de la fonction de perte au point $v + m_h$.

La fonction de perte est définie de manière que sa valeur et sa pente soient faibles sur les intervalles de valeurs acceptables de l'objectif, c'est-à-dire $[v - m_b, v[$ et $[v, v + m_h[$, et une pente plus forte sur les autres intervalles pour pénaliser les valeurs non acceptables. Les valeurs acceptables ou non sont déterminées en fonction des objectifs définis pour la tâche.

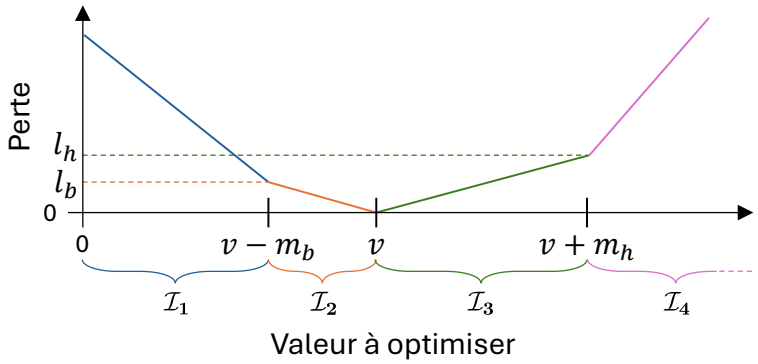


Figure 6.1: **Décomposition de la fonction de perte en intervalles.** La fonction de perte est définie sur $n_{\text{intervalles}}$ sous-intervalles de $[0, \infty[$. Sur chaque intervalle $\mathcal{I}_i$, la fonction de perte est une fonction linéaire ayant une pente s_i.

Une telle fonction de perte est dérivable sur chaque intervalle, mais pas aux points de jonction entre les intervalles. En effet, puisque sur chaque intervalle f_i est une fonction linéaire, sa dérivée est $f_i'(x) = s_i$ pour $x \in \mathcal{I}_i$. Puisque généralement $s_i \neq s_{i+1}$, la dérivée à gauche et à droite des points de jonction entre les intervalles est différente, ce qui rend la fonction non dérivable en ces points. Cependant, cela ne pose pas de problème pour l'optimisation avec la bibliothèque TensorFlow qui renvoie le gradient de la fonction définie pour ce point, selon l'intervalle $\mathcal{I}_i$ dans lequel il se trouve lors de l'implémentation de la fonction. L'optimisation par descente de gradient est donc possible, mais le gradient change aux points de jonction.

Contrairement à DIFAIR v1, la représentation $\boldsymbol{z}$ n'est pas considérée comme un unique vecteur de taille $\mathcal{N} \times K$ (où $\mathcal{N}$ est le nombre de dimensions allouées à chaque classe et K le nombre de classes). Dans cette nouvelle version de DIFAIR, chaque classe $j \in \{1, 2, \ldots, K\}$ est représentée

par un vecteur $\boldsymbol{r}^{(j)}$ de taille $\mathcal{N}$. Les vecteurs $\boldsymbol{r}^{(j)}$ sont extraits de la représentation $\boldsymbol{z}$, ainsi : $\boldsymbol{z} = [\boldsymbol{r}^{(1)}, \boldsymbol{r}^{(2)}, \ldots, \boldsymbol{r}^{(K)}]$, où l'opérateur $[\,\cdot\,,\,\cdot\,]$ représente la concaténation des vecteurs. Chaque vecteur a des objectifs spécifiques, en fonction de s'il est associé à la classe y de l'image ou non. Par exemple, si $y = 1$, alors les attributs du vecteur $\boldsymbol{r}^{(1)}$ doivent être activés, tandis que les autres vecteurs ne doivent pas être activés à moins d'exprimer des caractéristiques communes à la classe 1. Ainsi, deux fonctions de perte doivent être définies pour chaque objectif : $\mathcal{L}$, qui pénalise le vecteur d'attributs $\boldsymbol{r}^{(y)}$ associé à la classe cible y correspondant à la vérité terrain ; et une fonction antagoniste $\overline{\mathcal{L}}$ qui pénalise les vecteurs $\boldsymbol{r}^{(j)}$ associés aux *autres classes* $(j \neq y)$.

6.2.1 Pénalisation du nombre d'attributs activés

Pour un vecteur d'attributs $\boldsymbol{r}^{(j)}$ représentant la classe j avec $j \in \{1, 2, \ldots, K\}$, le nombre d'attributs activés $A^{(j)}$ est défini par :

$$A^{(j)} = \sum_{i=1}^{\mathcal{N}} \mathbb{1}_{\{r_i^{(j)} > 0.5 \cdot \alpha\}}, \tag{6.1}$$

avec α la valeur d'activation souhaitée pour les attributs. Nous considérons, ici, qu'un attribut est activé s'il a une valeur supérieure à $0.5 \cdot \alpha$; cette borne est déterminée selon notre conception théorique du problème. Rappelons que la fonction indicatrice $\mathbb{1}_{\{c\}}$ vaut 1 si la condition c est vraie, et 0 sinon. La valeur $A^{(j)}$ est optimisée avec les fonctions décrites ci-dessous, de façon à ce que, pour chaque classe, le nombre souhaité d'attributs soit activé.

Classe y. La fonction $\mathcal{L}_{\text{attr}}(x)$ pénalise le nombre d'attributs activés, x, pour la classe y, correspondant à la vérité terrain de l'image $\boldsymbol{X}$. L'objectif de cette fonction est de contraindre l'activation de $v^{(\mathcal{L})}$ attributs dans le vecteur $\boldsymbol{r}^{(y)}$. Elle est définie sur l'intervalle $[0, v^{(\mathcal{L})}]$: dans ce cas, puisqu'il y a $\mathcal{N}$ attributs associés à chaque classe, $v^{(\mathcal{L})} = \mathcal{N}$ et la fonction est définie sur $[0, \mathcal{N}]$. La marge $m_b^{(\mathcal{L})}$ indique le nombre d'attributs qui peuvent être désactivés dans le vecteur $\boldsymbol{r}^{(y)}$, de façon à de ne pas toujours contraindre tous les attributs représentant une classe à être activés : il est probable qu'une image ne les contienne pas tous, par exemple dans le cas où la classe ne serait pas visible dans son intégralité. Nous notons a le nombre d'attributs minimum qui doivent être activés pour la classe y, soit :

$$a = \mathcal{N} - m_b^{(\mathcal{L})}.$$

L'intervalle $[0, \mathcal{N}]$ est divisé en deux sous-intervalles :

$$\mathcal{I}_1 = [0, a[\quad \text{et} \quad \mathcal{I}_2 = [a, \mathcal{N}].$$

La fonction $\mathcal{L}_{\text{attr}}$ est définie sur ces intervalles comme suit :

$$\mathcal{L}_{\text{attr}}(x) = \begin{cases} -s_1^{(\mathcal{L})} \cdot (x - a) + l_b^{(\mathcal{L})} & \text{pour } x \in \mathcal{I}_1, \\ -s_2^{(\mathcal{L})} \cdot (x - \mathcal{N}) & \text{pour } x \in \mathcal{I}_2, \end{cases}$$

avec :

$$s_2^{(\mathcal{L})} = \frac{l_b^{(\mathcal{L})}}{m_b^{(\mathcal{L})}}.$$

Cette fonction est donc paramétrée par le nombre d'attributs associés à chaque classe, $\mathcal{N}$, la pente $s_1^{(\mathcal{L})}$ sur l'intervalle $[0, a]$, la marge d'activation des attributs $m_b^{(\mathcal{L})}$ et la valeur de la fonction de perte $l_b^{(\mathcal{L})}$ au point a. La perte vaut 0 pour $x = \mathcal{N}$, ce qui signifie que le réseau n'est pas pénalisé si tous les attributs sont activés dans le vecteur $\boldsymbol{r}^{(y)}$. Si $x = a$, la fonction vaut $l_b^{(\mathcal{L})}$, une valeur choisie empiriquement et de l'ordre de 0.5 pour ne pas pénaliser trop fortement le réseau, puisque nous considérons que a est un nombre suffisant d'attributs activés pour la classe y.

Autres classe. La fonction $\overline{\mathcal{L}}_{\text{attr}}(x)$ pénalise le nombre d'attributs activés pour les autres classes. L'objectif de cette fonction est de réduire le nombre d'attributs activés dans les vecteurs associés aux autres classes, $\boldsymbol{r}^{(j)}$ avec $j \neq y$. Cette fonction est aussi définie sur l'intervalle $[0, \mathcal{N}]$. Cependant, le nombre d'attributs activés souhaité est $v^{(\overline{\mathcal{L}})} = 0$. Afin que la représentation puisse exprimer une sémantique, nous voulons tout de même permettre au modèle d'activer des attributs d'autres classes, s'ils représentent des caractéristiques communes à la classe y. La marge $m_h^{(\overline{\mathcal{L}})}$ indique le nombre d'attributs qui peuvent être activés pour les autres classes. Pour que le modèle soit capable de faire une distinction entre le vecteur de la classe y et les autres, il faut que $m_h^{(\overline{\mathcal{L}})} < \mathcal{N} - m_b^{(\mathcal{L})}$; autrement dit, le nombre d'attributs maximum activé pour les autres classes est strictement inférieur au nombre d'attributs minimum activé pour la classe y. L'intervalle $[0, \mathcal{N}]$ est donc divisé en deux sous-intervalles :

$$\mathcal{I}_1 = [0, m_h^{(\overline{\mathcal{L}})}] \quad \text{et} \quad \mathcal{I}_2 = [m_h^{(\overline{\mathcal{L}})}, \mathcal{N}].$$

La fonction $\overline{\mathcal{L}}_{\text{attr}}$ est définie comme suit :

$$\overline{\mathcal{L}}_{\text{attr}}(x) = \begin{cases} s_1^{(\overline{\mathcal{L}})} \cdot x & \text{pour } x \in \mathcal{I}_1, \\ s_2^{(\overline{\mathcal{L}})} \cdot (x - m_h^{(\overline{\mathcal{L}})}) + l_h^{(\overline{\mathcal{L}})} & \text{pour } x \in \mathcal{I}_2, \end{cases}$$

avec :

$$s_1^{(\overline{\mathcal{L}})} = \frac{l_h^{(\overline{\mathcal{L}})}}{m_h^{(\overline{\mathcal{L}})}}.$$

Cette fonction est paramétrée par la marge d'activation des attributs $m_h^{(\overline{\mathcal{L}})}$, la valeur de la fonction de perte $l_h^{(\overline{\mathcal{L}})}$ au point $m_h^{(\overline{\mathcal{L}})}$ et la pente $s_2^{(\overline{\mathcal{L}})}$ sur l'intervalle $[m_h^{(\overline{\mathcal{L}})}, \mathcal{N}]$. La perte vaut 0 pour $x = 0$, lorsque aucun attribut n'est activé. Cependant, en choisissant une valeur assez faible pour $l_h^{(\overline{\mathcal{L}})}$, nous permettons au modèle d'activer jusqu'à $m_h^{(\overline{\mathcal{L}})}$ attributs pour les autres classes sans être trop pénalisé.

En pratique, la fonction de perte utilisée pour optimiser le nombre d'attributs activés dans la représentation $\boldsymbol{z}$, divisée en K vecteurs d'attributs $\boldsymbol{r}^{(j)}$, est la suivante :

$$\mathcal{L}_{\text{attributs}}(\boldsymbol{z}, y) = \mathcal{L}_{\text{attr}}(A^{(y)}) + \max \mathbb{A}, \tag{6.2}$$

où :

$$\mathbb{A} = \left\{ \overline{\mathcal{L}}_{\text{attr}}(A^{(j)}) : j \in \{1, 2, \ldots, K\} \setminus \{y\} \right\},$$

avec $\mathbb{A}$ l'ensemble des pénalisations du nombre d'attributs pour les autres classes, y l'indice de la classe de l'image d'entrée et $A^{(j)}$ (équation 6.1) est le nombre d'attributs activés dans le vecteur $\boldsymbol{r}^{(j)}$.

6.2.2 Pénalisation de la magnitude des attributs

Le deuxième élément que nous voulons optimiser est la magnitude des attributs. Pour un vecteur d'attributs $\boldsymbol{r}^{(j)}$ représentant la classe j avec $j \in \{1, 2, \ldots, K\}$, la magnitude (norme euclidienne) de ce vecteur est notée $M^{(j)}$ et définie par :

$$M^{(j)} = \sqrt{\sum_{i=1}^{\mathcal{N}} r_i^{(j)2}}. \tag{6.3}$$

La magnitude $M^{(j)}$ est optimisée pour que chaque classe ait une magnitude souhaitée grâce aux fonctions décrites ci-dessous.

Classe y. La fonction $\mathcal{L}_{\text{mgn}}(x)$ pénalise la magnitude x du vecteur d'attributs $\boldsymbol{r}^{(y)}$ associé à la classe y. L'objectif est que ce vecteur ait une magnitude élevée. La fonction est définie sur l'intervalle $[0, \infty[$. La magnitude souhaitée $v^{(\mathcal{L})}$ pour ce vecteur est égale à la magnitude du vecteur dans lequel toutes

les dimensions sont activées avec la valeur α, la valeur d'activation souhaitée sur les dimensions. Par conséquent, $v^{(\mathcal{L})} = \sqrt{\mathcal{N} \cdot \alpha^2}$. La marge $m_b^{(\mathcal{L})}$ indique la variation de la magnitude acceptée en dessous de $v^{(\mathcal{L})}$, et $m_h^{(\mathcal{L})}$ la variation acceptée au-dessus de $v^{(\mathcal{L})}$. L'idée d'avoir une marge inférieure est, ici encore, de permettre à des attributs d'être désactivés dans la représentation. La marge supérieure permet de ne pas pénaliser excessivement si des attributs sont plus fortement identifiés que d'autres. L'intervalle $[0, \infty]$ est donc divisé en quatre sous-intervalles :

$$\mathcal{I}_1 = [0, v^{(\mathcal{L})} - m_b^{(\mathcal{L})}[\,, \qquad \mathcal{I}_2 = [v^{(\mathcal{L})} - m_b^{(\mathcal{L})}, v^{(\mathcal{L})}[\,,$$
$$\mathcal{I}_3 = [v^{(\mathcal{L})}, v^{(\mathcal{L})} + m_h^{(\mathcal{L})}[\,, \qquad \mathcal{I}_4 = [v^{(\mathcal{L})} + m_h^{(\mathcal{L})}, \infty[\,.$$

La fonction $\mathcal{L}_{\text{mgn}}$ est définie sur ces intervalles comme suit :

$$\mathcal{L}_{\text{mgn}}(x) = \begin{cases} -s_1^{(\mathcal{L})} \cdot (x - v^{(\mathcal{L})} + m_b^{(\mathcal{L})}) + l_b^{(\mathcal{L})} & \text{pour } x \in \mathcal{I}_1, \\ -s_2^{(\mathcal{L})} \cdot (x - v^{(\mathcal{L})}) & \text{pour } x \in \mathcal{I}_2, \\ s_3^{(\mathcal{L})} \cdot (x - v^{(\mathcal{L})}) & \text{pour } x \in \mathcal{I}_3, \\ s_4^{(\mathcal{L})} \cdot (x - v^{(\mathcal{L})} - m_h^{(\mathcal{L})}) + l_h^{(\mathcal{L})} & \text{pour } x \in \mathcal{I}_4, \end{cases}$$

avec :

$$s_2 = \frac{l_b^{(\mathcal{L})}}{m_b^{(\mathcal{L})}}, \quad s_3 = \frac{l_h^{(\mathcal{L})}}{m_h^{(\mathcal{L})}}.$$

Cette fonction est paramétrée par les marges $m_b^{(\mathcal{L})}$ et $m_h^{(\mathcal{L})}$, les valeurs de la fonction de perte $l_b^{(\mathcal{L})}$ et $l_h^{(\mathcal{L})}$ aux points $v - m_b$ et $v + m_h$, respectivement, et les pentes $s_1^{(\mathcal{L})}$ et $s_4^{(\mathcal{L})}$ sur les intervalles $[0, v^{(\mathcal{L})} - m_b^{(\mathcal{L})}[$ et $[v^{(\mathcal{L})} + m_h^{(\mathcal{L})}, \infty[$. La perte vaut 0 pour $x = v^{(\mathcal{L})}$, et en choisissant des valeurs faibles pour $l_b^{(\mathcal{L})}$ et $l_h^{(\mathcal{L})}$, nous permettons au modèle de ne pas être fortement pénalisé si la magnitude s'écarte de $v^{(\mathcal{L})}$ selon les marges définies.

Autres classes. La fonction $\overline{\mathcal{L}}_{\text{mgn}}(x)$ pénalise la magnitude des vecteurs d'attributs $\boldsymbol{r}^{(j)}$ avec $j \neq y$ associés aux autres classes. Cette fonction est définie sur l'intervalle $[0, \infty[$. La magnitude souhaitée sur ces vecteurs est $v^{(\overline{\mathcal{L}})} = 0$, car les attributs des autres classes ne doivent pas être activés. La marge $m_h^{(\overline{\mathcal{L}})}$ indique la variation de la magnitude acceptée pour les autres classes afin que la représentation puisse exprimer une sémantique. Dans ce cas aussi, pour pouvoir distinguer le vecteur d'attributs de la classe y des autres vecteurs, il faut que $m_h^{(\overline{\mathcal{L}})} < v^{(\mathcal{L})} - m_b^{(\mathcal{L})}$. L'intervalle $[0, \infty[$ est divisé en 2 sous-intervalles :

$$\mathcal{I}_1 = [0, m_h^{(\overline{\mathcal{L}})}[\quad \text{et} \quad \mathcal{I}_2 = [m_h^{(\overline{\mathcal{L}})}, \infty[\,.$$

La fonction $\overline{\mathcal{L}}_{\text{mgn}}$ est définie comme suit :

$$\overline{\mathcal{L}}_{\text{mgn}}(x) = \begin{cases} s_1^{(\overline{\mathcal{L}})} \cdot x & \text{pour } x \in \mathcal{I}_1, \\ s_2^{(\overline{\mathcal{L}})} \cdot (x - m_h^{(\overline{\mathcal{L}})}) + l_h^{(\overline{\mathcal{L}})} & \text{pour } x \in \mathcal{I}_2, \end{cases}$$

avec :

$$s_1 = \frac{l_h^{(\overline{\mathcal{L}})}}{m_h^{(\overline{\mathcal{L}})}}.$$

Cette fonction est paramétrée par la marge $m_h^{(\overline{\mathcal{L}})}$, la valeur de la fonction de perte $l_h^{(\overline{\mathcal{L}})}$ au point $m_h^{(\overline{\mathcal{L}})}$ et la pente $s_2^{(\overline{\mathcal{L}})}$ sur l'intervalle $[m_h^{(\overline{\mathcal{L}})}, \infty[$. La perte vaut 0 pour $x = 0$, mais en choisissant une valeur faible, de l'ordre de 0.5 pour $l_h^{(\overline{\mathcal{L}})}$, nous permettons au modèle d'avoir une magnitude allant jusqu'à $m_h^{(\overline{\mathcal{L}})}$ pour les vecteurs des autres classes sans que cela soit trop pénalisant.

À partir de ces deux fonctions, la fonction de perte utilisée pour optimiser la magnitude des vecteurs d'attributs $\boldsymbol{r}^{(j)}$ est la suivante :

$$\mathcal{L}_{\text{magnitude}}(\boldsymbol{z}, y) = \mathcal{L}_{\text{mgn}}(M^{(y)}) + \max \mathbb{M}, \tag{6.4}$$

où :

$$\mathbb{M} = \left\{ \overline{\mathcal{L}}_{\text{mgn}}(M^{(j)}) : j \in \{1, 2, \ldots, K\} \setminus \{y\} \right\},$$

avec $\mathbb{M}$ l'ensemble des pénalisations de la magnitude pour les autres classes, y l'indice de classe de l'image d'entrée et $M^{(j)}$ (équation 6.3) est la magnitude du vecteur $\boldsymbol{r}^{(j)}$.

Les quatre fonctions intermédiaires que nous avons définies ont de nombreux hyperparamètres. Pour nos expériences préliminaires, nous avons sélectionné des valeurs qui semblent adéquates, selon notre conception théorique du problème, afin d'obtenir des premiers résultats exploitables. Par ailleurs, les valeurs de certains paramètres sont contraintes, notamment pour garantir la distinction entre la classe à prédire et les autres classes. Toutefois, une analyse plus approfondie devra être menée pour optimiser ces paramètres.

6.3 Expérimentations avec DIFAIR v2

6.3.1 Expériences

Jeux de données. Pour tester cette nouvelle version de DIFAIR, nous utilisons deux jeux de données. Tout d'abord, nous utilisons, comme dans les autres expériences, la séparation 2 du jeu de données CIFAR-10 du

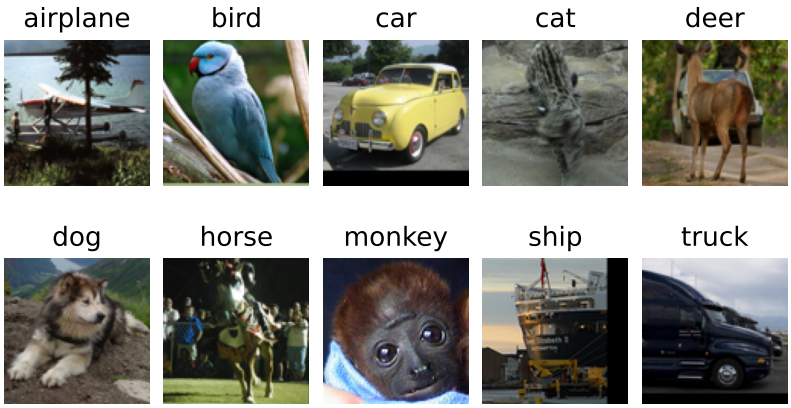


Figure 6.2: **Exemples d'images du jeu de données STL-10.**

benchmark d'OSR de Neal et al. (2018), car celle-ci contient des couples de classes sémantiquement proches. Ensuite, afin de vérifier si des images de meilleure qualité peuvent aider le modèle à apprendre une représentation exprimant une sémantique, nous avons choisi d'utiliser le jeu de données STL-10 (Coates et al., 2011). Ce jeu de données est similaire à CIFAR-10, mais les images sont de taille 96×96 pixels, et sont donc plus détaillées. Les classes sont les mêmes que pour CIFAR-10 à une différence près : la classe *frog* est remplacée par la classe *monkey*. STL-10 contient 13 000 images dont 5 000 pour l'entraînement, 8 000 pour le test et 100 000 images non annotées que nous n'utilisons pas. Il y a donc uniquement 500 images par classe pour l'entraînement, ce qui est peu comparé aux 5 000 images par classe pour CIFAR-10. La figure 6.2 montre un exemple d'image pour chaque classe de STL-10.

Sur le jeu de données STL-10, nous avons séparé les classes connues et inconnues de façon à conserver deux classes sémantiquement proches parmi les classes connues, tandis que les autres classes sémantiquement proches sont séparées. Les classes *cat* et *dog* sont connues (proches), tandis que *airplane* est connu et *bird* inconnu, *car* est connu et *truck* inconnu, *horse* est connu et *deer* inconnu. Les deux classes restantes ne sont pas sémantiquement proches, la classe *monkey* est connue et *ship* inconnu.

Méthodes. Les fonctions $\mathcal{L}_{\text{attributs}}$ (équation 6.1) et $\mathcal{L}_{\text{magnitude}}$ (équation 6.3) sont utilisée pour définir deux méthodes d'entraînement de réseau de neurone. La première méthode utilise les deux fonctions de perte optimisant conjointement le nombre d'attributs activés et la magnitude des vecteurs d'attributs. Cette méthode est nommée « DIFAIR v2 (*double*) ».

Ainsi, la fonction de perte utilisée est la somme des deux fonctions avec un terme de pondération λ :

$$\mathcal{L}_{\text{double}}(\boldsymbol{z}, y) = \lambda \mathcal{L}_{\text{attributs}}(\boldsymbol{z}, y) + \mathcal{L}_{\text{magnitude}}(\boldsymbol{z}, y).$$

Le terme λ est fixé à 1 pour ces expériences préliminaires. La deuxième méthode utilise uniquement la fonction de perte $\mathcal{L}_{\text{magnitude}}$ pour entraîner le réseau, car les premiers résultats ont montré que la contrainte d'activation des attributs est plus difficilement optimisable. Cette méthode est nommée « DIFAIR v2 (*magnitude*) ». Les deux méthodes sont comparées à MLS et à DIFAIR v1 (chapitre 5).

Pour appliquer les modèles entraînés aux tâches d'OSR, le score d'OSR utilisé pour DIFAIR v2 (double) est une combinaison des valeurs mesurées pour chacun des objectifs (attributs et magnitude) :

$$S_{\text{double}}(y \in \mathbb{C} | \boldsymbol{X}) = \max \left\{ A^{(j)} \cdot M^{(j)} : j \in \{1, 2, \ldots, K\} \right\}, \tag{6.5}$$

avec $A^{(j)}$ le nombre d'attributs activés pour la classe j et $M^{(j)}$ la magnitude du vecteur d'attributs de la classe j. Pour DIFAIR v2 (magnitude), le score d'OSR est la magnitude maximale des vecteurs d'attributs :

$$S_{\text{mgn}}(y \in \mathbb{C} | \boldsymbol{X}) = \max \left\{ M^{(j)} : j \in \{1, 2, \ldots, K\} \right\}. \tag{6.6}$$

Le score MOS est aussi appliqué aux représentations obtenues avec les deux méthodes. Les résultats avec le MOS sont nommés DIFAIR v2 (double + MOS) et DIFAIR v2 (magnitude +MOS).

Expériences. Pour ces expériences les paramètres $\mathcal{N} = 10$, $\alpha = 4$ ont été choisis (une recherche d'hyperparamètres devra être menée pour trouver les meilleurs paramètres). Les valeurs choisies pour configurer les fonctions de perte pénalisant le nombre d'attributs activés sont présentées dans le tableau 6.1. Les hyperparamètres des fonctions de perte pénalisant la magnitude des attributs sont présentés dans le tableau 6.2. Chacune des fonctions de perte est tracée dans la figure 6.3 pour les hyperparamètres choisis.

Un modèle est entraîné pour chaque fonction de perte et sur chacun des jeux de données. Le protocole d'entraînement est le protocole amélioré proposé par Vaze et al. (2022) et décrit dans la section 4.1. Une seule exécution est réalisée pour chaque modèle, une recherche plus exhaustive reste donc à réaliser.

Le modèle entraîné avec DIFAIR v1 utilise les hyperparamètres déterminés par la recherche d'hyperparamètres dans la section 4.3, à savoir $\mathcal{N} = 5$ et $\alpha = 10$. La pénalisation de la corrélation des filtres est utilisée pour DIFAIR v1, ce qui n'est pas le cas dans les expériences menées avec DIFAIR v2.

$\mathcal{L}_{\text{attr}}(x)$	$\mathcal{N}$	$m_b^{(\mathcal{L})}$	$l_b^{(\mathcal{L})}$	$s_1^{(\mathcal{L})}$
	10	3	0.5	2

$\overline{\mathcal{L}}_{\text{attr}}(x)$	$m_h^{(\overline{\mathcal{L}})}$	$l_h^{(\overline{\mathcal{L}})}$	$s_2^{(\overline{\mathcal{L}})}$
	2	0.5	2

TABLEAU 6.1: **Hyperparamètres des fonctions de perte optimisant le nombre d'attributs activés.**

$\mathcal{L}_{\text{mgn}}$	$v^{(\mathcal{L})}$	$m_b^{(\mathcal{L})}$	$m_h^{(\mathcal{L})}$	$l_b^{(\mathcal{L})}$	$l_h^{(\mathcal{L})}$	$s_1^{(\mathcal{L})}$	$s_4^{(\mathcal{L})}$
	$\sqrt{\mathcal{N} \cdot \alpha^2}$	3	2	0.5	0.5	2	1

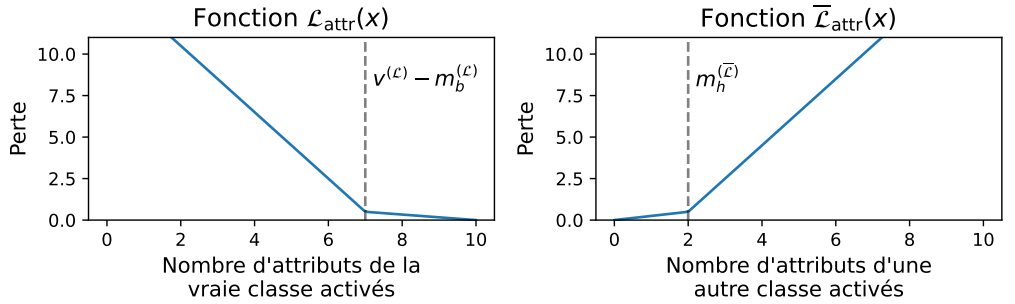
$\overline{\mathcal{L}}_{\text{mgn}}$	$m_h^{(\overline{\mathcal{L}})}$	$l_h^{(\overline{\mathcal{L}})}$	$s_2^{(\overline{\mathcal{L}})}$
	$\sqrt{2 \cdot \alpha^2}$	1	2

TABLEAU 6.2: **Hyperparamètres des fonctions de perte optimisant la magnitude des attributs.**

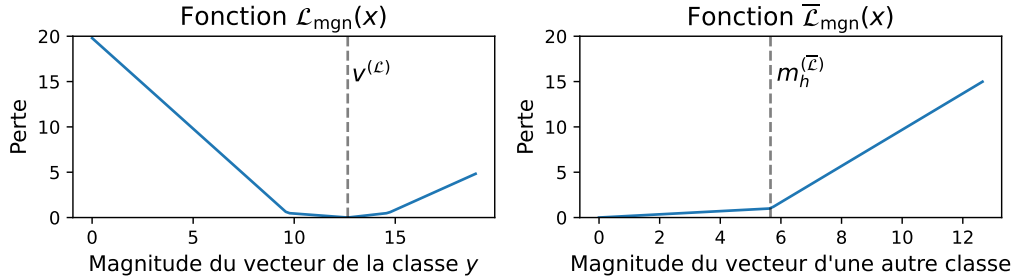
6.3.2 Résultats en OSR et discussion

Les résultats obtenus pour les différentes méthodes sur les jeux de données CIFAR-10 et STL-10 sont présentés dans le tableau 6.3. Sur les deux jeux de données, l'Exactitude en *Closed-Set* (ECS) mesurée pour les nouvelles versions de DIFAIR v2 est proche de celle obtenues avec MLS et DIFAIR v1. Les performances sont supérieures sur CIFAR-10, mais il s'agit d'un unique entraînement pour chaque modèle, il est donc difficile de conclure à un réel avantage de DIFAIR v2. Les différences sur les scores AUROC sont plus prononcées que sur l'ECS. En particulier, les nouvelles versions de DIFAIR restent moins performantes que le modèle MLS. Nous pouvons noter que seule la version DIFAIR v2 (double + MOS) obtient de meilleures performances que DIFAIR v1 sur CIFAR-10. Sur les données STL-10, le modèle DIFAIR v2 (magnitude) est meilleure que DIFAIR v1.

L'application du MOS sur les représentations obtenues avec DIFAIR v2 permet d'améliorer l'AUROC sur CIFAR-10, mais la dégrade sur STL-10. Ce comportement est difficile à expliquer, car il est à l'opposé de ce qui



(a) Fonction de perte pour le nombre d'attributs activés.



(b) Fonction de perte pour la magnitude des vecteurs d'attributs.

FIGURE 6.3: **Tracé des fonctions de perte pour les attributs activés et la magnitude des vecteurs d'attributs.** Les fonctions de perte sont paramétrées par les valeurs présentées dans les tableaux 6.1 et 6.2 pour les fonctions de perte pénalisant le nombre d'attributs activés et la magnitude des vecteurs d'attributs, respectivement.

a été observé avec DIFAIR v1 sur TinyImageNet. Sur TinyImageNet, le MOS a permis d'améliorer les résultats (suivant l'hypothèse que, le modèle n'ayant pas totalement convergé, le MOS permet de mieux exploiter les représentations obtenues). Dans le cas de DIFAIR v2, les résultats sur STL-10 suggèrent qu'utiliser la magnitude ou le nombre d'attributs activés est plus avantageux qu'utiliser uniquement la valeur maximale de la sortie du modèle.

Remarquons tout de même que le modèle DIFAIR v2 (double) n'est pas du tout capable de détecter les classes inconnues sur CIFAR-10. L'AUROC étant de 49.3%, déterminer aléatoirement si la classe est inconnue donnerait de meilleurs résultats. Pour ce modèle l'objectif d'activation des attributs n'a pas été correctement optimisé. Or, comme le score d'OSR est une multiplication du nombre d'attributs activés et de la magnitude, une mauvaise optimisation du nombre d'attributs activés peut impliquer des scores d'OSR

	CIFAR10		STL10	
Méthode	ECS(%)	AUROC(%)	ECS(%)	AUROC(%)
MLS	93.8	**92.3**	*84.8*	**72.4**
DIFAIR v1	94.1	83.9	83.5	62.0
DIFAIR v1 (MOS)	94.1	82.9	83.5	69.2
DIFAIR v2 (double)	**94.5**	49.3	83.5	67.4
DIFAIR v2 (double + MOS)	**94.5**	*86.4*	83.5	63.3
DIFAIR v2 (magnitude)	*94.4*	80.5	**85.7**	*69.9*
DIFAIR v2 (magnitude + MOS)	*94.4*	83.4	**85.7**	67.3

TABLEAU 6.3: **Résultats d'OSR sur CIFAR10 et STL10**. L'ECS(%) et le score AUROC(%) pour la tâche de détection des classes inconnues sont reportés pour une seule exécution sur les jeux de données étudiés. Les résultats proviennent tous de nos expériences. L'ECS est la même pour les modèles et leur version MOS, car c'est le même modèle qui est utilisé, seul le score d'OSR change. Les meilleurs résultats sont indiqués en gras, les seconds en italique.

très mal distribués. En effet, une image où un seul attribut est activé avec une forte magnitude implique que le score d'OSR vaut la magnitude. Cependant, si deux attributs sont activés pour une autre image, avec une magnitude deux fois plus faible, alors les scores d'OSR sont égaux et la tâche de déterminer si la classe est inconnue devient complexe. Dans cette situation, le MOS amène à la conclusion que la première image est connue, car un attribut fortement activé signifie que des caractéristiques ont été reconnues par le modèle. Le fait que ce problème n'ait pas été rencontré sur SLT-10 suggère que les images de meilleure qualité ont permis au modèle d'extraire des attributs plus pertinents, activés de manière plus équilibrée.

6.3.3 Analyse des modèles

Les modèles DIFAIR v2 (double) et DIFAIR v2 (magnitude) sont analysés sur CIFAR-10 et STL-10 pour étudier à quel point les caractéristiques souhaitées pour les représentations de DIFAIR sont présentes et déterminer les avantages et limites de tels modèles. Nous commençons par analyser en détails le modèle DIFAIR v2 (double) entraîné sur CIFAR-10, utilisant un score d'OSR basé sur le nombre d'attributs et la magnitude, pour lequel l'AUROC obtenue est proche de l'aléatoire.

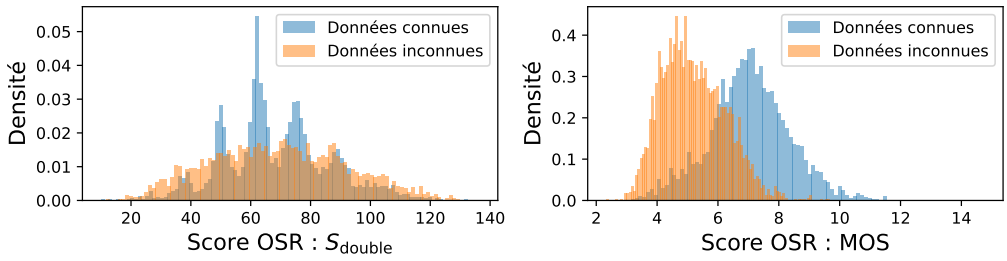


FIGURE 6.4: **Distributions des scores d'OSR obtenus sur CIFAR-10 avec DIFAIR v2 (double).**

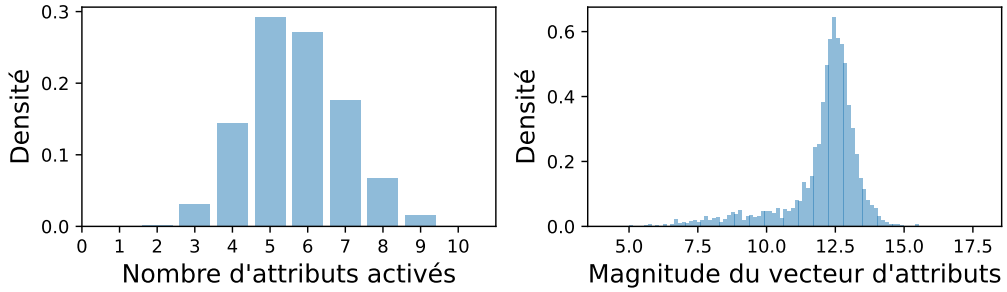


FIGURE 6.5: **Distributions du nombre d'attributs activés et de la magnitude des vecteurs d'attributs utilisés pour calculer le score d'OSR S_{double} pour le modèle DIFAIR v2 (double).**

DIFAIR v2 (double) - CIFAR-10. Le score d'OSR S_{double} (équation 6.5), basé sur le nombre d'attributs et la magnitude, ne permet pas de détecter les classes inconnues de CIFAR-10, l'AUROC obtenue étant de 49.3%. Pour comprendre pourquoi ce score ne permet pas de détecter les classes inconnues, nous avons analysé les distributions des scores obtenus pour les données connues et inconnues. La figure 6.4 montre ces distributions. En utilisant ce score, les deux distributions sont superposées, il est donc impossible de déterminer un seuil de décision pour séparer les classes connues des classes inconnues. La distribution des scores pour les données connues présente plusieurs pics séparés de manière régulière. Ces pics correspondent à un nombre d'attributs activés qui varie, multiplié par une magnitude stable. La distribution des scores en utilisant le MOS présentée dans la figure 6.4 exhibe un chevauchement moins prononcé entre les scores des classes connues et inconnues, ce qui permet d'obtenir une AUROC de 86.4%.

La figure 6.5 montre les distributions du nombre d'attributs activés et de la magnitude des vecteurs d'attributs qui ont été utilisés pour calculer le score S_{double}. La distribution de la magnitude est centrée autour de 12.5, ce qui correspond à la magnitude souhaitée pour les vecteurs d'attributs de la classe à prédire. Cependant, le nombre d'attributs activés est très variable. Cela signifie que certains attributs sont bien plus activés que d'autres, puisque la même magnitude est atteinte avec moins d'attributs activés, et que le score d'OSR dépend presque uniquement du nombre d'attributs activés. De toute évidence, ce comportement ne permet pas d'utiliser le score d'OSR combinant le nombre d'attributs activés et la magnitude pour détecter les classes inconnues. Une observation intéressante est qu'il n'y a jamais de cas où tous les attributs d'une même classe sont activés : il y a en moyenne cinq ou six attributs activés par classe. Si pour différentes images d'une classe ce ne sont pas les mêmes attributs qui sont activés, cela signifie que le modèle est capable d'extraire des attributs pertinents et sémantiquement représentatifs de la classe. Il ne s'agit pas d'attributs représentant des caractéristiques globales qui sont toujours présentes pour une classe, comme nous l'avons observé avec DIFAIR v1.

Bien que l'optimisation semble poser problème, car la même magnitude est toujours obtenue indépendamment du nombre d'attributs, nous étudions tout de même la représentation apprise afin de vérifier si elle contient de la sémantique. La figure 6.6 montre la distribution des activations de chacun des attributs pour la classe *cat*, à partir des données qui ont été correctement prédites. Ce graphique indique que seuls cinq attributs en moyenne sont activés avec des valeurs supérieures à $0.5 \cdot \alpha$, alors que l'objectif d'optimisation était d'en activer au moins 7. On note également qu'un attribut de la classe chat n'est jamais activé, et que d'autres sont uniquement faiblement activés. Ce comportement est aussi observé sur les distributions d'activations d'autres classes. Il est possible que $\mathcal{N} = 10$ soit une valeur trop élevée pour une application de DIFAIR sur CIFAR-10, où les images sont de faible qualité : le modèle peine à extraire autant d'attributs pertinents que demandés. Au vu de ces résultats, le modèle ne semble pas extraire de sémantique, les attributs de la classe *dog* étant tous désactivés pour une image de la classe *cat* à l'exception de quelques valeurs extrêmes. Comparé aux activations des autres classes, rien ne laisse penser qu'il puisse y avoir une proximité entre ces deux classes.

La représentation d'une image de la classe inconnue *airplane* est présentée dans la figure 6.7. Cette représentation d'une classe inconnue se rapproche de celle que nous souhaitons obtenir avec DIFAIR pour plusieurs raisons. Tout d'abord, des attributs sont activés ou peu activés *sur plu-*

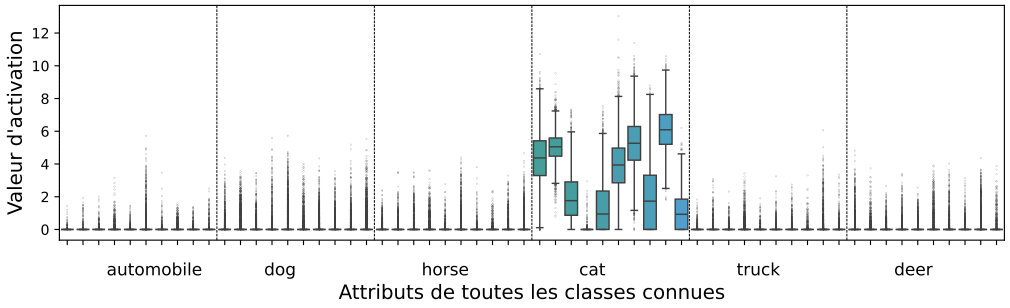


FIGURE 6.6: **Distribution des activations des attributs pour la classe *cat* sur CIFAR-10 pour le modèle DIFAIR v2 (double).** Les points noirs en dehors des boites à moustache sont des valeurs extrêmes par rapport à la distribution des valeurs. Seules les données correctement prédites sont utilisées dans ce graphique.

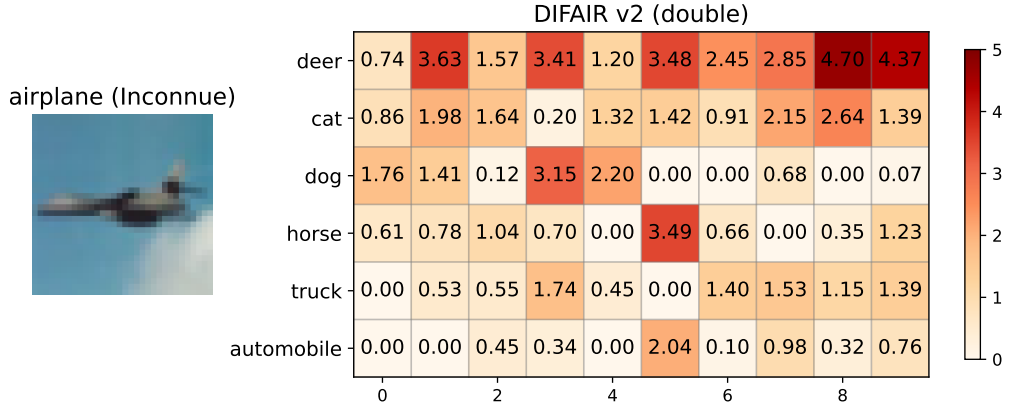


FIGURE 6.7: **Représentation de l'activation des attributs pour une image de la classe *airplane* (inconnue) sur CIFAR-10 avec DIFAIR v2 (double).**

sieurs classes, indiquant que le modèle a des difficultés à prédire une classe avec certitude. Ensuite, contrairement à DIFAIR v1, nous n'observons pas d'activation de tous les attributs d'une classe, ceux-ci semblent être plus indépendants. Par contre, certains attributs sont tout de même activés avec une valeur assez élevée, supérieure à α, face à une image d'une classe inconnue. L'obtention d'une telle représentation pour une classe inconnue nous a poussé à évaluer notre modèle avec un autre score d'OSR : la somme de tous les attributs de la représentation. Ainsi, lorsque le modèle est incertain et active des attributs d'un grand nombre de classes, le score d'OSR est élevé, indiquant une donnée probablement inconnue. Il se trouve qu'ef-

fectivement, ce score permet d'améliorer l'AUROC de 1.8% pour atteindre 88.2% par rapport aux 86.4% obtenus avec le MOS, mais ce résultat reste inférieur à l'AUROC de 92.3% obtenue avec le MLS.

Dans les représentations de DIFAIR v0, nous avions observé une corrélation des filtres de la dernière couche de convolution associés à une même classe ; corrélation qui est problématique, car elle empêche l'extraction d'attributs distincts. Par conséquent, nous réalisons aussi cette analyse pour DIFAIR v2. Pour le modèle DIFAIR v2 (double), l'étude de la corrélation des filtres de la dernière couche indique que certains filtres d'une même classe sont corrélés. La corrélation est tout de même plus faible qu'avec la première version de DIFAIR v0 et tous les filtres ne sont pas corrélés. Néanmoins, il pourrait être utile de pénaliser à nouveau la corrélation, comme cela a été fait avec DIFAIR v1. Il existe aussi des corrélations entre les filtres de plusieurs classes différentes, ce qui pourrait indiquer l'extraction d'informations similaires pour ces classes et donc une possible proximité sémantique. Pourtant, la distribution des activations dans les représentations de la classe *chat* (figure 6.6) ne semble pas indiquer une telle sémantique, puisque seuls les attributs de cette classe sont activés. La corrélation des filtres d'une même classe est peut-être aussi un indicateur du fait qu'il y a trop d'attributs à extraire pour décrire ces classes.

DIFAIR v2 (double) - STL-10. Le modèle DIFAIR v2 (double) n'a pas permis d'obtenir un résultat satisfaisant sur CIFAR-10 avec le score d'OSR S_{double} (équation 6.5), l'AUROC obtenue étant de 49.3%. Les résultats sur le jeu de données STL-10 indiquent cependant que ce problème n'a pas été rencontré, l'AUROC mesurée étant de 67.4%. La figure 6.8 présente les distributions du score S_{double} sur STL-10, pour les données connues et inconnues. Ces distributions sont moins superposées que dans le cas de CIFAR-10 (figure 6.4). Cependant, le chevauchement des distributions reste important, pour le score S_{double} comme pour le MOS, ce qui explique que l'AUROC reste basse. La distribution du score S_{double} pour les données connues ne présente pas les pics caractéristiques de la distribution de score observée sur CIFAR-10 (figure 6.4), ce qui indique que le score est moins dépendant du nombre d'attributs activés.

La distribution du nombre d'attributs activés et de la magnitude des vecteurs d'attributs utilisés pour calculer le score d'OSR présentée dans la figure 6.9 indique que la magnitude est plus largement répartie entre 7 et 15, soit sur l'intervalle autorisé par la fonction de perte (« autorisé » au sens où la pénalité est moindre). En pratique, cette magnitude prend toutefois

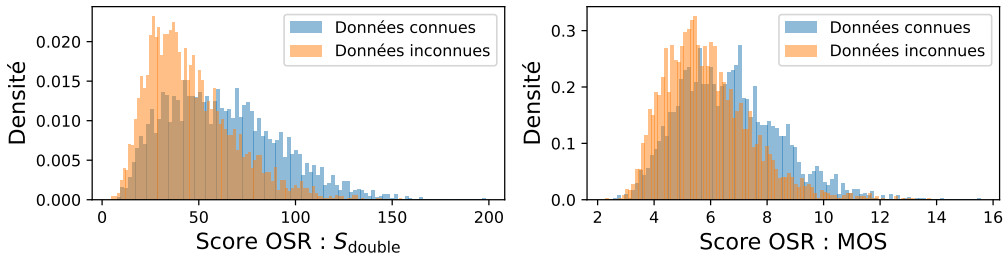


FIGURE 6.8: **Distributions des scores d'OSR obtenus sur STL-10 avec DIFAIR v2 (double).**

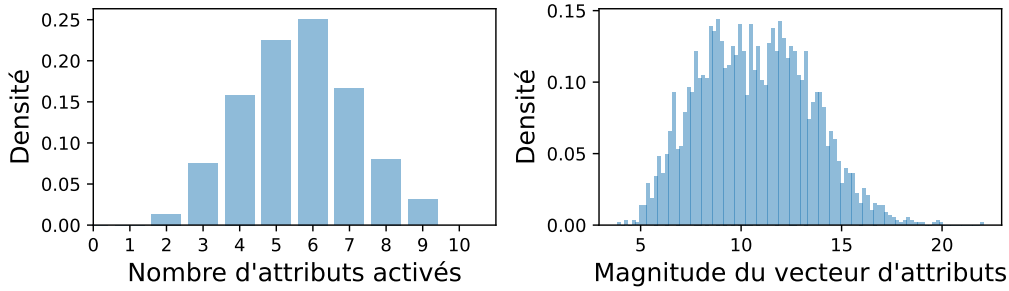


FIGURE 6.9: **Distributions du nombre d'attributs activés et de la magnitude des vecteurs d'attributs utilisés pour calculer le score d'OSR pour le modèle DIFAIR v2 (double).**

des valeurs en deçà de la borne inférieure que nous autorisons, celle-ci étant autour de 10 (tableau 6.2). Le nombre d'attributs activés est similaire à celui observé sur CIFAR-10, avec majoritairement 5 ou 6 attributs activés.

Étudions maintenant la sémantique exprimée par la représentation. Dans la séparation des classes utilisée avec le jeu de données STL-10, seules les classes *cat* et *dog* sont sémantiquement proches et toutes deux connues. La figure 6.10 montre la distribution des activations des attributs pour la classe *cat*. Les attributs de la classe *cat* sont activés, 8 attributs sur 10 sont en moyenne activés avec une valeur supérieure à $0.5 \cdot \alpha$. Il semble qu'avec des images de plus haute qualité, le modèle soit capable d'extraire plus d'attributs : fixer à 10 le nombre d'attributs à apprendre semble être une valeur correcte, car il y a peu d'attributs qui sont toujours désactivés. Cependant, cette représentation montre qu'il y a une légère activation des attributs des autres classes, bien que celle-ci reste inférieure à $0.5 \cdot \alpha$. Sur CIFAR-10 les attributs des autres classes étaient totalement désactivés,

ce qui n'est pas forcément souhaitable non plus. Nous en déduisons que l'objectif permettant d'activer des attributs des autres classes doit être mieux formalisé.

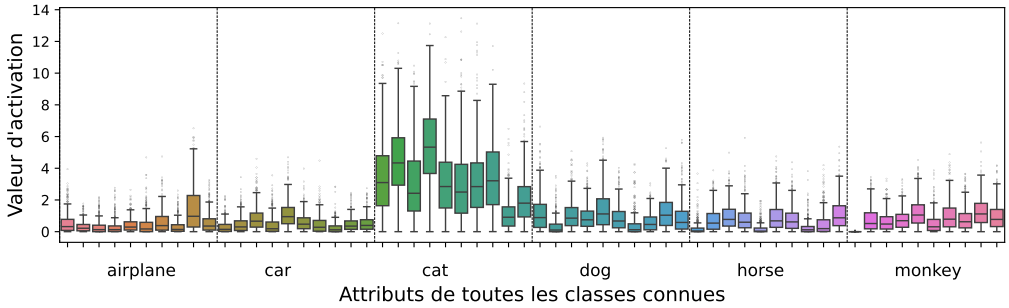


Figure 6.10: **Distribution des activations des attributs pour la classe *cat* sur STL-10 pour le modèle DIFAIR v2 (double).** Les points noirs en dehors des boites à moustache sont des valeurs extrêmes par rapport à la distribution des valeurs. Seules les données correctement prédites sont utilisées pour ces distributions.

Ici aussi, il est difficile de déterminer si les attributs de la classe *dog* s'activent lorsque l'image est de la classe *cat*, étant donné que tous les attributs d'autres classes sont aussi activés dans une certaine mesure. Cependant, en calculant la représentation moyenne de chaque classe (le centre des représentations de chaque classe) et en mesurant la distance euclidienne entre chacun de ces centres, nous obtenons une matrice de distances : cette matrice de distances nous permet de réaliser une classification hiérarchique des classes, et le dendrogramme (section 3.4.3) sur la figure 6.11 exprime la proximité sémantique des classes. Sur cette figure, les animaux sont regroupés à gauche et les véhicules à droite. Les animaux les plus proches sont *dog* et *monkey* puis vient la classe *cat*. Bien que les animaux soient regroupés dans le dendrogramme, la distance entre des classes proches comme *cat* et *dog* reste tout de même élevée, presque autant que la distance entre ces classes et les autres animaux, ce qui suggère un potentiel manque de sémantique dans la représentation.

La représentation d'une image de la classe *cat*, une classe connue, est présentée dans la figure 6.12. Tout d'abord, remarquons grâce à l'image que dans le jeu de données STL-10, une image peut contenir plusieurs classes. La représentation obtenue montre que le modèle a détecté la classe *cat* et la classe *car* présente en arrière-plan.

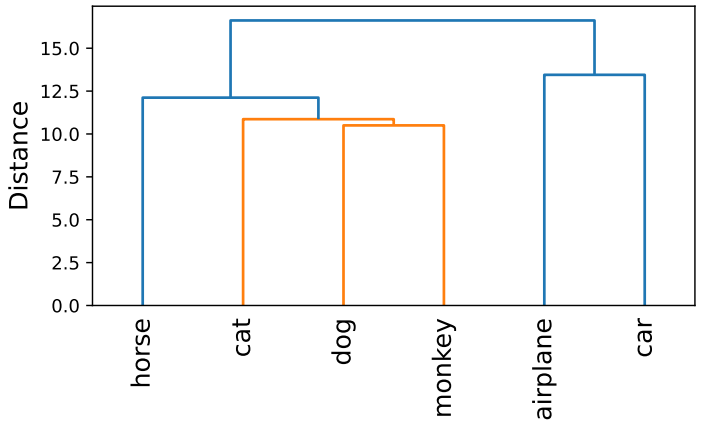


FIGURE 6.11: **Dendrogramme représentant une classification hiérarchique des classes de STL-10 à partir des représentations moyennes du modèle DIFAIR v2 (double).**

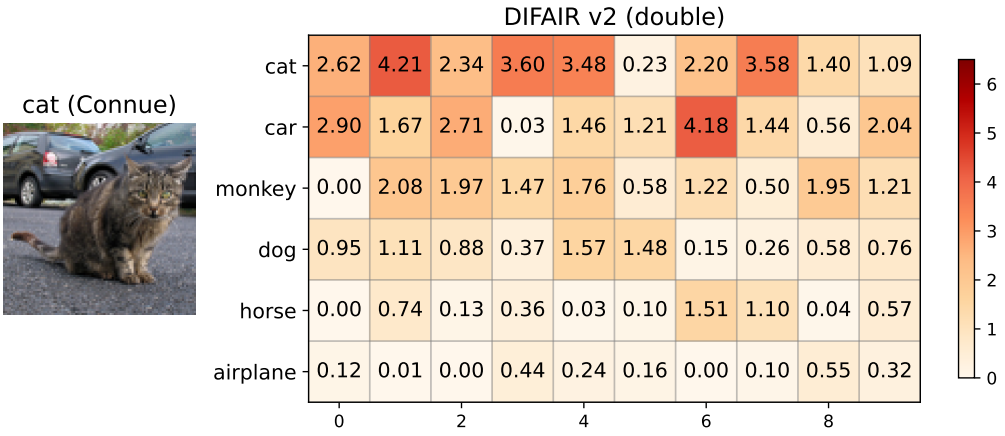


FIGURE 6.12: **Représentation de l'activation des attributs pour une image de la classe *cat* (connue) sur STL-10 avec DIFAIR v2 (double).**

Nous observons également que des attributs de la classe *monkey* et *dog* sont détectés avec une valeur supérieure à 1, soit 25% de l'activation cible α. Selon le critère, fixé heuristiquement, pour le calcul du nombre d'attributs activés (équation 6.1), où un attribut est considéré activé si sa valeur est supérieure à $0.5 \cdot \alpha$, seul un attribut de la classe *monkey* est activé. Cependant, s'agit-il d'activations liées à une sémantique apprise par le réseau ? Les animaux ont forcément des caractéristiques communes, par exemple des yeux, et l'activation de certains attributs pour la plupart des animaux

correspond peut-être à un comportement normal et inévitable, dans une représentation sémantique. Cela se produit peut-être aussi dans les représentations apprises par les réseaux de neurones entraînés avec l'entropie croisée, mais dans ce cas, les attributs n'étant pas explicitement associés à chaque classe, contrairement à DIFAIR, cette réflexion est plus complexe.

Pour DIFAIR v2 (double) sur STL-10, nous avons constaté que l'utilisation de la somme de tous les attributs, comme nous l'avons essayé sur CIFAR-10, ne constitue pas un score d'OSR adéquat. Trop d'attributs s'activent pour les autres classes en plus de ceux de la classe prédite, même face à des données connues. Puisque utiliser le nombre d'attributs activés directement dans le score d'OSR peut poser problème, comme l'ont montré les résultats sur CIFAR-10, nous avons entraîné un modèle avec la fonction de perte pénalisant la magnitude uniquement : le modèle est nommé DIFAIR v2 (magnitude) et utilise le score d'OSR S_{mgn} (équation 6.6).

DIFAIR v2 (magnitude) - STL-10. Pour terminer, nous étudions la version DIFAIR v2 (magnitude) sur STL-10, car ce modèle a bien fonctionné sur CIFAR-10, contrairement à la version DIFAIR v2 (double). Les distributions des scores d'OSR obtenus avec ce modèle sont présentées dans la figure 6.13. Le score d'OSR S_{mgn} (équation 6.6) correspond à la magnitude maximale des vecteurs d'attributs. Les distributions des scores pour les classes connues et inconnues sont mieux séparées en utilisant le score S_{mgn} que le MOS, ce qui est reflété par l'AUROC mesurée. Cependant, on note un important chevauchement des distributions.

Bien que l'objectif contraignant l'activation d'un minimum d'attributs ne soit plus utilisé, la distribution présentée dans la figure 6.14 montre qu'il y a tout de même plusieurs attributs activés, majoritairement 5. Cela reste inférieur aux résultats obtenus avec DIFAIR v2 (double), pour lequel la moyenne du nombre d'attributs activés était autour de 6. Ce nombre n'est pas nécessairement problématique, car sans optimiser l'objectif du nombre d'attributs activés, le réseau aurait pu activer un seul attribut pour obtenir la magnitude souhaitée. La magnitude des attributs est majoritairement comprise entre 7 et 15, comme pour DIFAIR v2 (double). Cela montre qu'il existe des représentations dont la magnitude n'est pas assez élevée, la valeur minimale acceptable étant autour de 10, selon la marge définie dans la fonction de perte (tableau 6.2.)

La figure 6.15 présente la distribution des activations des attributs pour la classe *cat*. En moyenne, seuls quatre attributs s'activent en moyenne au-dessus de $0.5 \cdot \alpha$, tandis que deux attributs de la classe chat restent tou-

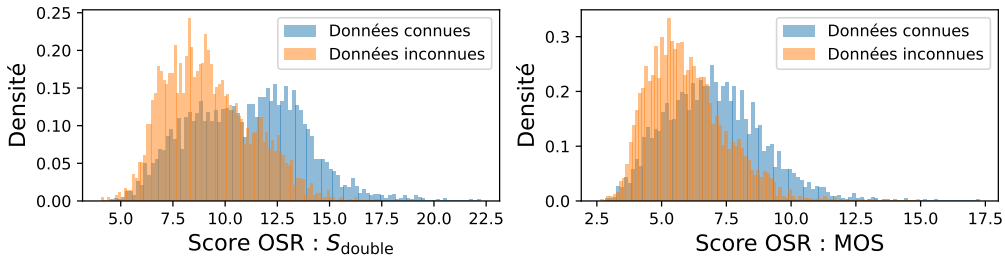


FIGURE 6.13: **Distributions des scores d'OSR obtenus sur STL-10 avec DIFAIR v2 (magnitude).**

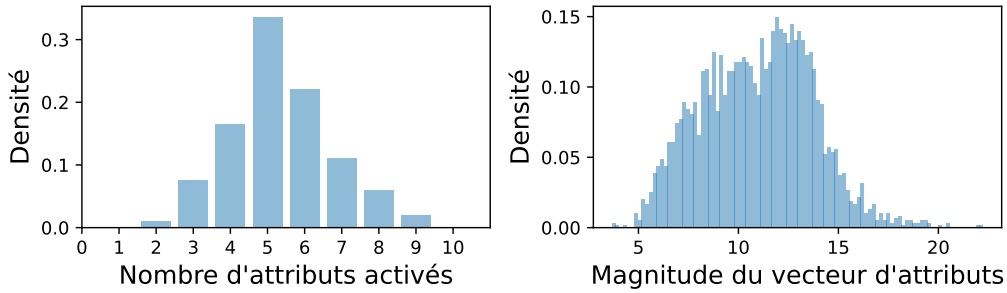


FIGURE 6.14: **Distributions du nombre d'attributs activés et de la magnitude des vecteurs d'attributs utilisés pour calculer le score d'OSR pour le modèle DIFAIR v2 (magnitude).**

jours désactivés : borner les valeurs d'activation sur les dimensions pourrait permettre que le calcul de la magnitude soit dépendant de l'ensemble des attributs sans être influencé par des attributs activés avec une valeur trop importante. Comme précédemment, il est difficile d'observer une activation spécifique des attributs de la classe *dog* pour une image de la classe *cat*. En mesurant la distance entre les centres moyens calculés pour chaque classe, le centre de la classe *cat* est d'ailleurs plus proche de la classe *monkey* que de la classe *dog*. L'expression de la sémantique est plus réduite dans cette représentation que dans celle obtenue avec DIFAIR v2 (double) sur STL-10.

La représentation de la même image de la classe *cat* que celle étudiée pour DIFAIR v2 (double) est présentée dans la figure 6.16. Cette représentation montre que moins d'attributs de la classe chat sont activés qu'avec DIFAIR v2 (double) (figure 6.12) : cela est certainement dû à l'absence de l'objectif sur le nombre d'attributs à activer. La plupart des attributs ont des valeurs inférieures à α, compensées dans le calcul de la magnitude par un attribut activé avec une valeur de 25% supérieure à α. L'objectif, en péna-

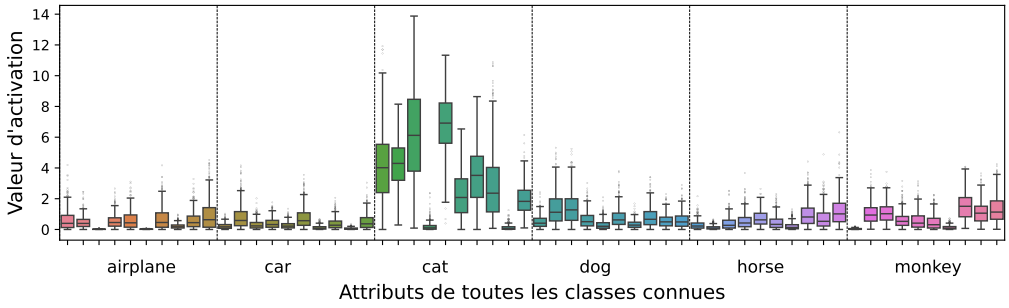


Figure 6.15: **Distribution des activations des attributs pour la classe *cat* sur STL-10 pour le modèle DIFAIR v2 (magnitude).** Les points noirs en dehors des boites à moustache sont des valeurs extrêmes par rapport à la distribution des valeurs. Seules les données correctement prédites sont utilisées pour ces distributions.

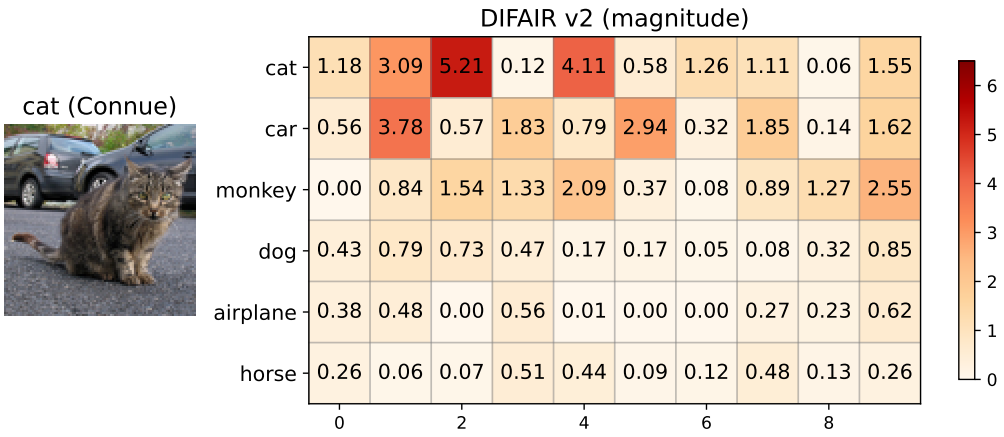


Figure 6.16: **Représentation de l'activation des attributs pour une image de la classe *cat* (connue) sur STL-10 avec DIFAIR v2 (magnitude).**

lisant uniquement la magnitude était d'obtenir plusieurs attributs activés, puisque la magnitude souhaitée correspond à l'activation de plusieurs attributs. L'objectif sur l'activation des attributs reste peut-être utile, même s'il est optimisé plus difficilement. Il semble préférable d'éviter d'utiliser le nombre d'attributs activés comme facteur multiplicatif dans le calcul du score d'OSR, car cela risque de fausser la distribution des scores, comme cela a été observé sur CIFAR-10 avec le modèle DIFAIR v2 (double).

Pour DIFAIR v2 (magnitude), nous avons également étudié la corrélation des filtres de la dernière couche de convolution. Ces filtres exhibent des corrélations plus élevées entre les filtres associés à une même classe qu'entre les filtres associés à d'autres classes. Ce n'est par contre pas le cas de toutes les classes : certaines ont moins de filtres corrélés que les autres. Dans tous les cas, ajouter le terme de pénalisation de la corrélation des filtres d'une même classe permettrait sûrement d'améliorer les résultats obtenus.

L'analyse des nouvelles versions DIFAIR v2 montre des résultats mitigés, mais intéressants en regard de notre problématique, en gardant à l'esprit que les paramètres ont été choisis arbitrairement et qu'un seul entraînement a été réalisé poru chaque tâche. L'optimisation des hyperparamètres ainsi que la combinaison des objectifs d'apprentissage et des méthodes de calcul de score d'OSR sont des points à approfondir. La sémantique reste difficilement observable dans les représentations obtenues : aucune activation claire d'attributs d'autres classes n'est observée, potentiellement à cause du paramétrage des fonctions de perte.

6.4 Conclusions sur DIFAIR v2 et perspectives

Les résultats présentés ont été obtenus avec une version préliminaire de DIFAIR v2, sans recherche des hyperparamètres les plus adaptés pour les fonctions de perte et sans un nombre suffisant d'exécutions pour tirer de solides conclusions. Nous estimons cependant que les résultats obtenus sont encourageants : l'ECS mesurée est au même niveau que la *baseline*, et l'AUROC est équivalente à celle obtenue avec la version DIFAIR v1. La méthode proposée est capable d'activer les attributs de la classe correspondant à la vérité terrain et de désactiver les attributs des autres classes. De plus, le comportement problématique de DIFAIR v1, où tous les attributs d'une même classe étaient activés quelle que soit l'image, n'est plus observé. Au contraire, pour une même classe, selon l'image, des attributs différents peuvent s'activer et avec des valeurs différentes. C'est un comportement que nous souhaitons observer : les images étant différentes les unes des autres, ce ne sont pas toujours les mêmes caractéristiques de la classe qui sont visibles.

Concernant la sémantique des représentations, les résultats sont mitigés et parfois difficilement interprétables. La sémantique apparente de DIFAIR v2 (double) sur le jeu de données STL-10 est-elle liée à un problème d'optimisation ou ce comportement émerge-t-il du modèle ? Car ce comportement

n'a pas été observé sur CIFAR-10, où tous les attributs de classes autres que la classe y sont généralement désactivés (avec une valeur proche de 0). Cela pourrait-il être dû à la meilleure qualité des images de STL-10, qui permettrait au modèle d'extraire davantage d'attributs pour représenter chaque classe ? Sur STL-10 des attributs ont été faiblement activés pour toutes les classes autres que la classe y, sans pour autant que l'activation de classes sémantiquement proches se démarque. Une étude plus approfondie sur plus de jeux de données est nécessaire pour déterminer que DIFAIR v2 est réellement capable d'extraire des attributs sémantiques.

Puisque pour une image, des attributs s'activent faiblement pour toutes les classes, notamment sur le jeu de données STL-10, cela pose aussi un problème d'interprétabilité de la représentation : il est impossible de dire si un attribut est activé parce qu'il est commun à plusieurs classes ou non.

Avec les fonctions de perte et les scores d'OSR proposés, de nombreuses combinaisons de ces éléments sont encore possibles pour proposer de nouvelles approches. Par exemple, le score d'OSR utilisé devrait aussi tenir compte des activations des autres classes et pas uniquement de celle reconnue, comme nous l'avons expérimenté sur CIFAR-10 avec les représentations du modèle DIFAIR v2 (double). Pour ces représentations, la somme de toutes les activations a été utilisée en tant que score d'OSR, ce qui a permis d'améliorer l'AUROC. L'utilité d'un tel score est de prendre en compte les situations où des attributs sont activés dans de multiples classes, signifiant que le modèle est probablement incertain. De même, si les vecteurs de représentation de deux classes ont des magnitudes très proches, le réseau est probablement incertain par rapport à la classe à prédire, ou bien plusieurs classes ont été reconnues dans l'image.

Les hyperparamètres de base de DIFAIR, comme le nombre d'attributs $\mathcal{N}$ et la valeur cible sur les dimensions α, doivent aussi être étudiés. Les quelques expériences proposées montrent que $\mathcal{N} = 10$ semble trop élevé pour CIFAR-10, mais adapté à STL-10. La valeur α choisie est souvent dépassée, ce qui signifie que la magnitude de la représentation est donc plus influencée par certains attributs que d'autres. Il faudrait étudier le comportement du modèle en augmentant cette valeur : serait-elle toujours atteinte et dépassée ? Une alternative serait de borner l'activation des attributs pour rendre la magnitude représentative de l'activation de l'ensemble des attributs.

Inspirée par les résultats obtenus sur DIFAIR v0, la corrélation des filtres associés à une même classe a été étudiée. Pour les nouvelles approches DIFAIR v2, la distribution des coefficients de corrélation se rapproche de

celle d'un modèle entraîné avec entropie croisée, bien que certains filtres d'une même classe expriment malgré tout une corrélation élevée. Le terme de perte pénalisant la corrélation des filtres utilisé dans DIFAIR v1, pourrait être réintroduit pour tenter d'améliorer l'indépendance des attributs extraits et les résultats.

En conclusion, de nombreux essais ont été réalisés pour tenter d'obtenir une représentation exhibant un maximum de spécificités de la méthode DIFAIR décrite dans le chapitre 3. Plusieurs problèmes ont pu être résolus, mais de nouveaux sont apparus. Cependant, au cours de ce processus, notre compréhension du comportement des modèles a été améliorée grâce à des analyses en profondeur de ces modèles et des représentations apprises. Cela a permis de raffiner les objectifs d'apprentissage, et les scores d'OSR utilisés, afin de progresser dans la recherche de représentations différenciées et leur application au domaine de l'OSR.

Conclusion

Dans ce travail de thèse, nous nous sommes intéressés à la détection de données de classes inconnues à l'aide de réseaux de neurones convolutifs appliqués à la classification d'images. Classiquement, dans les problèmes de classification d'images, les modèles intègrent une tête de classification qui calcule une distribution de probabilités sur les K classes apprises durant l'entraînement. Ainsi, si une donnée n'appartenant pas à ces K classes est par la suite fournie au modèle, celui-ci réalisera tout de même une prédiction (erronée) de la classe de cette donnée. Ce comportement est problématique dans des situations réelles, sujettes à une certaine variabilité ou à l'apparition de nouveautés, en particulier lorsque le domaine d'application peut être considéré comme critique, par exemple en santé ou encore en conduite autonome de véhicules.

D'une part, pour répondre à ce problème de détection de l'inconnu, nous nous sommes tournés vers le domaine de l'*Open-Set Recognition* (OSR), un paradigme d'apprentissage automatique dans lequel les modèles sont évalués non seulement sur des données dont la classe est connue, mais aussi sur des données dont la classe est inconnue. Dans un tel contexte, il est attendu des modèles qu'ils parviennent à déterminer qu'une donnée d'entrée est de classe inconnue, et de prédire correctement la classe de la donnée dans le cas contraire.

D'autre part, nous avons également souhaité considérer la nature critique de certaines applications des systèmes d'intelligence artificielle : dans de telles applications, il est souhaitable que le modèle et ses résultats puissent être interprétés, au moins dans une certaine mesure, par l'utilisateur. Sans nécessairement plonger profondément dans le domaine de l'intelligence artificielle explicable (*eXplainable Artificial Intelligence*,

XAI), nous avons toutefois tenu à proposer une approche donnant lieu à une forme d'interprétabilité, tant des résultats du modèle sur lequel nous greffons notre approche, que du modèle en lui-même.

Ainsi, et en gardant à l'esprit les contraintes d'interprétabilité que nous souhaitons mettre en œuvre, le problème de la détection de données de classe inconnue nous a amenés à nous interroger sur la possibilité d'exploiter *directement la représentation latente* apprise par un modèle, pour détecter les situations dans lesquelles celui-ci fait face à des données de classe inconnue. Nous avons alors proposé l'approche DIFAIR (*DIFferentiAted Images Representations*), consistant à appliquer les contraintes d'optimisation sur la représentation latente elle-même.

Notre étude de l'état de l'art du domaine de l'OSR nous a permis de mettre en lumière deux idées saillantes à exploiter pour aller vers la réalisation de notre objectif : d'une part, de bonnes performances en détection de classes inconnues peuvent être atteintes en entraînant un modèle uniquement sur des données dont la classe est connue ; d'autre part, l'extraction d'attributs suffisamment représentatifs de chaque classe connue améliore la capacité du modèle à détecter des données de classe inconnue, *via l'absence* de détection de ces attributs. L'approche DIFAIR se positionne au croisement de ces idées, et l'une de nos principales contributions réside dans l'ajout de contraintes sur l'apprentissage de la représentation latente (c'est-à-dire des attributs descriptifs des classes connues) visant à rendre celle-ci plus interprétable en associant chacune des dimensions de la représentation à une classe.

Dans ce document, nous avons exposé trois versions de l'approche DIFAIR : toutes sont conçues dans l'optique de permettre la détection de données de classes inconnues à partir de la représentation latente apprise par le modèle. Les deux premières versions de DIFAIR (DIFAIR v0 et DIFAIR v1, chapitres 4 et 5) visaient à exploiter la représentation sous forme d'un *vecteur*. Des ancres, équidistantes dans l'espace latent, sont fixées *a priori* : chacune des ancres dénote le centre des sous-espaces latents dans lesquels nous souhaitons contraindre la représentation de chacune des classes. Ainsi, les instances peuvent être représentées à une proximité plus ou moins grande de l'ancre de leur classe tout en étant circonscrites au sous-espace latent de cette classe, dans l'idée qu'une instance peut partager des attributs communs à plusieurs classes, signalant alors une proximité sémantique avec ces différentes classes.

L'espace de représentation ainsi construit a permis la détection de données appartenant à des classes inconnues, en mesurant la distance de la représentation de l'instance aux ancres. Une instance éloignée de toutes les ancres au-delà d'un certain seuil est alors détectée comme appartenant à une classe inconnue. Cependant, bien que nous soyons parvenus à exploiter l'espace latent pour détecter des données de classe inconnues, les performances de ces deux versions en OSR sont inférieures à celles des performances des approches de l'état de l'art. Malgré tout, DIFAIR v1 atteint des performances équivalentes en classification des données de classe connue, indiquant alors que la représentation extraite n'impose pas un compromis entre la classification des données de classes connues et la détection des données de classes inconnues.

L'analyse des représentations construites par ces modèles par différents outils de visualisation a révélé que celles-ci ne répondaient cependant pas à nos attentes, notamment en raison de l'absence de sémantique dans la représentation.

Les bonnes performances de DIFAIR en classification de données de classes connues confirment que c'est bien la « qualité » des attributs extraits qui fait défaut pour détecter les données de classes inconnues. Lorsque notre objectif est de représenter chaque attribut sur une dimension distincte de l'espace latent, et de n'activer ces dimensions qu'en présence de la classe à laquelle elles sont associées, DIFAIR v0 répond à ces contraintes en activant l'ensemble des attributs d'une classe avec des valeurs similaires. Cependant, les contraintes d'interprétabilité inhérentes à cette représentation nous ont permis d'identifier ce problème. Notons ici, sans surprise, qu'il est bien entendu plus aisé de déterminer que le modèle n'exhibe pas les propriétés que nous en attendons, que de déterminer ce qu'il calcule et apprend exactement.

Nous nous sommes ainsi confrontés à la difficulté de formaliser l'objectif de notre modèle dans nos fonctions de perte : ces fonctions ont été optimisées d'une manière que nous n'avions pas anticipée ; cette optimisation est de plus très triviale, ainsi que DIFAIR v0 nous l'a montré, en provoquant la convergence des filtres de la couche de convolution ajoutée dans notre approche pour atteindre l'objectif imposé. Cette convergence des filtres permet d'atteindre l'objectif, car elle implique l'extraction d'attributs dupliqués pour une classe donnée, plaçant plus aisément l'instance à proximité de son ancre, alors qu'avec DIFAIR nous souhaitons extraire

des attributs distincts décrivant les classes. Une contrainte supplémentaire a alors été apportée dans la fonction de perte de DIFAIR v1, de façon à prévenir l'extraction d'attributs dupliqués pour une classe.

Étant donné que des images d'une même classe d'objets peuvent présenter de la variabilité dans les caractéristiques qu'elles présentent (en cas d'occlusion d'une portion de l'objet, si le point de vue est différent, *etc.*), il serait souhaitable que les attributs représentant les caractéristiques non présentes dans l'image soient inactifs, mais ce n'est pas ce que nous avons observé. Avec DIFAIR v2, nous avons esquissé des pistes de réflexion pour raffiner la formalisation de notre fonction de perte. Les objectifs d'apprentissage ont été repensés et les fonctions de pertes reformulées de manière à les rendre à la fois plus spécifiques à des sous-objectifs précis tout en permettant une optimisation plus relâchée par l'acceptation d'un ensemble de solutions satisfaisant un objectif donné plutôt que d'attendre une solution unique.

Nous pouvons nous interroger sur la pertinence de notre approche pour caractériser ce que devrait être une *représentation sémantique* des données, et comment celle-ci peut être extraite par le modèle.

Les différentes versions de DIFAIR que nous avons proposées ont toutes rencontré des difficultés à construire une représentation sémantique. Une perspective consisterait à introduire des connaissances sémantiques *a priori* dans le modèle, de façon à guider l'apprentissage de la représentation. Ces connaissances pourraient être fournies sous forme d'ontologies ou de graphes de connaissances, exprimant des relations sémantiques entre les différentes classes. L'idée que nous proposons est alors d'associer les dimensions de la représentation latente aux classes en fonction de ces relations. Actuellement, les approches DIFAIR v0 et v1 associent $\mathcal{N}$ dimensions distinctes à chaque classe et se basent sur l'hypothèse que le modèle sera capable, de lui-même, d'activer une dimension commune aux deux classes. Le modèle n'étant pas contraint à extraire des attributs communs, il peut se focaliser sur l'extraction d'attributs discriminants et ignorer les attributs communs, qui sont porteurs de la sémantique des classes et qui permettraient d'extraire des informations sur des données inconnues. Or, grâce à l'apport de connaissances *a priori* indiquant que deux classes sont proches, les ancres utilisées par DIFAIR pourraient directement présenter une dimension commune à ces deux classes : ces dernières seraient donc représentées avec $\mathcal{N}+1$ dimensions, et la dimension supplémentaire est commune aux deux représentations, avec l'objectif de l'activer sur les deux classes. Pour donner un exemple, les ancres des classes chat et chien, deux classes sémantiquement

proches, pourraient être définies ainsi avec $\mathcal{N} = 2 : \mathcal{A}^{(\text{chat})} = (\alpha, \alpha, \alpha, 0, 0)$ et $\mathcal{A}^{(\text{chien})} = (0, 0, \alpha, \alpha, \alpha)$, où α est la valeur à activer sur chaque dimension. Sur les ancres présentées, la troisième dimension est commune et doit être activées pour les deux classes : la sémantique est donc intégrée dans le problème d'optimisation.

Si l'apport de connaissances au modèle permet effectivement d'incorporer la sémantique que nous recherchons à la représentation construite par DIFAIR, nous pourrions positionner notre approche dans un cadre plus large d'apprentissage semi-supervisé : une donnée de classe inconnue peut malgré tout partager certains attributs de classes connues, indiquant alors une possible proximité sémantique de cette donnée avec la ou les classes dont elle partage des attributs. Cette hypothèse pourrait être testée dans un contexte de *propagation de labels*, où l'on cherche à assigner une classe à des données non étiquetées. De surcroît, la distance aux ancres pourrait être utilisée pour déterminer un degré de certitude quant à la classe assignée.

Les difficultés que nous avons éprouvées pour parvenir à détecter des données de classes inconnues ne doivent cependant pas occulter un apport prégnant de DIFAIR, à savoir sa capacité à réaliser des prédictions correctes sur des données dont la classe est connue en utilisant la représentation latente contrainte que nous avons imposé par notre processus d'optimisation. Cette représentation présente l'avantage d'être interprétable sans nécessiter le recours à des approches tierces classiquement utilisées en XAI pour « expliquer » les résultats d'un modèle. Ces approches, qualifiées d'approches *post-hoc*, représentent des modèles dont on veut expliquer les résultats, et différentes approches peuvent donner lieu à différentes explications pour un seul et même résultat. Parvenir à contraindre un réseau de neurones profond à construire une représentation intrinsèquement interprétable permettrait de se passer de modèle proxy et ainsi des limites épistémiques qui leur sont propres : ce sont là aussi des perspectives que nous continuerons à examiner.

Bibliographie

(Arrieta et al., 2020) Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable Artificial Intelligence (XAI) : Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information fusion*, 58 :82–115, 2020.

(BBC, 2019) BBC. How 'smart' email could change the way we talk, 2019. URL `https://www.bbc.com/future/article/20190812-how-ai-powered-predictive-text-affects-your-brain`. Consulté le 04/10/2024.

(Bendale & Boult, 2015) Abhijit Bendale and Terrance Boult. Towards Open Set Deep Networks, Novembre 2015.

(Blundell et al., 2015) Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pp. 1613–1622. PMLR, 2015.

(Bremner et al., 1994) Frederick J. Bremner, Stephen J. Gotts, and Dina L. Denham. Hinton diagrams : Viewing connection strengths in neural networks. *Behavior Research Methods, Instruments, & Computers*, 26 (2) :215–218, Juin 1994. ISSN 0743-3808, 1532-5970. doi : 10.3758/BF03204624.

(Cevikalp et al., 2023) Hakan Cevikalp, Bedirhan Uzun, Yusuf Salk, Hasan Saribas, and Okan Köpüklü. From anomaly detection to open set recognition : Bridging the gap. *Pattern Recognition*, 138 :109385, Juin 2023. ISSN 00313203. doi : 10.1016/j.patcog.2023.109385.

(Chen et al., 2021) Guangyao Chen, Peixi Peng, Xiangqian Wang, and Yonghong Tian. Adversarial Reciprocal Points Learning for Open Set Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021. ISSN 0162-8828, 2160-9292, 1939-3539. doi : 10.1109/TPAMI.2021.3106743.

(Coates et al., 2011) Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 215–223. JMLR Workshop and Conference Proceedings, 2011.

(Cubuk et al., 2020) Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment : Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 702–703, 2020.

(Deng et al., 2009) Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet : A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255. Ieee, 2009.

(Dietterich & Guyer, 2022) Thomas G. Dietterich and Alexander Guyer. The Familiarity Hypothesis : Explaining the Behavior of Deep Open Set Methods. *Pattern Recognition*, 132 :108931, Décembre 2022. ISSN 00313203. doi : 10.1016/j.patcog.2022.108931.

(Dosovitskiy et al., 2021a) Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words : Transformers for Image Recognition at Scale. *arXiv :2010.11929 [cs]*, Juin 2021a.

(Dosovitskiy et al., 2021b) Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al.

An image is worth 16x16 words : Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021b.

(Gal & Ghahramani, 2016) Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation : Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pp. 1050–1059. PMLR, 2016.

(Ge et al., 2017) ZongYuan Ge, Sergey Demyanov, Zetao Chen, and Rahil Garnavi. Generative OpenMax for Multi-Class Open Set Classification, Juillet 2017.

(Geng et al., 2021) Chuanxing Geng, Sheng-jun Huang, and Songcan Chen. Recent Advances in Open Set Recognition : A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(10) :3614–3631, Octobre 2021. ISSN 0162-8828, 2160-9292, 1939-3539. doi : 10.1109/TPAMI.2020.2981604.

(Géron, 2019) Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow. Concepts, Tools, and Techniques to Build Intelligent Systems*, volume 1. O'Reilly Media, 2019.

(Goodfellow et al., 2015) Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

(Google, 2024) Google. Google assistant, 2024. URL `https://assistant.google.com/`. Consulté le 04/10/2024.

(Guo et al., 2017) Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pp. 1321–1330. PMLR, 2017.

(He et al., 2016) Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

(Hinton, 1984) Geoffrey E Hinton. Distributed representations. 1984.

(Hinton & Roweis, 2002) Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15, 2002.

(Hinton & Shallice, 1991) Geoffrey E Hinton and Tim Shallice. Lesioning an attractor network : Investigations of acquired dyslexia. *Psychological review*, 98(1) :74, 1991.

(Hornik et al., 1989) Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5) :359–366, 1989.

(Hubel, 1959) David H Hubel. Single unit activity in striate cortex of unrestrained cats. *The Journal of physiology*, 147(2) :226, 1959.

(Hubel et al., 1959) David H Hubel, Torsten N Wiesel, et al. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3) :574–591, 1959.

(Ioffe & Szegedy, 2015) Sergey Ioffe and Christian 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, pp. 448–456, Lille, France, 2015. JMLR.org.

(Jumper et al., 2021) John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *nature*, 596 (7873) :583–589, 2021.

(Knudsen et al., 2024) J Everett Knudsen, Umar Ghaffar, Runzhuo Ma, and Andrew J Hung. Clinical applications of artificial intelligence in robotic surgery. *Journal of Robotic Surgery*, 18(1) :102, 2024.

(Kotz & Nadarajah, 2000) Samuel Kotz and Saralees Nadarajah. *Extreme Value Distributions : Theory and Applications*. world scientific, 2000.

(Krizhevsky, 2009) Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. 2009.

(Kullback & Leibler, 1951) Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22 (1) :79–86, 1951.

(Kuncheva & Bezdek, 1998) Ludmila I Kuncheva and James C Bezdek. Nearest prototype classification : Clustering, genetic algorithms, or random search ? *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 28(1) :160–164, 1998.

(Lake et al., 2015) Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266) :1332–1338, 2015. doi : 10.1126/science.aab3050.

(Lakshminarayanan et al., 2017) Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

(Le & Yang, 2015) Ya Le and Xuan Yang. Tiny ImageNet Visual Recognition Challenge. 2015.

(LeCun et al., 1998) Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998.

(LeCun et al., 2010) Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available : http ://yann.lecun.com/exdb/mnist*, 2, 2010.

(Lin et al., 2014) Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *ICLR*, 2014.

(Liu et al., 2020) Yuan Liu, Ayush Jain, Clara Eng, David H Way, Kang Lee, Peggy Bui, Kimberly Kanada, Guilherme de Oliveira Marinho, Jessica Gallegos, Sara Gabriele, et al. A deep learning system for differential diagnosis of skin diseases. *Nature medicine*, 26(6) :900–908, 2020.

(Loshchilov & Hutter, 2016) Ilya Loshchilov and Frank Hutter. SGDR : Stochastic gradient descent with warm restarts. Août 2016.

(Maas et al., 2013) Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. Icml*, volume 30, pp. 3. Atlanta, GA, 2013.

(Macrae, 2022) Carl Macrae. Learning from the failure of autonomous and intelligent systems : Accidents, safety, and sociotechnical sources of risk. *Risk Analysis*, 42(9) :1999–2025, 2022. doi : 10.1111/risa.13850.

(Martín Abadi et al., 2015) Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia,

Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow : Large-scale machine learning on heterogeneous systems, 2015.

(MidJourney, 2024) MidJourney. MidJourney, 2024. URL `https://www.midjourney.com/home`. Consulté le 04/10/2024.

(Miller et al., 2021) Dimity Miller, Niko Sünderhauf, Michael Milford, and Feras Dayoub. Class Anchor Clustering : A Loss for Distance-based Open Set Recognition, Mars 2021.

(Nair & Hinton, 2010) Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814, 2010.

(Narkhede et al., 2022) Meenal V Narkhede, Prashant P Bartakke, and Mukul S Sutaone. A review on weight initialization strategies for neural networks. *Artificial intelligence review*, 55(1) :291–322, 2022.

(Neal et al., 2018) Lawrence Neal, Matthew Olson, Xiaoli Fern, Weng-Keen Wong, and Fuxin Li. Open Set Learning with Counterfactual Images. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (eds.), *Computer Vision – ECCV 2018*, volume 11210, pp. 620–635. Springer International Publishing, Cham, 2018. ISBN 978-3-030-01230-4 978-3-030-01231-1. doi : 10.1007/978-3-030-01231-1_38.

(Netzer et al., 2011) Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading Digits in Natural Images with Unsupervised Feature Learning. 2011.

(Nguyen et al., 2015) Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled : High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 427–436, Boston, MA, USA, Juin 2015. IEEE. ISBN 978-1-4673-6964-0. doi : 10.1109/CVPR.2015.7298640.

(NTSB, 2019) National Transportation Safety Board NTSB. Collision between vehicle controlled by developmental automated driving system and pedestrian. Highway Accident Report, NTSB, 2019.

(OpenAI, 2024) OpenAI. ChatGPT : A conversational agent by OpenAI, 2024. URL `https://chatgpt.com/`. Consulté le 04/10/2024.

(Ovadia et al., 2019) Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can you trust your model's uncertainty ? evaluating predictive uncertainty under dataset shift. *Advances in neural information processing systems*, 32, 2019.

(Oza & Patel, 2019) Poojan Oza and Vishal M. Patel. C2AE : Class Conditioned Auto-Encoder for Open-Set Recognition. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2302–2311, Long Beach, CA, USA, Juin 2019. IEEE. ISBN 978-1-72813-293-8. doi : 10.1109/CVPR.2019.00241.

(Pang et al., 2021) Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection : A review. *ACM computing surveys (CSUR)*, 54(2) :1–38, 2021.

(Paszke et al., 2019) Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch : An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

(Pearson, 1901) Karl Pearson. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11) :559–572, 1901. doi : 10.1080/14786440109462720.

(Perez et al., 2018) Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film : Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

(Pimentel et al., 2014) Marco AF Pimentel, David A Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal processing*, 99 :215–249, 2014.

(Ridnik et al., 2021) Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, and Lihi Zelnik-Manor. Imagenet-21k pretraining for the masses. *arXiv preprint arXiv :2104.10972*, 2021.

(Rosenblatt, 1958) Frank Rosenblatt. The perceptron : A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6) :386, 1958.

(Rumelhart et al., 1986) David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986. *Biometrika*, 71(599-607) :6, 1986.

(Russell, 2019) Stuart J. Russell. *Human Compatible : Artificial Intelligence and the Problem of Control*. Viking, [New York, New York ?], 2019. ISBN 978-0-525-55861-3 0-525-55861-6 978-0-525-55863-7 0-525-55863-2.

(Salehi et al., 2022) Mohammadreza Salehi, Hossein Mirzaei, Dan Hendrycks, Yixuan Li, Mohammad Hossein Rohban, and Mohammad Sabokrou. A Unified Survey on Anomaly, Novelty, Open-Set, and Out-of-Distribution Detection : Solutions and Future Challenges, Décembre 2022.

(Scheirer et al., 2011) W J Scheirer, A Rocha, R J Micheals, and T E Boult. Meta-Recognition : The Theory and Practice of Recognition Score Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8) :1689–1695, Août 2011. ISSN 0162-8828, 2160-9292. doi : 10.1109/TPAMI.2011.54.

(Scheirer et al., 2012) Walter J Scheirer, Anderson Rocha, Archana Sapkota, and Terrance E Boult. Towards Open Set Recognition. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 2012.

(Schrage, 2020) Michael Schrage. *Recommendation Engines*. MIT press, 2020.

(Shannon, 1948) Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3) :379–423, 1948.

(Simonyan & Zisserman, 2015) K Simonyan and A Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations (ICLR 2015)*. Computational and Biological Learning Society, 2015.

(Srivastava et al., 2014) Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout : A simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1) :1929–1958, 2014.

(Szegedy et al., 2016) Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

(Thoma, 2017) Martin Thoma. The HASYv2 dataset, Janvier 2017.

(Torralba et al., 2008) Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images : A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11) :1958–1970, 2008.

(Tosh & Ruxton, 2010) Colin R Tosh and Graeme D Ruxton. *Modelling Perception with Artificial Neural Networks*. Cambridge University Press, 2010.

(Van der Maaten & Hinton, 2008) Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

(Vaze et al., 2022) Sagar Vaze, Kai Han, Andrea Vedaldi, and Andrew Zisserman. Open-Set Recognition : A Good Closed-Set Classifier is All You Need ?, Avril 2022.

(Wen et al., 2016) Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A Discriminative Feature Learning Approach for Deep Face Recognition. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (eds.), *Computer Vision – ECCV 2016*, volume 9911, pp. 499–515. Springer International Publishing, Cham, 2016. ISBN 978-3-319-46477-0 978-3-319-46478-7. doi : 10.1007/978-3-319-46478-7_31.

(Xu et al., 2023) Baile Xu, Furao Shen, and Jian Zhao. Contrastive Open Set Recognition. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(9) :10546–10556, Juin 2023. ISSN 2374-3468, 2159-5399. doi : 10.1609/aaai.v37i9.26253.

(Yoshihashi et al., 2019) Ryota Yoshihashi, Wen Shao, Rei Kawakami, Shaodi You, Makoto Iida, and Takeshi Naemura. Classification-Reconstruction Learning for Open-Set Recognition, Octobre 2019.

(Yurtsever et al., 2020) Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving : Common practices and emerging technologies. *IEEE access : practical innovations, open solutions*, 8 :58443–58469, 2020.

(Zhang et al., 2020) Hongjie Zhang, Ang Li, Jie Guo, and Yanwen Guo. Hybrid Models for Open Set Recognition. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (eds.), *Computer Vision – ECCV 2020*, volume 12348, pp. 102–117. Springer International Publishing, Cham, 2020. ISBN 978-3-030-58579-2 978-3-030-58580-8. doi : 10.1007/978-3-030-58580-8_7.

(Zhang et al., 2018) Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. Mixup : Beyond Empirical Risk Minimization, Avril 2018.

(Zhou et al., 2016) Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning Deep Features for Discriminative Localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2921–2929, Las Vegas, NV, USA, Juin 2016. IEEE. ISBN 978-1-4673-8851-1. doi : 10.1109/CVPR.2016.319.

(Zhou, 2020) Ding-Xuan Zhou. Universality of deep convolutional neural networks. *Applied and computational harmonic analysis*, 48(2) :787–794, 2020.

Quentin CHRISTOFFEL

Apprentissage de représentations différenciées dans des modèles d'apprentissage profond : détection de classes inconnues et interprétabilité

Résumé

L'apprentissage profond, et en particulier les réseaux de neurones convolutifs, a révolutionné de nombreux domaines tels que la vision par ordinateur. Cependant, ces modèles restent limités lorsqu'ils rencontrent des données issues de classes inconnues (jamais vues durant l'entraînement) et souffrent souvent d'un manque d'interprétabilité. Nous avons proposé une méthode visant à optimiser directement l'espace de représentation appris par le modèle. Chaque dimension de la représentation est associée à une classe connue. Une dimension doit être activée avec une certaine valeur lorsque le modèle fait face à la classe associée, donc lorsque certaines caractéristiques ont été détectées dans l'image. Cela permet au modèle de détecter les données inconnues par leur représentation distincte des données connues, puisqu'elles ne doivent pas partager les mêmes caractéristiques. Notre approche favorise également des rapprochements sémantiques dans l'espace de représentation en allouant un sous-espace à chaque classe connue. De plus, une certaine interprétabilité est possible en analysant les dimensions activées pour une image donnée, permettant de comprendre quels attributs de quelle classe sont détectés. Cette thèse détaille le développement et l'évaluation de notre méthode à travers plusieurs versions, chacune visant à améliorer les performances et à adresser des limites identifiées grâce à l'interprétabilité, telles que la corrélation des attributs extraits. Les résultats obtenus sur un benchmark de détection de classes inconnues montrent une amélioration notable des performances entre nos différentes versions, bien que présentant des résultats inférieurs à l'état de l'art.

Mots clés : Apprentissage profond, Réseaux de neurones convolutifs, Détection de classes inconnues (Open-Set Recognition), Interprétabilité, Apprentissage de représentation

Abstract

Deep learning, and particularly convolutional neural networks, has revolutionized numerous fields such as computer vision. However, these models remain limited when encountering data from unknown classes (never seen during training) and often suffer from a lack of interpretability. We proposed a method aimed at directly optimizing the representation space learned by the model. Each dimension of the representation is associated with a known class. A dimension is activated with a specific value when the model faces the associated class, meaning that certain features have been detected in the image. This allows the model to detect unknown data by their distinct representation from known data, as they should not share the same features. Our approach also promotes semantic relationships within the representation space by allocating a subspace to each known class. Moreover, a degree of interpretability is achieved by analysing the activated dimensions for a given image, enabling an understanding of which features of which class are detected. This thesis details the development and evaluation of our method across multiple iterations, each aimed at improving performance and addressing identified limitations through interpretability, such as the correlation of extracted features. The results obtained on an unknown class detection benchmark show a notable improvement in performance between our versions, although they remain below the state-of-the-art.

Keywords: Deep Learning, Convolutional Neural Networks, Unknown classes detection (Open-Set Recognition), Intepretability, Representation Learning