

République Algérienne Démocratique et Populaire
الجمهوريّة الجزائريّة الديمُقراطِيَّة الشعُوبِيَّة
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي



المدرسة العليا للعلوم الالى
(المعهد الوطني للتكوين في الاعلام الالى سابقا)
Ecole Supérieure d'Informatique
ex INI (Institut National de formation en Informatique)

Mémoire de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'Etat en Informatique

OPTION : Systèmes Informatiques

Thème

**Proposition et implémentation d'une
nouvelle solution permettant l'explication
des résultats des réseaux de neurones
artificiels.**

Mémoire de projet de fin d'études

Réalisé par :
TOUIL Mohamed Ameziane

Encadré par :
DR MEZIANI Lila
DR H. RUDIN Stuart

Soutenu le : 15/02/2021

Devant le jury composé de :
KHELIFATI Si Larabi
GUERROUT Elhachemi
AYAD Khadidja

Promotion : 2019/2020

Remerciement

Je remercie ALLAH de m'avoir donné la force, le courage, et la patience pour accomplir ce travail.

Je tiens à remercier Dr. Lila MEZIANI mon encadrante à l'École Supérieure d'Informatique pour son suivi continu, son accompagnement, sa patience, sa disponibilité et surtout ses judicieux conseils, qui ont contribué à alimenter ma réflexion et m'orienter dans la réalisation de mon travail.

Je remercie les membres de jury pour l'intérêt qu'ils portent à ce travail et d'avoir accepté de l'examiner et de le juger.

Je tiens à remercier également Mme. AIT ALI YAHIA Dahbia, pour tous ses efforts afin d'assurer le bon déroulement des stages de fin d'étude, et pour sa compréhension et sa gentillesse.

Pour terminer je témoigne mes reconnaissances et ma profonde gratitude envers toute personne qui a contribué de loin ou de près à la réalisation de ce travail.

Dédicace

Ma chère tente Eldjida défunte je commence ce dédicace par priant à dieu à ce qu'il t'accueille en son vaste paradis. Je te remercie infiniment pour tout l'amour que t'as porté pour moi et toutes les fois ou t'étais à mes cotés depuis que j'étais petit, ce qui a fortement contribué à cette réussite. Je te dédie ce travail en première place et te dis à dieu ma chère deuxième maman.

Ma chère maman Djamila, Mon chère papa Amara, je vous dédie ce travail comme preuve de reconnaissance à tout ce que vous m'avez offerts, à tout votre soutiens, et à tous vos sacrifices pour m'offrir les moyens et le confort dont j'avais besoin pour bien mener mes études, toute cette réussite n'est qu'un simple résultat de l'amour que vous portez pour moi et je vous en remercie tellement.

Mon chère Frère Billel, je te dédie ce travail pour te remercier pour tous tes mots profonds, tes encouragements, ton soutiens, tes conseils, et ton amour, je te remercie tellement pour ta présence et je ne cesserai de te rendre encore et encore fier.

Mes deux chères soeurs Kahina et Chahinaz, le symbole d'amour et de douceur de notre petite chère famille, je vous dédie cette réalisation pour vous remercier à tout ce que vous m'avez offerts comme soutiens, amour et confiance, je vous en remercie fort.

Mes deux beaux frères Moussa et Juba, vous étiez amis, frères, parents, conseillers, coachs, et soutiens, je vous dédie ce travail pour vous remercier de tout ce que m'avez offerts et qui m'a beaucoup aider.

Mes petits Anges Anna, Ismail et Islam, je vous dédie aussi ce travail et je vous aime beaucoup.

Mes Amis, et frères, Abdennour, Belkacem, Abdeslam et Nacer, je vous dédie ce travail et je vous remercie tellement pour nos moments passés durant ces cinq années, pour nos efforts partagés, nos soutiens réciproques, et je vous souhaite toute la réussite que vous méritiez fort.

A tout autre membre de famille, tout ami, proche de moi et de mon coeur dont je n'ai pas cité le nom, je vous dédie ce modeste travail.

Résumé

Les réseaux de neurones artificiels (RNA) sont de plus en plus utilisés dans différents domaines pour résoudre des problèmes d'apprentissage automatique. Ils font partie des systèmes nommés "les systèmes à boîtes noires", est ceci car leurs logique de fonctionnement interne est floue face à l'utilisateur.

Faire confiance à un système à boîte noire lorsqu'il traite un problème de décision critique, telle que détecter une fraude bancaire, ou décider qu'une cellule est cancéreuse, est une tâche très difficile. Plusieurs travaux visent à trouver des solutions qui expliquent les résultats obtenus par les systèmes à boîtes noires, ou concevoir des modèles interprétables qui accompagnent une explication aux prédictions qu'ils trouvent. Cette explication peut être une interprétation locale de la prédition, qui vise à nous faire comprendre pourquoi notre RNA a donné une telle décision. Elle peut être post-hoc, qui signifie que cette explication est présentée par un autre modèle qui accompagne notre RNA. Ce modèle explicatif est dit spécifique si il vise à expliquer un type de modèle spécifique et ne peut pas être généralisé sur d'autres.

Dans ce projet de fin d'étude nous avons proposé une solution qui nous permet d'expliquer les résultats obtenus par les réseaux de neurones convolutifs (CNN), en ce basant sur le transfert learning pour l'extraction d'une carte de caractéristiques d'une image, la transférer à un MLP pour obtenir une prédition, et l'utiliser par un KNN qui nous donnera une explication visuelle de la prédition à la base de k plus proches voisins à notre requête. Ce qui nous permettra d'interpréter d'une façon visuelle la prédition d'un RNA, et comprendre sur quelle bases de la requête nous avons obtenus une telle prédition.

Mots clés : Systèmes à Boîtes Noires, l'Intelligence Artificielle Explicable, Apprentissage Profonds, Systèmes Jumeaux, Réseaux de Neurones Convolutifs, k-Plus Proches Voisins, Reconnaissance et Classification.

Abstract

Artificial neural networks (ANNs) are increasingly being used in various fields to solve machine learning problems. They are part of systems called "black box systems", because their internal logic is fuzzy to the user.

Trusting a black box system when dealing with a critical decision problem, such as detecting bank fraud, or deciding that a cell is cancerous, is a very difficult task. Several works are aimed at finding solutions that explain the results obtained by black-box systems, or at designing interpretable models that accompany an explanation of the predictions they make. This explanation may be a local interpretation of the prediction, which aims to make us understand why our ANN made such a decision. It may be post-hoc, which means that this explanation is presented by another model that accompanies our ANN. This explanatory model is said to be specific if it aims to explain a specific type of model and cannot be generalised to others.

In this end-of-study project, we proposed a solution that allows us to ex-plain the results obtained by convolutional neural networks (CNN), based on learning transfer to extract a map of characteristics from an image, transfer it to an MLP to obtain a prediction, and use it by a KNN that gives us a visual explanation of the prediction based on k closest neighbours at our request. This will allow us to visually interpret the prediction of an ANN, and to understand on which basis of the query we have obtained such a prediction. **Keywords :** Black Box Systems, Explainable Artificial Intelligence, Deep Learning, Twin Systems, Convolutional Neuron Networks, k-Neighbourhood Neural Networks, k-Nearest Neighbours, Recognition and Classification. In this end-of-study project, we proposed a solution that allows us to ex-plain the results obtained by convolutional neural networks, based on learning transfer to extract a map of characteristics from an image, transfer it to an MLP to obtain a prediction, and use it by a KNN that gives us a visual explanation of the prediction based on k closest neighbours at our request. This will allow us to visually interpret the prediction of an ANN, and to understand on which basis of the query we have obtained such a prediction.

Keywords : Black Box Systems, Explainable Artificial Intelligence, Deep Learning, Twin Systems, Convolutional Neuron Networks, k-Nearest Neighbours, Recognition and Classification.

ملخص

تُستخدم الشبكات العصبية الاصطناعية (Artificial Neural Networks) بشكل متزايد في مجالات مختلفة لحل مشكلات التعلم الآلي. وهي جزء من أنظمة تسمى "أنظمة الصندوق الأسود" ، وذلك لأن منطق التشغيل الداخلي الخاص بها غير واضح أمام المستخدم.

إن الثقة في نظام الصندوق الأسود عند التعامل مع مشكلة قرار حرجة ، مثل اكتشاف الاحتيال المصرفـي ، أو تحديد أن الخلية سرطانية ، هي مهمة صعبة للغاية. تهدف العديد من الأعمال إلى إيجاد حلول تشرح النتائج التي تم الحصول عليها من خلال أنظمة الصندوق الأسود ، أو لتصميم نماذج قابلة للتفسير مصحوبة بشرح للتنبؤات التي يحدونها. يمكن أن يكون هذا التفسير تفسيرًا محليًّا للتنبؤ ، والذي يهدف إلى جعلنا نفهم سبب اتخاذ شبكتنا العصبية الاصطناعية مثل هذا القرار. يمكن أن يكون ما بعد المخصص ، مما يعني أن هذا التفسير مقدم من خلال نموذج آخر يصاحب شبكتنا العصبية الاصطناعية. يُقال أن هذا النموذج التوضيحي محدد إذا كان يهدف إلى شرح نوع معين من النموذج ولا يمكن تعميمه على الآخرين.

في مشروع نهاية الدراسة هذا ، اقترحنا حلًّا يسمح لنا بشرح النتائج التي تم الحصول عليها بواسطة الشبكات العصبية التلaffيفية ، استنادًا إلى تعلم النقل لاستخراج خريطة مميزة للصورة. ، نقوم بنقلها إلى MLP للحصول على تنبؤ ، واستخدامه بواسطة KNN والذي سيعطينا شرحاً مرجئياً للتنبؤ في قاعدة k أقرب جيران لاستعلامنا. سيسمح لنا ذلك بتفسير توقع ANN بطريقة مرجئة ، وفهم أساس الطلب التي حصلنا عليها مثل هذا التنبؤ.

الكلمات الرئيسية: أنظمة الصندوق الأسود ، الذكاء الاصطناعي القابل للتفسير ، التعلم العميق ، الأنظمة المزدوجة ، الشبكات العصبية التلaffيفية ، قاعدة k أقرب جiran ، والاعتراف والتصنيف.

Liste des abréviations :

- **IA** : Intelligence Artificielle
- **XAI** : eXplainable Artificial Intelligence
- **AA** : Apprentissage Automatique
- **CBR** : Case Based Reasoning
- **RàPC** : Raisonnement à Partir de Cas
- **RNA** : Réseaux de Neuronnes Artificielles
- **CNN** : Réseaux de Neurones Convolutif
- **MLP** : Multi Layer Perceptron
- **FC** : Fully Connected Layers
- **KNN** : K Nearest Neighbors
- **RBF** : Radial Basis Functions
- **SVM** : Support Vector Machine
- **ReLU** : Rectified Linear Unit
- **Tanh** : Tangente Hyperbolique
- **MSE** : Mean Square Error
- **MAE** : Mean Absolute Error
- **GD** : Gradient Descent
- **NAG** : Nestrov Accelerated Gradient
- **SBN** : Système à Boite Noire
- **ABELE** : dversarialBlack box Explainer generating Latent Exemplars
- **AAE** : Advesial Autoencode
- **LIME** : Local InterpretableModel-Agnostic Explanations
- **CI** : Contextual Importance
- **CU** : Contextual Utility
- **EBN** : Explication de la Boite Noire
- **ERBN** : Explication des Résultats de la Boite Noire
- **IBN** : Inspection de la Boite Noire
- **CBT** : Conception de la Boite Transparente

Table des matières

1	Introduction Générale	11
2	L'intelligence artificielle et l'apprentissage par transfert	12
2.1	L'intelligence artificielle	12
2.2	Apprentissage automatique	12
2.2.1	L'apprentissage supervisé	13
2.2.2	L'apprentissage non supervisé	15
2.2.3	L'apprentissage semi-supervisé	15
2.2.4	L'apprentissage par renforcement	15
2.3	Apprentissage par transfert	16
2.3.1	Principe de l'apprentissage par transfert	16
2.3.2	Motivation de l'apprentissage par transfert	17
2.4	Conclusion	17
3	Réseaux de neurones artificiels et la classification	18
3.1	Neurone formel	18
3.1.1	Structure et fonctionnement	19
3.2	Fonction d'activation	21
3.3	Architectures des réseaux de neurones	22
3.3.1	Réseau monocouche	23
3.3.2	Réseau multicouche	23
3.3.3	Réseau bouclé ou récurrent	24
3.3.4	Réseau maillé	25
3.3.5	Réseau à connexion complète	26
3.3.6	Réseau à connexion locale	27
3.4	Les réseaux de neurones convolutifs	27
3.4.1	Couches de convolution	28
3.4.2	Le pas	29
3.4.3	Le Padding	29
3.4.4	Couche de Pooling	30
3.4.5	Couches entièrement connectées	31
3.5	Technique des K-plus proches voisins	32
3.6	Mesure de similarité	35
3.6.0.1	Distances de mesures de similarité	36

TABLE DES MATIÈRES

3.7 Conclusion	36
4 Apprentissage des réseaux de neurones	37
4.1 Principe	37
4.2 Les fonctions d'erreur	38
4.2.1 Erreurs de classification	38
4.2.1.1 La perte d'entropie croisée (Binary Cross-Entropy Loss / Log Loss) :	38
4.2.1.2 La perte de charnière (Hinge Loss) :	39
4.2.2 Erreurs de Regression	39
4.2.2.1 La fonction de perte quadratique moyenne (Mean Square Error / Quadratic Loss / L2 Loss) :	39
4.2.2.2 La fonction de perte absolue moyenne (Mean Absolute Error / L1 Loss) :	39
4.3 Algorithmes d'optimisation	40
4.3.1 Algorithme de descente du gradient	41
4.3.2 Autres algorithmes d'optimisation	43
4.3.2.1 AdaGrad	43
4.3.2.2 Adadelta	43
4.3.2.3 RMS Prop	44
4.3.2.4 Adam	44
4.4 Propagation Avant	44
4.5 Rétropropagation	46
4.6 Relation Propagation Avant et Arrière	47
4.7 Conclusion	48
5 Systèmes à boites noires	49
5.1 Modèles paramétrique et non-paramétriques	49
5.2 Explication ou interprétation des modèles à BN	50
5.2.1 Explication d'un modèle à boite noire	51
5.2.2 Explication des résultats d'une boîte noire	52
5.2.3 Problème d'inspection d'une boîte noire	52
5.2.4 Problème de conception de la boîte transparente	53
5.3 Techniques d'interprétation des systèmes à BN	53
5.3.1 Modèle interprétable intrinsèque	55
5.3.2 Explication globale post-hoc	55
5.3.3 Explication locale post-hoc	56
5.4 Travaux existants	56
5.4.1 Méthode ABELE	56
5.4.2 Méthode LIME	58

5.4.3	Méthodes à base de IC & UC	61
5.5	Méthode basée sur le Raisonnement à partir de Cas	64
5.5.1	Les systèmes jumeaux RNA-RàPC	65
5.5.2	Autres travaux	67
5.6	Synthèse des travaux	68
5.7	Conclusion	72
6	Conception	73
6.1	Architecture de la solution	73
6.2	Acquisition des données en entrée	74
6.2.1	Choix de notre base de données	75
6.3	Module d'extraction de caractéristiques	76
6.3.1	Technique d'extraction de caractéristiques	77
6.3.2	Réalisation du réseau de neurones convolutif VGG16	80
6.3.2.1	Architecture globale du réseau VGG-16	80
6.3.2.2	Base de donnée d'entraînement du VGG16	82
6.3.2.3	Visualisation des filtres des couches de convolutions	83
6.3.2.4	Visualisations des caractéristiques apprises dans le VGG16	84
6.3.2.5	Calcul de paramètres d'une couche de convolution	87
6.4	Module de classification	88
6.5	Module d'explication	90
6.5.1	Base de cas	91
6.5.2	Description d'un cas	92
6.5.3	Algorithme des plus proches voisins	92
6.6	Conclusion	93
7	Implémentation de la solution	95
7.1	Technologies et outils	95
7.2	Étapes de réalisation de notre système	99
7.2.1	Chargement des données	100
7.2.2	Extraction de caractéristiques	100
7.2.3	Modèle MLP	101
7.2.4	Modèle KNN	101
7.2.5	Visualisation et Sauvegarde	102
7.3	Conclusion	102
8	Test et Évaluation	103
8.1	Environnement de Test	103
8.2	Métriques d'évaluations	103
8.3	Choix des hyper-paramètres	105

TABLE DES MATIÈRES

8.3.1	Évaluation du modèle MLP	105
8.3.1.1	Résultats de tests	107
8.3.2	Évaluation du modèle KNN	108
8.3.2.1	Résultats de Tests	110
8.4	Résultat du couple CNN-KNN	112
8.4.1	Cas de prédiction correcte	113
8.4.2	Cas de prédiction fausse	114
8.5	Conclusion	115
9	Conclusion et perspectives	116

Introduction Générale

Le développement récent dans le domaine de l'intelligence artificielle a motivé plusieurs chercheurs pour œuvrer dans ce domaine. De plus en plus d'applications se basent sur les techniques d'apprentissage automatique. En effet, les réseaux de neurones artificiels (RNA) sont bien adaptés pour la prévision, la généralisation des décisions sur la base de données précédemment traitées, la classification des données, la création de méta-modèles à partir d'échantillons, etc.

Cependant, le fonctionnement de ces réseaux est souvent traité de fonctionnement en boite noire dû à leurs structures complexes. En effet, il est difficile de comprendre comment les résultats sont obtenus et quelle est la logique sous-jacente ayant conduits à ces derniers.

Ainsi, lorsqu'il y a erreurs ou les résultats ne correspondent pas, il est n'est pas facile d'identifier ce qui ne va pas. Le problème d'explication de résultats de système à boite noir est un sujet de recherche très actifs.

Le raisonnement par cas (CBR) est souvent utilisé pour l'explication de résultats. Les systèmes CBR se basent sur un type de raisonnement à partir d'exemples ou de cas utilisant la récupération et la réutilisation. Dans sa forme la plus simple, dans un CBR, lorsqu'un cas est présenté, les cas les plus similaires sont retrouvés avant d'être adaptés (ou utilisés directement) pour effectuer une prédiction / classification. En règle générale, l'étape de récupération trouve les cas en faisant correspondre les caractéristiques des cas dans la base de cas. Les CBR possèdent une transparence «naturelle» comme raisonnement à partir d'exemples précédents parallèle à ce que font parfois les experts. L'objectif de ce PFE consiste en la proposition et l'implémentation d'une nouvelle solution permettant l'explication de résultats de réseaux de neurones en se basant sur les CBR.

L'intelligence artificielle et l'apprentissage par transfert

2.1 L'intelligence artificielle

La technologie émergente de l'intelligence artificielle, ou (*IA*), croise plusieurs techniques simulant les processus cognitifs humains. Existant depuis les années 60, la recherche s'est développée récemment au point de multiplier les applications : voitures autonomes, diagnostics médicaux, assistants personnels, finance algorithmique, robots industriels, jeux vidéo... etc.

Le terme d'Intelligence Artificielle (*IA*), au demeurant largement controversé, prêté à de nombreuses confusions et interprétations. La notion d'intelligence étant elle-même complexe et relative, il n'est pas aisément de définir la discipline scientifique qu'est l'*IA*, elle peut être envisagée selon deux points de vue complémentaires :

- L'un concerne l'étude des mécanismes de l'intelligence artificielle, l'ordinateur étant utilisé comme moyen de simulation pour tester un modèle ou une théorie.
- L'autre concerne les efforts faits pour doter un ordinateur de capacités habituellement attribuées à l'intelligence humaine (acquisition de connaissances, perception, raisonnement, prise de décision,... etc). (Jean-Paul, Marie-Christine, 1993 [1]).

Définition : la construction de programmes informatiques qui s'adonnent à des tâches qui sont pour l'instant , accomplies de façon plus satisfaisante par des êtres humains car elles demandent des processus mentaux de haut niveau tels que : l'apprentissage perceptuel , l'organisation de la mémoire et le raisonnement critiquée (Minsky,2007 [2]) .

2.2 Apprentissage automatique

L'apprentissage automatique (*AA*) (ou Machine Learning en anglais) fait référence au développement, à l'analyse et à l'implémentation de méthodes qui permettent à une machine d'évoluer grâce à un processus d'apprentissage, et ainsi de remplir des tâches qu'il est difficile ou impossible de remplir par des moyens algorithmiques plus classiques

(A.Cornuéjols et al., 2002 [3]).

L'AA est une branche de l'IA qui concerne le développement d'algorithmes permettant de rendre une machine capable d'accomplir des tâches complexes sans avoir été explicitement programmée dans ce but.

Définition : Un programme informatique apprend de l'expérience **E** par rapport à une certaine classe de tâches **T** et à la mesure de performance **P**, si sa performance aux tâches en **T**, telle que mesurée par **P**, s'améliore avec l'expérience **E** (Tom M. Mitchell, 1997 [4]).

Exemple : dans un problème d'apprentissage de reconnaissance d'écriture manuscrite :

- Tâche **T** est : reconnaître et classer les mots manuscrits dans les images.
- Mesure de performance **P** est : pourcentage de mots correctement classés.
- Expérience **E** est : une base de données de mots manuscrits avec des classifications données.

L'apprentissage automatique est une discipline consacrée à l'analyse des données. Son but est de créer de la connaissance de manière automatique à partir de données brutes. Cette connaissance (ou modèle) peut alors être exploitée pour prendre des décisions. Comme le modèle est construit à partir des données, il est clair que plus on dispose de données, plus le modèle construit est précis et permettra ainsi de prendre de bonne décisions.

Tous les algorithmes d'apprentissage automatique utilisent un des types d'apprentissages principaux suivants [Figure2.1] :

- Apprentissage supervisé.
- Apprentissage non supervisé.
- Apprentissage semi-supervisé.
- Apprentissage par renforcement.

2.2.1 L'apprentissage supervisé

Dans l'apprentissage supervisé, l'ordinateur est fourni avec des exemples d'entrées ($x_i \in X^n$, avec $i \in 1 \dots n$)¹ qui sont étiquetés avec les sorties souhaitées ($y_i \in Y$)². Le but de cette méthode est que l'algorithme puisse «apprendre» en comparant sa sortie

1. Les exemples d'entrées ,ou encore échantillons, sont représentées sous forme d'un vecteur de n coordonnées : $X^n = (x_1, x_2, x_3, \dots, x_n)$.

2. Y identifie la décision à prendre pour chaque échantillon.

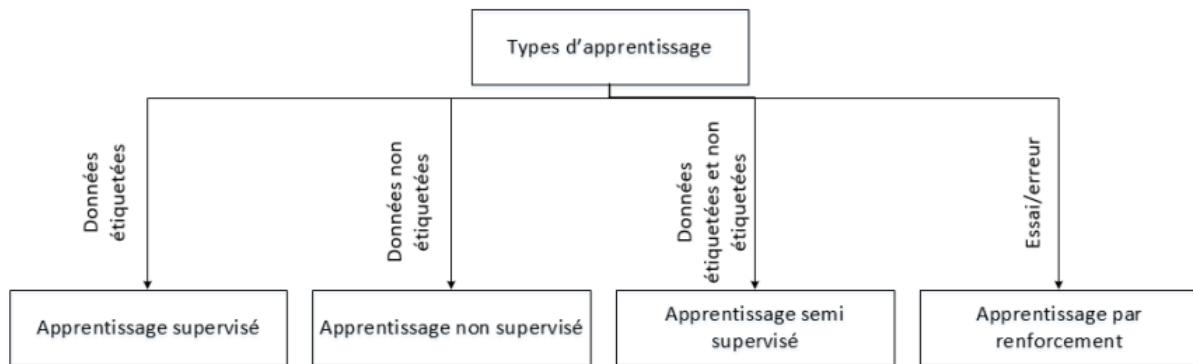


FIGURE 2.1 – Types d'apprentissage automatique

réelle avec les sorties «enseignées» pour trouver des erreurs et modifier le modèle en conséquence.

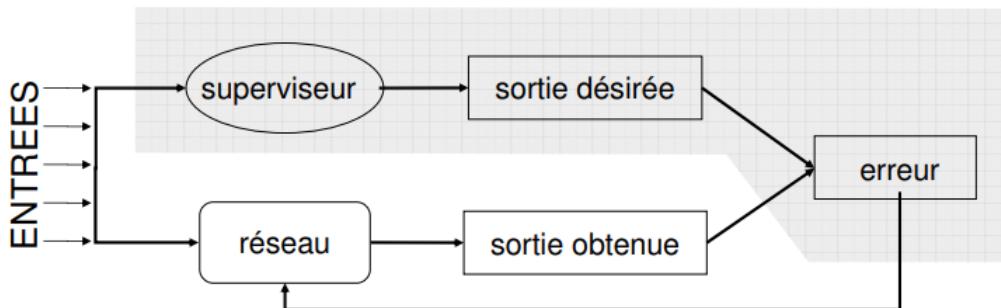


FIGURE 2.2 – Apprentissage supervisé.

L'apprentissage supervisé utilise donc des modèles pour prédire les valeurs d'étiquettes sur des données non étiquetées supplémentaires.

C'est ce qu'on appelle l'apprentissage supervisé, car le processus d'un algorithme tiré de l'ensemble de données de entraînement (training set) peut être considéré comme un enseignant supervisant le processus d'apprentissage. Nous connaissons les réponses correctes, l'algorithme effectue des prédictions itératives sur les données d'apprentissage et est corrigé par l'enseignant. L'apprentissage s'arrête lorsque l'algorithme atteint un niveau de performance acceptable [Figure 2.2]. Cette approche a pour objectif la conception d'un modèle reliant des données d'apprentissage à un ensemble de valeurs de sortie.

2.2.2 L'apprentissage non supervisé

Dans l'apprentissage non supervisé, les données (entrées $x_i \in X^n$, avec $i \in 1 \dots n$) sont non étiquetées, de sorte que l'algorithme d'apprentissage trouve tout seul des points communs parmi ses données d'entrée. Le but de cette méthode est de découvrir des modèles cachés dans un ensemble de données, mais il peut aussi avoir un objectif d'apprentissage des caractéristiques, qui permet à la machine intelligente de découvrir automatiquement les représentations nécessaires pour classer les données brutes (SUPINFO, 2017 [5]).

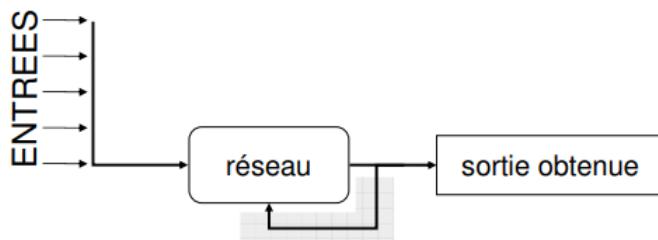


FIGURE 2.3 – Apprentissage non supervisé.

On l'appelle apprentissage non supervisé car les algorithmes sont laissés à leurs propres mécanismes pour découvrir et présenter la structure intéressante des données [Figure 2.3]. Cette approche vise à concevoir un modèle structurant l'information.

2.2.3 L'apprentissage semi-supervisé

L'apprentissage semi-supervisé utilise un ensemble de données étiquetées et non-étiquetées. Il se situe ainsi entre l'apprentissage supervisé qui n'utilise que des données étiquetées et l'apprentissage non-supervisé qui n'utilise que des données non-étiquetées.

Ce processus d'apprentissage permet d'obtenir un modèle pondéré pour prédire les réponses de données similaires qui n'ont pas encore été labellisées. L'apprentissage semi-supervisé utilise à la fois des données labellisées et non labellisées pour s'adapter à un modèle.

2.2.4 L'apprentissage par renforcement

L'apprentissage par renforcement est une méthode d'apprentissage pour les modèles de l'AA. Cette méthode consiste à laisser l'algorithme apprendre de ses propres essais/erreurs. Afin d'apprendre à prendre les bonnes décisions, l'intelligence artificielle se retrouve directement confrontée à des choix. Si elle se trompe, elle est «pénalisée». Au contraire, si elle prend la bonne décision, elle est «récompensée».

Un **agent** dans un **état actuel S** apprend de son **environnement** en interagissant avec ce dernier par le moyen d'**actions**. Suite à **une action A**, l'environnement retourne un **nouvel état S'** et une **récompense R** associée, qui peut être «positive» ou «négative» [Figure 2.4].

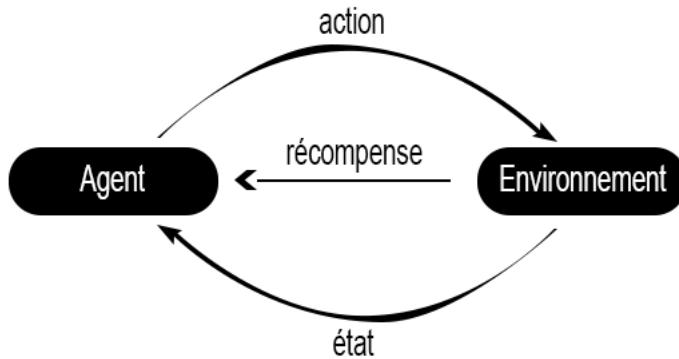


FIGURE 2.4 – Apprentissage par renforcement.

2.3 Apprentissage par transfert

L'apprentissage par transfert (ou transfer learning, en Anglais) est une méthode d'apprentissage automatique dans laquelle un modèle développé pour une tâche est réutilisé comme point de départ pour un modèle sur une deuxième tâche. C'est une approche populaire dans l'apprentissage approfondi où des modèles préformés sont utilisés comme point de départ pour des tâches de vision par ordinateur et de traitement du langage naturel, étant donné les vastes ressources de calcul et de temps nécessaires pour développer des modèles de réseaux de neurones sur ces problèmes et les énormes sauts de compétence qu'ils permettent d'obtenir sur des problèmes connexes.(Machine learning mastery, 2017 [55])

Définition : L'apprentissage par transfert et l'adaptation au domaine font référence à la situation dans laquelle ce qui a été appris dans un cadre, est exploité pour améliorer la généralisation dans un autre cadre (Ian Goodfellow et al., 2016 [56]).

2.3.1 Principe de l'apprentissage par transfert

Le principe du Transfer Learning (ou apprentissage par transfert) est d'utiliser les connaissances acquises par un réseau de neurones lors de la résolution d'un problème afin d'en résoudre un autre plus ou moins similaire, dans notre cas il s'agit du problème d'interprétabilité ou d'explication.

2.3.2 Motivation de l'apprentissage par transfert

La motivation principale, surtout dans le contexte de l'apprentissage approfondi, est le fait que la plupart des modèles qui résolvent des problèmes complexes nécessitent un grand nombre de données, et qu'il peut être très difficile d'obtenir de grandes quantités de données étiquetées pour les modèles supervisés, compte tenu du temps et des efforts nécessaires pour étiqueter les points de données. En outre, la plupart des modèles d'apprentissage approfondi sont très spécialisés dans un domaine particulier, voire dans une tâche spécifique. Il peut s'agir de modèles de pointe, d'une très grande précision et dépassant tous les critères de référence, mais ils ne portent que sur des ensembles de données très spécifiques et finissent par subir une perte de performance importante lorsqu'ils sont utilisés dans une nouvelle tâche qui peut encore être similaire à celle pour laquelle ils ont été formés. C'est ce qui motive l'apprentissage par transfert, qui va au-delà des tâches et des domaines spécifiques, et tente de voir comment tirer parti des connaissances des modèles préformés et les utiliser pour résoudre de nouveaux problèmes (Towards data science, 2018 [58]).

2.4 Conclusion

Dans ce chapitre nous avons donné une définition de l'intelligence artificielle et nous avons exposé les différents types d'apprentissage automatique les plus utilisés lors d'une résolution d'un problème d'IA. Nous avons en plus vu le principe de l'apprentissage par transfert qui nous permet d'utiliser des algorithmes pré-entraînés et les adapter à de nouveaux cas d'études. Nous allons maintenant présenter l'une des approches les plus utilisées dans le problème de classification dans l'apprentissage automatique, les réseaux de neurones artificiels.

Réseaux de neurones artificiels et la classification

Un réseau de neurones artificiels (*RNA*) (ou Artificial Neural Networks , en anglais) est un ensemble d’algorithmes dont le fonctionnement est inspiré des neurones biologiques et qui s’appuie également sur les méthodes statistiques.

L’usage de réseaux de neurones est une technique de l’Intelligence Artificielle, optimisant les méthodes d’apprentissage automatiques. Ils démontrent un mécanisme perceptif indépendant des idées propres du développeur (Simon Haykin, 2009 [7]).

Les réseaux de neurones sont des techniques de modélisation et de prévision, permettant de modéliser des relations entre des données ou des fonctions particulièrement complexes (David Kriesel,2005 [8]).

Définition : Un réseau de neurones est un ensemble interconnecté d’éléments, d’unités ou de noeuds de traitement simples, dont la fonctionnalité est vaguement basée sur le neurone biologique. La capacité de traitement du réseau est stockée dans les forces de connexion interunités, ou poids, obtenues par un processus d’adaptation à, ou d’apprentissage à partir de, un ensemble de modèles d’entraînement (Kevin Gurney, 2004[6]).

L’étude des réseaux de neurones artificiels est motivée par leur similitude avec le bon fonctionnement des systèmes biologiques, ce qui ,en comparaison avec le système global, se compose de cellules nerveuses très simples mais nombreuses qui travaillent massivement en parallèle et (qui est probablement l’un des plus importants aspects) ont la capacité d’apprendre (David Kriesel,2005 [8]).

3.1 Neurone formel

Un réseau de neurones tire sa puissance de calcul de sa structure distribuée massivement parallèle et de sa capacité à apprendre et donc à généraliser. La généralisation fait

référence à la production par le réseau de neurones de sorties raisonnables pour des entrées non rencontrées pendant l'entraînement (apprentissage). Ces deux capacités de traitement de l'information permettent aux réseaux de neurones de trouver des solutions approximatives satisfaisantes à des problèmes complexes (à grande échelle) qui sont insolubles. Le neurone formel est une unité de traitement de l'information qui est fondamentale pour le fonctionnement d'un réseau de neurones. Il constitue la base de la conception d'une grande famille de réseaux de neurones (Simon Haykin, 2009 [7]).

3.1.1 Structure et fonctionnement

La [Figure 3.1] montre la structure d'un neurone artificiel. Chaque neurone artificiel est un processeur élémentaire alimenté par des signaux d'entrées provenant d'autres neurones amonts. À ces signaux, des poids synaptiques w (weight, en Anglais) sont associés pour représenter la force de connexion. Une fonction d'activation calcule la valeur d'état du neurone pour alimenter d'autres neurones avants à travers une sortie unique (Claude Touzet, 1992[9]).

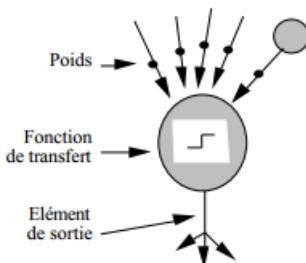


FIGURE 3.1 – Structure d'un neurone artificiel.

Les neurones artificiels utilisés dans les réseaux de neurones artificiels sont non linéaires, fournissent des résultats continus et exécutent des fonctions simples, telles que la collecte des signaux disponibles sur leurs entrées, en les assemblant en fonction de leur état de fonctionnement, et produisent une réponse en tenant compte de leurs fonctions d'activation innées (Ivan et al., 2017[10]).

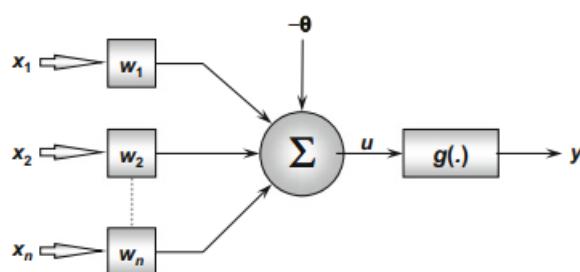


FIGURE 3.2 – Représentation d'un neurone artificiel.

Le modèle de neurone le plus simple qui inclut les principales caractéristiques d'un modèle de réseau de neurones biologique a été proposé par (McCulloch & Pitts, 1943), et est encore le modèle le plus utilisé dans différentes architectures de réseaux de neurones artificiels. Dans ce modèle chaque neurone du réseau peut être implémenté comme montré dans la [Figure 3.2].

En considérant la [Figure 3.2], il est possible de voir que le neurone artificiel est composé de sept éléments de base :

- **Les signaux d'entrée** (x_1, x_2, \dots, x_n) : sont les signaux ou échantillons provenant de l'environnement externe et représentant les valeurs assumées par les variables d'une application particulière. Les signaux d'entrée sont généralement normalisés afin d'améliorer l'efficacité informatique des algorithmes d'apprentissage.
- **Les poids synaptiques** (w_1, w_2, \dots, w_n) : sont les valeurs utilisées pour pondérer chacune des variables d'entrée, ce qui permet de quantifier leur pertinence par rapport à la fonctionnalité du neurone.
- **L'agrégateur linéaire** (\sum) : rassemble tous les signaux d'entrée pondérés par les poids synaptiques pour produire une tension d'activation.
- **Le seuil d'activation ou biais** (θ) : est une variable utilisée pour spécifier le seuil approprié que le résultat produit par l'agrégateur linéaire doit avoir pour générer une valeur de déclenchement vers le neurone de sortie.
- **Le potentiel d'activation** (u) : est le résultat produit par la différence entre l'agrégateur linéaire et le seuil d'activation. Si cette valeur est positive, c'est-à-dire si $u \geq \theta$, le neurone produit un potentiel excitateur, sinon il sera inhibiteur.
- **Fonction d'activation** (g) : dont le but est de limiter la sortie du neurone dans une plage raisonnable de valeurs, assumée par sa propre image fonctionnelle.
- **Le signal de sortie** (y) : consiste en la valeur finale produite par le neurone étant donné un ensemble particulier de signaux d'entrée, et peut également être utilisé comme entrée pour d'autres neurones interconnectés de manière séquentielle.

Les deux expressions suivantes synthétisent le résultat produit par le neurone artificiel proposé par (McCulloch et Pitts) :

$$u = \sum_{i=1}^n w_i \times x_i - \theta \quad (3.1)$$

$$y = g(u) \quad (3.2)$$

Nous pouvons résumer l'opération du neurone artificiel par les étapes suivantes :

- Présenter un ensemble de valeurs au neurone, représentant les variables d'entrée.

- Multiplier chaque entrée du neurone par son poids synaptique correspondant.
- Obtenir le potentiel d'activation produit par la somme pondérée des valeurs d'entrée et soustraire le seuil d'activation.
- Appliquer une fonction d'activation appropriée pour limiter la sortie des neurones.
- Compiler la sortie en utilisant la fonction d'activation neuronale dans le potentiel d'activation.

3.2 Fonction d'activation

C'est une fonction présentée généralement par une non linéarité appelée aussi fonction de seuil. Elle permet de définir l'état interne du neurone en fonction de son entrée totale. Les fonctions d'activation sont sélectionnées en fonction des applications pour but de modéliser le comportement non linéaire de la cellule où il n'y a pas de sortie en dessous d'une certaine valeur de l'argument. Il existe de nombreuses formes possibles pour la fonction d'activation, les plus courantes sont présentées sur la [Figure 3.3].

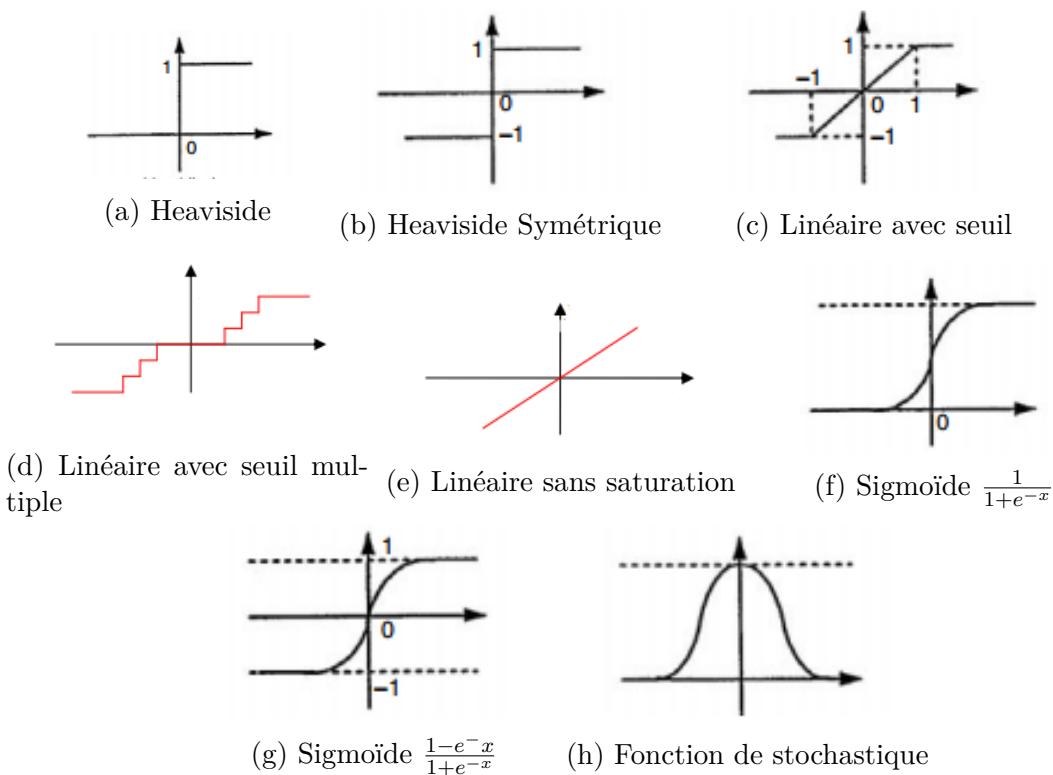


FIGURE 3.3 – Différentes formes de la fonction d'activation pour le neurone artificiel.

La plupart des fonctions d'activations sont continues, offrant une infinité de valeurs possibles comprises dans l'intervalle $[0, +1]$ (ou $[-1, +1]$). Toutes les fonctions d'activation utilisées doivent être différentiables, car l'architecture du réseau de neurones l'impose pour que l'apprentissage soit possible (Jagannathan Sarangapani, 2006[11]).

3.3 Architectures des réseaux de neurones

L’architecture d’un réseau de neurones artificiels définit comment ses différents neurones sont disposés ou placés les uns par rapport aux autres. Ces arrangements sont structurés essentiellement en dirigeant les connexions synaptiques des neurones (Ivan et al., 2017[10]).

En général, un réseau de neurones artificiels peut être divisé en trois parties, appelées couches [Figure 3.4], qui sont connus sous les noms de :

- **Couche d’entrée** : Cette couche est responsable de la réception des informations (données), signaux, caractéristiques ou mesures de l’environnement externe.
- **Couches cachées, intermédiaires ou invisibles** : Ces couches sont composées de neurones responsables de l’extraction des motifs associés au processus ou au système analysé. Elles effectuent la majeure partie du traitement interne à partir d’un réseau.
- **Couche de sortie** : Cette couche est responsable de la production et de la présentation des sorties finales du réseau , qui résultent du traitement effectué par les neurones dans les couches précédentes.

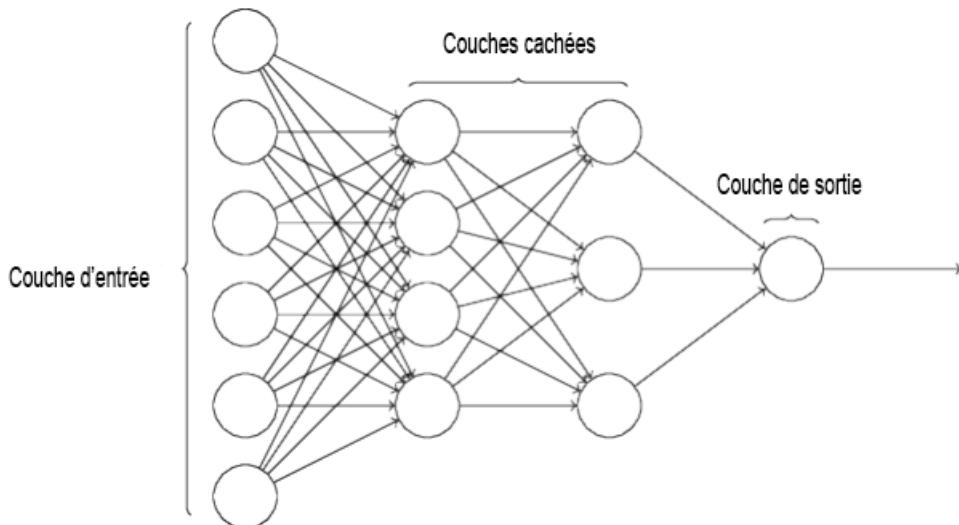


FIGURE 3.4 – Couches d’un réseau de neurones artificiels.

Les principales architectures des réseaux de neurones artificiels, compte tenu de la disposition des neurones, ainsi que la façon dont ils sont interconnectés et comment ses couches sont composées, peuvent être divisées comme suit :

- Réseau monocouche.
- Réseau multicouche.
- Réseau bouclé ou récurrent.

- Réseau maillé.
- Réseau à connexion complète.
- Réseau à connexions locales.

3.3.1 Réseau monocouche

Ce réseau de neurones artificiels n'a qu'une seule couche d'entrée et une seule couche neuronale, qui est aussi la couche de sortie. La [Figure 3.5] illustre une couche simple à (*propagation avant ou feedforward en anglais*)¹ du réseau composé de n entrées et m sorties.

L'information circule toujours dans une seule direction (donc unidirectionnelle), c'est-à-dire de la couche d'entrée à la couche de sortie. A partir de la [Figure 3.5] il est possible de voir que dans les réseaux appartenant à cette architecture, le nombre de sorties réseau coïncidera toujours avec le nombre de ses neurones. Ces réseaux sont généralement utilisés pour la classification des motifs et les problèmes de filtrage linéaire (Ivan et al., 2017[10]).

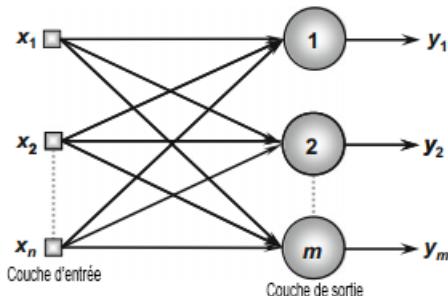


FIGURE 3.5 – Architecture d'un réseau monocouche.

3.3.2 Réseau multicouche

Les réseaux multicouche à propagation avant sont composés d'une ou plusieurs couches neuronales cachées [Figure 3.6]. Ils sont employés dans la résolution de divers problèmes, comme ceux liés à l'approximation des fonctions, à la classification des modèles, à l'identification des systèmes, au contrôle des processus, à l'optimisation, à la robotique, etc. La [Figure 3.6] montre un réseau multicouche à propagation avant composé d'une couche d'entrée avec n signaux d'échantillonnage, deux couches neuronales cachées constituées de n_1 et n_2 neurones respectivement, et, enfin, une couche neuronale de sortie composée

1. Dans un réseau de neurones à propagation avant, l'information ne se déplace que dans une seule direction, vers l'avant, à partir des noeuds d'entrée, en passant par les couches cachées (le cas échéant) et vers les noeuds de sortie. Il n'y a pas de cycles ou de boucles dans le réseau

de m neurones représentant les valeurs de sortie respectives du problème en analyse (Ivan et al., 2017[10]).

A partir de la [Figure 3.6], il est possible de comprendre que la quantité de neurones composant la première couche cachée est généralement différente du nombre de signaux composant la couche d'entrée du réseau. En fait, le nombre de couches cachées et leur quantité respective de neurones dépendent de la nature et de la complexité du problème cartographié par le réseau, ainsi que de la quantité et de la qualité des données disponibles sur ce problème. La quantité de signaux de sortie coïncidera toujours avec le nombre de neurones de cette couche respective.

Parmi les principaux réseaux utilisant des architectures de type multicouche à propagation vante, on peut citer les réseaux Perceptron multicouche (ou Multiple layer Perceptrons en anglais, MLP) et la fonction de base radiale (ou Radial Basis Functions en anglais, RBF), dont les algorithmes d'apprentissage utilisés dans leurs processus d'apprentissage sont respectivement basés sur les algorithmes généralisés de la règle *delta* et la règle *competitive/delta*.

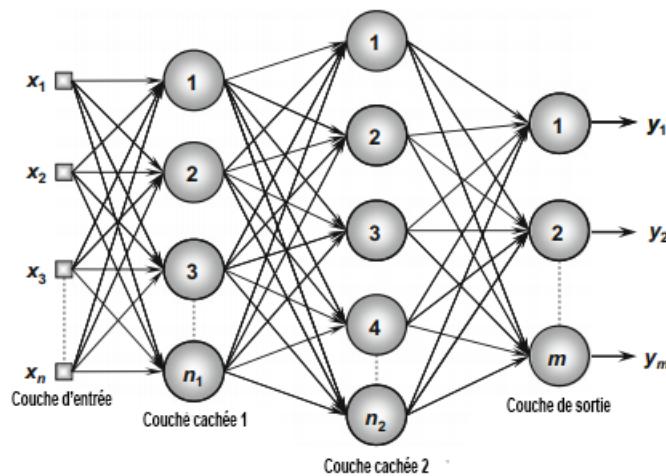


FIGURE 3.6 – Architecture d'un réseau multicouche.

3.3.3 Réseau bouclé ou récurrent

Dans ces réseaux, les sorties des neurones sont utilisées comme entrées de rétroaction (ou Feedback en anglais) pour d'autres neurones. La fonction de rétroaction qualifie ces réseaux pour le traitement dynamique de l'information, ce qui signifie qu'ils peuvent être utilisés sur des systèmes à variation temporelle, comme la prédition de séries temporelles, l'identification et l'optimisation de systèmes, le contrôle de processus, etc. La [Figure 3.7] illustre un exemple de réseau Perceptron avec rétroaction, où un de ses signaux de sortie

est renvoyé à la couche intermédiaire. Ainsi, en utilisant le processus de rétroaction, les réseaux dotés de cette architecture produisent en tenant également compte des valeurs de sortie précédentes (Ivan et al., 2017[10]).

Parmi les principaux réseaux récurrents, on trouve le Hopfield et le Perceptron avec rétroaction entre neurones de couches distinctes, dont les algorithmes d'apprentissage utilisés dans leurs processus d'apprentissage sont respectivement basés sur la minimisation de la fonction énergétique et la règle *delta* généralisée.

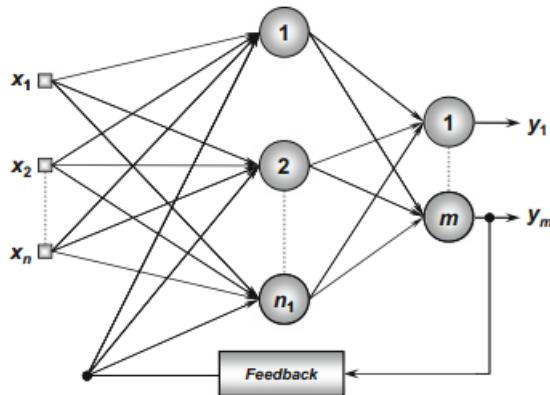


FIGURE 3.7 – Architecture d'un réseau bouclé ou récurrent.

Définition : Un réseau de neurone bouclé à temps discret réalise une ou plusieurs équations aux différences non linaires, par composition des fonctions réalisées par chacun des neurones et des retards associés à chacune des connexions (Dreyfus et al., 2002 [12]).

Ces réseaux se caractérisent par la présence d'au moins une boucle de rétroaction au niveau des neurones ou entre les couches, et la prise en compte de l'aspect temporel du phénomène.

3.3.4 Réseau maillé

Les principales caractéristiques des réseaux à maillage résident dans la prise en compte de l'arrangement spatial des neurones à des fins d'extraction de modèles, c'est-à-dire que la localisation spatiale des neurones est directement liée au processus d'ajustement de leurs poids et seuils synaptiques. Ces réseaux desservent un large éventail d'applications et sont utilisés pour des problèmes de regroupement de données, de reconnaissance de formes, d'optimisation de systèmes, de graphiques, etc (Ivan et al., 2017[10]).

A partir de la [Figure 3.8], il est possible de vérifier que dans cette catégorie de réseau, les différents signaux d'entrée sont lus par tous les neurones du réseau.

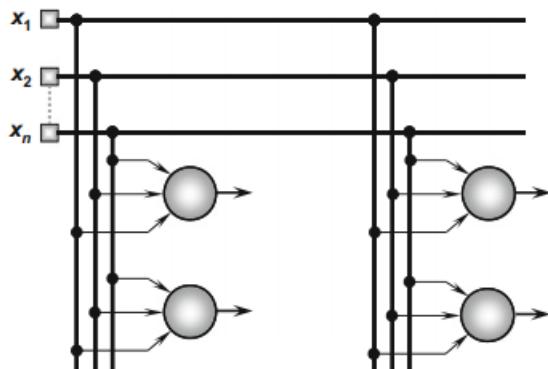


FIGURE 3.8 – Architecture d'un réseau maillé.

Le réseau Kohonen est le principal représentant des architectures de maillage, et son entraînement est réalisée dans le cadre d'un processus concurrentiel. La [Figure 3.8] illustre un exemple du réseau de Kohonen où ses neurones sont disposés dans un espace bidimensionnel.

3.3.5 Réseau à connexion complète

Les réseaux entièrement reliés entre eux permettent des connexions entre tous les neurones [Figure 3.9]. Un exemple populaire est celui des cartes auto-organisatrices (ou self organizing maps en anglais, SOM) de Kohonen, qui sont utilisées pour cartographier un espace réel, c'est-à-dire pour étudier la répartition de données dans un espace à grande dimension. En pratique, cette cartographie peut servir à réaliser des tâches de discréétisation, quantification vectorielle ou classification (David Kriesel, 2005 [8]).

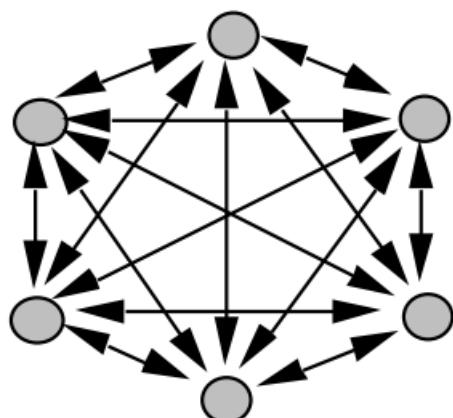


FIGURE 3.9 – Architecture d'un réseau à connexion complète.

Dans ce cas, chaque neurone est toujours autorisé à être connecté à tous les autres neurones, mais en conséquence, chaque neurone peut devenir un neurone d'entrée.

3.3.6 Réseau à connexion locale

Il s'agit d'une structure multicouche, mais qui à l'image de la rétine, conserve une certaine topologie. Chaque neurone entretient des relations avec un nombre réduit et localisé de neurones de la couche avale [Figure 3.10]. Les connexions sont donc moins nombreuses que dans le cas d'un réseau multicouche classique (Calude Touzet, 1992[9]).

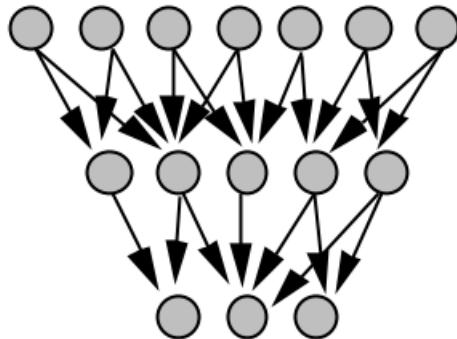


FIGURE 3.10 – Architecture d'un réseau à connexion locale.

Remarque : Les réseaux de neurones étant des boîtes noires, les résultats fournis ne sont guère explicites et ne conduisent donc pas à des interprétations peu informatives du modèle. Seule une étude des erreurs de prévisions et, dans le cas d'une régression, une étude des résidus, permet de se faire une idée de la qualité du modèle.

3.4 Les réseaux de neurones convolutifs

Dans les réseaux de neurones, le réseau de neurones convolutionnels est l'une des principales catégories pour faire de la reconnaissance d'images, la classification d'images, la détection d'objets, et la reconnaissance faciale, etc.

Les CNNs prennent une image d'entrée, la traitent et la classent dans une certaine catégorie (par exemple, chien, chat, tigre, lion). L'image d'entrée est représentée par un ensemble de pixels organisés une matrice. Si l'image est de couleur, les pixels sont organisés dans 3 matrices, chaque matrice reflète un canal du RVB [Figure 3.11], sinon si elle est en niveau de gris une seule matrice est présentée.

Un réseau CNN est composée de trois types de couches :

- **Couches de convolution** : Elles réalisent un filtrage par convolution pour repérer la présence d'un ensemble de caractéristiques dans les images en entrée.
- **Couches de pooling** : Elles réduisent la taille des images en préservant leurs caractéristiques les plus importantes.

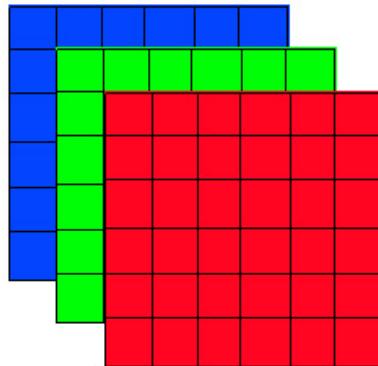


FIGURE 3.11 – Tableau d'une matrice RVB.

- **Couches entièrement connectées** : Permettent de classer l'image en entrée du réseau en appliquant une combinaison linéaire et une fonction d'activation.

La [Figure 3.12] montre les différentes couches d'un CNN, leur contribution pour la classification d'une image dans une classe parmi celle possible.

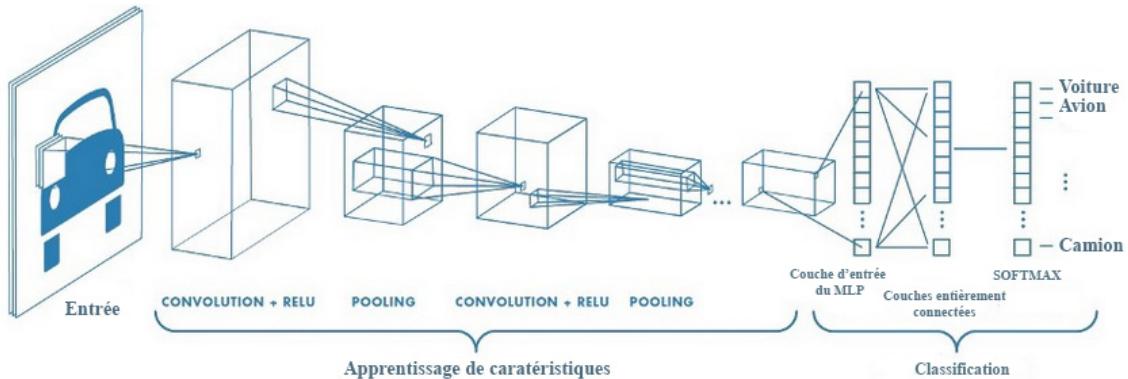


FIGURE 3.12 – Réseau de neurones avec de nombreuses couches convolutives.

3.4.1 Couches de convolution

La convolution est la première couche qui permet d'extraire des caractéristiques d'une image d'entrée. Les convolutions fonctionnent comme des filtres et glissent à travers toute l'image en capturant les caractéristiques les plus marquantes. C'est une opération mathématique qui nécessite deux entrées telles qu'une matrice d'image et un filtre ou un noyau.

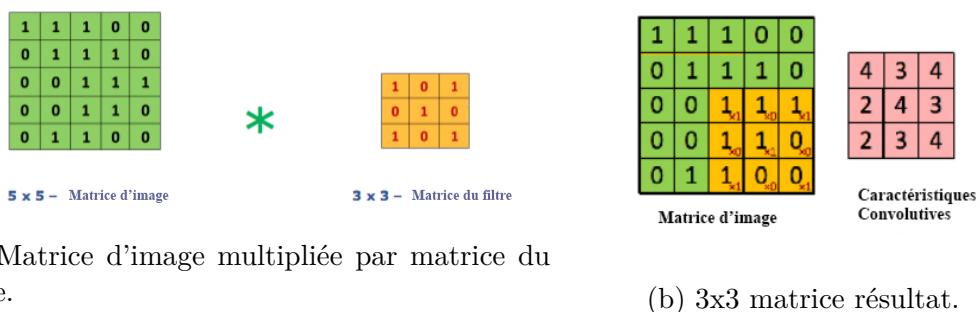
la couche convolutionnelle génère des cartes de caractéristiques par des filtres convolutionnels linéaires suivis de fonctions d'activation non linéaires, telles que *ReLU*, *sigmoïde*, *tanh*, etc. Dans cette couche, la k ème carte de caractéristiques de sortie y_k peut être

calculée comme suit :

$$y_k = f(w_k * x), \quad (3.3)$$

où x désigne l'image d'entrée, w_k représente le filtre convolutionnel associé à la k ième carte de caractéristiques, $*$ indique l'opérateur de convolution 2D qui est utilisé pour calculer le produit interne du modèle de filtre à chaque endroit de l'image d'entrée, et f est la fonction d'activation non linéaire.

Considérons une matrice 5×5 dont les valeurs des pixels de l'image sont 0, 1 et la matrice de filtrage 3×3 . Ensuite, la convolution de la matrice d'image 5×5 se multiplie avec la matrice de filtrage 3×3 qui est appelée "Feature Map" comme indiqué dans la [Figure 3.13].



(a) Matrice d'image multipliée par matrice du filtre. (b) 3×3 matrice résultat.

FIGURE 3.13 – Processus des couches de convolution.

La profondeur de sortie de la convolution est égale à la quantité de filtres appliqués. Plus les couches convolutionnelles sont profondes, plus les caractéristiques identifiées par la carte d'activation sont détaillées. La convolution d'une image avec différents filtres permet d'effectuer des opérations telles que la détection des contours, le flou et la netteté en appliquant des filtres. Le filtre, également appelé noyau, est formé par des poids initialisés de façon aléatoire, mis à jour par chaque nouvelle entrée par rétropropagation.

3.4.2 Le pas

le pas est le nombre de décalages de pixels sur la matrice d'entrée. Lorsque le pas est de 1, nous déplaçons les filtres à 1 pixel à la fois. La [Figure 3.14] montre la convolution dans une image à 3 canaux RVB avec le décalage du filtre d'un pas de 1.

3.4.3 Le Padding

En général, les pixels du milieu sont plus souvent utilisés que ceux des coins et des bords. Par conséquent, les informations sur les bords des images ne sont pas aussi bien

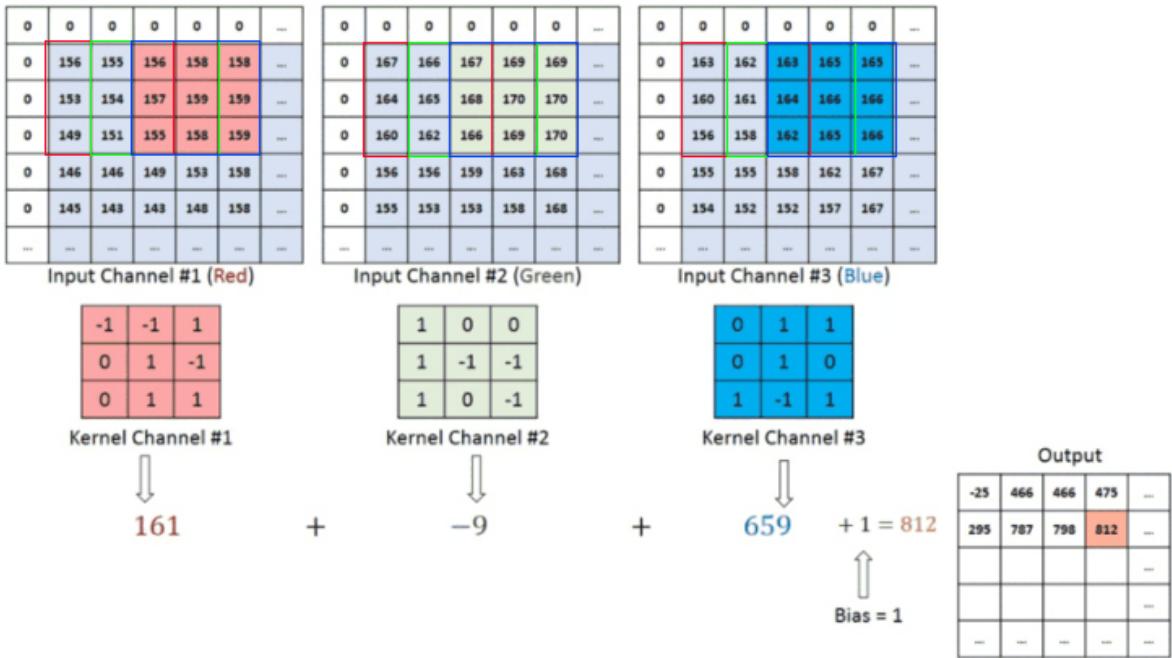


FIGURE 3.14 – Réseau convolutionnel avec filtre 3x3, 1 pas et entrée de padding de zéro.

préserver que celles du milieu et le filtre ne s'adapte pas parfaitement à l'image d'entrée. Nous utilisons donc le Padding qui peut être utilisé de deux manière :

- Remplir l'image avec des zéros (zero-padding) pour qu'elle s'adapte [Figure 3.14].
- Déposez la partie de l'image où le filtre n'est pas adapté. C'est ce qu'on appelle un valid Padding qui ne conserve qu'une partie valide de l'image.

3.4.4 Couche de Pooling

Ce type de couches permettrait de réduire le nombre de paramètres lorsque les images sont trop grandes. Le pooling spatial aussi appelée sous-échantillonnage réduit la dimensionnalité de chaque carte des caractéristiques mais conserve des informations importantes. Le pooling spatial peut être de deux types :

- Max Pooling
- Average Pooling

La méthode de "Max Pooling" sélectionne l'élément le plus important dans chaque région de Pooling comme :

$$y_{kij} = \max_{(p,q) \in R_{ij}} x_{kpq}, \quad (3.4)$$

où y_{kij} est la sortie de l'opérateur de Pooling liée à la ki ème carte de caractéristiques, x_{kpq} est l'élément (p, q) dans la région de Pooling R_{ij} qui représente un voisinage local

autour de la position (i, j) .

La [Figure 3.15] présente un exemple de "Max Pooling" sur une matrice de 4×4 .

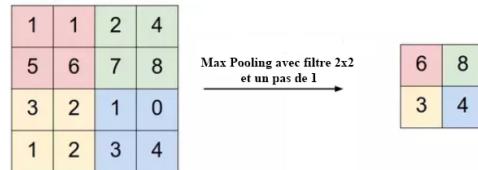


FIGURE 3.15 – Max Pooling.

la méthode "Average Pooling" prend la moyenne arithmétique des éléments dans chaque région de Pooling comme :

$$y_{kij} = \frac{1}{|R_{ij}|} \sum_{(p,q) \in R_{ij}} x_{kpq} \quad (3.5)$$

où $|R_{ij}|$ correspond à la taille de la région de Pooling de R_{ij} .

3.4.5 Couches entièrement connectées

La dernière couche du CNN contient les caractéristiques finales de l'image en entrée. Cette matrice de caractéristiques est aplatie en un vecteur et transmise vers la couche entièrement connectée (Couche FC) [Figure 3.16]. Au niveau des couches entièrement connectées, ces caractéristiques sont combinées. Puis, grâce à une fonction d'activation telle que *softmax* ou *sigmoïde* elle classe les sorties dans la catégorie correspondante est renvoyée.

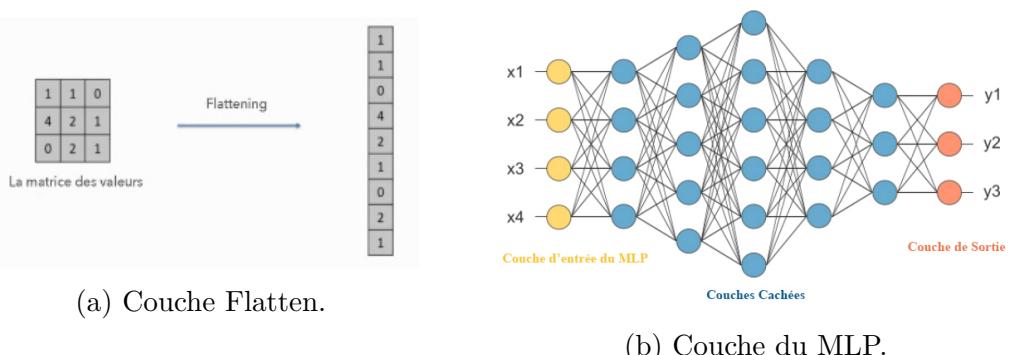


FIGURE 3.16 – La couche de pooling est aplatie en une couche FC.

3.5 Technique des K-plus proches voisins

La technique de classification la plus populaire est les K-Plus Proches Voisins (KNN, ou K nearest neighbors en Anglais). L'algorithme KNN est un algorithme qui est souvent utilisé dans la classification. L'objectif de cet algorithme est de classifier de nouveaux objets sur la base des attributs et des données de formation. La classification des objets basée sur les données d'entraînement qui ont la distance la plus proche d'un nouvel objet est basée sur les métriques de distance.

l'algorithme KNN fonctionne pour classer les nouvelles données en fonction de leur proximité avec les k plus proches voisins parmi les données de formation. Ainsi, si les nouvelles données sont entourées de données d'entraînement d'une classe, on peut conclure que les nouvelles données sont incluses dans cette même classe.

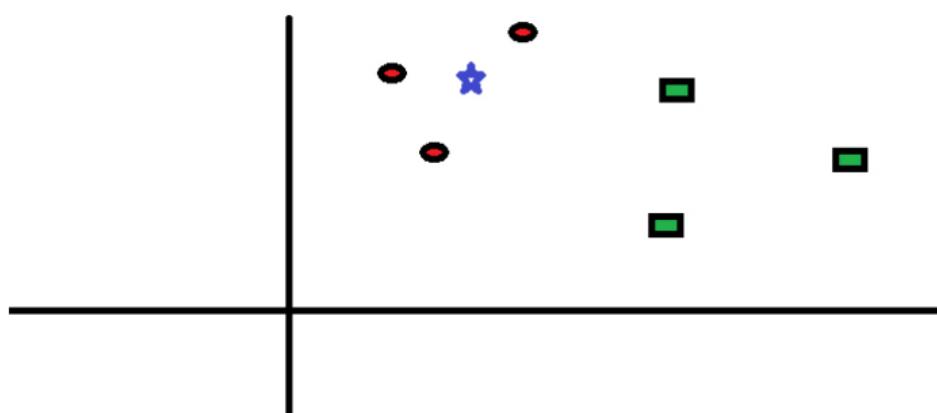


FIGURE 3.17 – Exemple 1 de classification par un KNN.

Si on prend exemple de classification de l'objet étoile bleue (EB) parmi les deux classes des cercles rouges (CR) et des carrés verts (CV) dans la [Figure 3.17]. Le "k" dans un knn est le nombre des voisins les plus proches qu'on veut sélectionner parmi nos données d'entraînement. Si on prend "k = 3" alors nous allons donc former un cercle avec EB au centre, aussi grand qui ne contient que trois points de données [Figure 3.18].

Le choix du paramètre "k" dans l'algorithme KNN est crucial, sa variation contribue à la variation des métriques d'évaluation d'un KNN. Dans la [Figure 3.18], étant donné que les 6 observations d'entraînement restent constantes, avec une valeur "K" donnée, nous pouvons établir des limites pour chaque classe. Ces limites sépareront CR de CV. De la même manière, essayons de voir l'effet de la valeur "K" sur les limites de la classe. La [Figure 3.19] illustre les différentes limites séparant les deux classes avec des valeurs de "K" différentes.

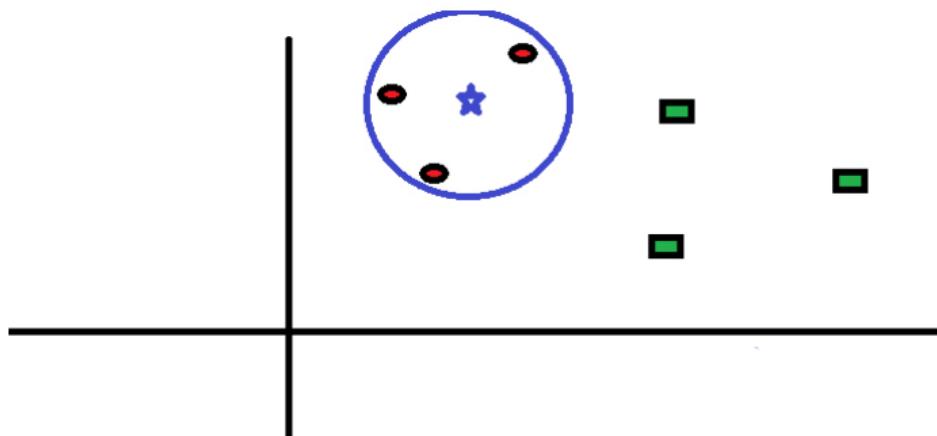


FIGURE 3.18 – Exemple 1 de classification par un KNN.

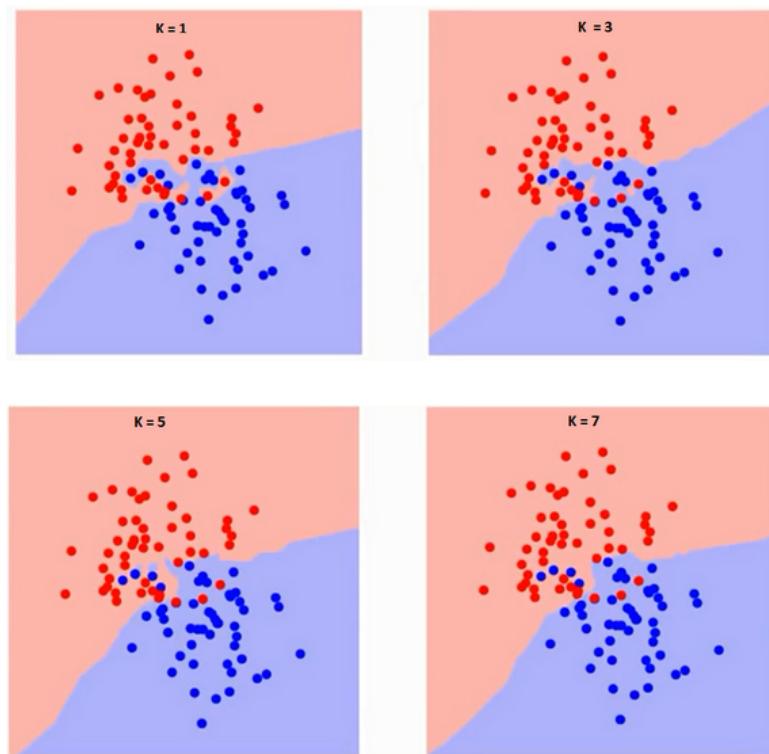


FIGURE 3.19 – Contribution de la valeur de "k" dans la classification dans un KNN.

Dans la [Figure 3.19], nous pouvons voir que la frontière devient plus lisse avec la valeur croissante de "K". Avec "K" croissant à l'infini, elle devient finalement toute bleue ou toute rouge selon la majorité totale. Le taux d'erreur d'entraînement et le taux d'erreur de validation sont deux paramètres dont nous avons besoin pour accéder aux différentes valeurs de "K". La [Figure 3.20] représente la courbe du taux d'erreur d'entraînement avec une valeur variable de "K" dans l'expérience de (Tavish Srivastava, 2018 [61]).

Comme montré dans la [Figure 3.20], le taux d'erreur à $K = 1$ est toujours égal à zéro pour l'échantillon de formation. C'est parce que le point le plus proche de tout point

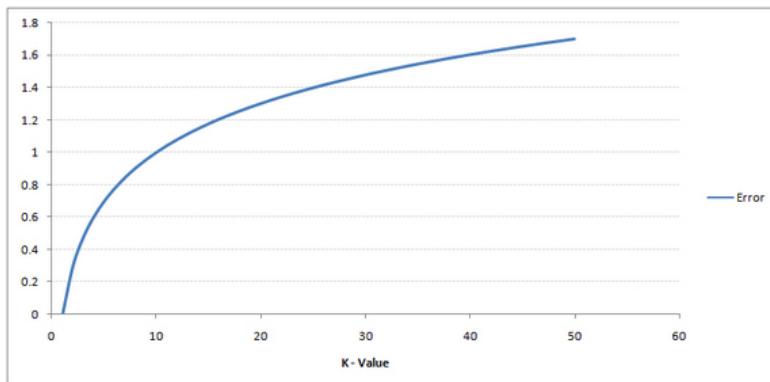


FIGURE 3.20 – Courbe du taux d'erreur d'entraînement avec une valeur variable de "K".

de données d'entraînement est lui-même, ce qui fait que la prédiction est toujours exacte avec $K = 1$. Si la courbe d'erreur de validation avait été similaire, notre choix de "K" aurait été de "1". La [Figure 3.21] illustre la courbe d'erreur de validation avec une valeur variable de K dans la même expérience.

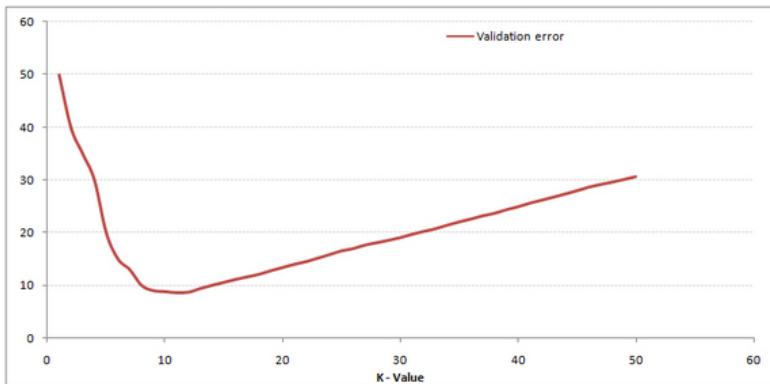


FIGURE 3.21 – Courbe d'erreur de validation avec une valeur variable de "K".

Nous pouvons constater dans la [Figure 3.21] qu'à $K = 1$, nous dépassions les limites. Par conséquent, le taux d'erreur diminue au départ et atteint un minimum. Après le point de minima, il augmente ensuite avec l'augmentation de "K". Pour obtenir la valeur optimale de "K", nous pouvons séparer l'entraînement et la validation de l'ensemble de données initial, ce que nous allons montrer plus tard dans la phase de tests. Puis tracer la courbe d'erreur de validation pour obtenir la valeur optimale de "K". Cette valeur de "K" doit être utilisée pour toutes les prédictions.

3.6 Mesure de similarité

Un point important dans la détection des cas similaires est la détermination de la similarité entre le cas présenté et l'expérience dans la base de cas. Nous avons besoin d'une fonction d'évaluation qui nous donne une mesure de la similarité entre ces deux cas. Cette fonction d'évaluation réduit chaque cas de sa description à une mesure de similarité numérique *sim*. Ces mesures de similarité montrent la relation avec d'autres cas dans la base de cas (Petra Perner, 2008 [62]).

La similarité peut se présenter de cinq types différents (Smith & Linda B, 1989 [63]) :

- **Similarité Générale** : c'est une relation globale qui inclut toutes les autres relations de similarité.
- **Similarité et Identité** : elles sont des relations qui prennent en compte toutes les propriétés des objets à la fois et dont aucune partie n'est laissée de côté. Une boule rouge et une boule rouge sont similaires, une boule rouge et une voiture rouge sont dissemblables. L'identité décrit des objets qui ne sont pas significativement différents. Toutes les boules rouges sont identiques et, par conséquent, elles sont également similaires. La similarité contient l'identité et est un concept plus général.
- **Similarité Partielles et Identité Partielle** : ces deux relations comparent les parties significatives des objets. Un aspect ou un attribut peut être marqué. La similarité partielle et l'identité partielle sont différentes en ce qui concerne le degré de similarité. Une boule rouge et un cube rose sont partiellement similaires, mais une boule rouge et un cube rouge sont partiellement identiques.

Les relations de similarité décrites sont en rapport à bien des égards. L'identité et la similarité sont des relations non spécifiées entre des objets. L'identité et la similarité partielles sont des relations entre les propriétés individuelles des objets. La similarité et l'identité sont deux concepts qui dépendent fortement du contexte. Le contexte définit les attributs essentiels des objets qui sont pris en considération lors de la détermination de la similarité. Un objet "balle rouge" peut être similaire à un objet "chaise rouge" en raison de la couleur "rouge". Cependant, les objets "balle" et "chaise" ne sont pas similaires. Ces attributs peuvent être pertinents, selon qu'on leur accorde la priorité ou qu'on les met en évidence dans le problème considéré (Petra Perner, 2008 [62]). Dans notre système puisque les caractéristiques extraites par le CNN prennent connaissance de tous les pixels se trouvant dans les images en entrée qui sont ces propriétés, nous pouvons constaté que nous allons avoir des explications d'une prédiction à base de la similarité et l'identité.

3.6.0.1 Distances de mesures de similarité

Une mesure de distance bien connue est la **métrique de Minkowski** :

$$d_{minkowski} = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}} \quad (3.6)$$

Le choix du paramètre p dépend de l'importance que l'on accorde aux différences dans la sommation. Si nous choisissons $p = 2$, alors nous accordons une importance particulière aux grandes différences dans les observations. La mesure est invariante aux translations et aux transformations linéaires orthogonales (rotation et réflexion). Elle est appelée **Distance Euclidienne** :

$$d_{euclidienne} = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (3.7)$$

Si nous choisissons $p = 1$, la mesure devient insensible aux valeurs aberrantes, puisque les grandes et les petites différences sont traitées de manière égale. Cette mesure est également appelée la **Métrique de Manhattan** :

$$d_{manhattan} = \sum_{i=1}^n |x_i - y_i| \quad (3.8)$$

3.7 Conclusion

Nous avons illustré dans ce chapitre la définition des RNA, leurs structure, et leurs types. Nous avons de même vu un type bien spécifique de ses réseaux, les CNNs, qui sont utilisés dans la classification des images et qui se base sur des fonctions de convolutions à base de filtre pour une extraction de caractéristiques importantes de nos données. Nous avons présenté encore une méthode de classification, les KNNs, qui se base sur le calcul de la mesure de similarité entre nos données pour trouver un cas plus proche et plus similaire à notre requête.

Apprentissage des réseaux de neurones

Les réseaux de neurones artificiels sont souvent considérés comme des boites noires qui contient une information à apprendre et à mémoriser. Lors du choix de la boite noire, à son démarrage elle est vide et ne contient aucune information ou connaissance sur le concept auquel elle doit s'adapter, c'est pour cela qu'un apprentissage est nécessaire. Les réseaux de neurones, qui sont des boites noires, doivent aussi subir ce processus.

Définition : est une phase de leurs développement durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré. L'apprentissage neuronal fait appel à des exemples de comportements. (Claude Touzet, 1992[9])

4.1 Principes

L'apprentissage des réseaux de neurones artificiels consiste à adapter ses différents paramètres (poids) d'après un algorithme itératif d'ajustement ou d'adaptation lui permettant de prendre en considération toutes les données (exemples) qui lui sont fournies à son entrée et ainsi ajuster ses paramètres pour trouver le juste milieu permettant de prendre en charge n'importe quel exemple ou donnée apparaissant à son entrée provenant de son environnement. Les algorithmes d'apprentissages donnent des meilleurs résultats lorsqu'on leur fournit des exemples multiples et variés, ainsi le réseau peut assimiler toutes les connaissances.

Ce processus alors consiste en l'entraînement du réseau en lui présentant des données et lui confiant la tache de modifier ses poids de telle sorte que l'on trouve la sortie désirée. Dans un premier temps l'algorithme consiste à propager les entrées vers les couches qui se suivent dans le réseau jusqu'à obtenir une prédiction calculée par le réseau. Ensuite la sortie calculée est comparée à la sortie réelle déjà connue (désirée). Les poids alors sont modifiés de telle sorte qu'à la prochaine itération, l'erreur commise entre la sortie calculée et la sortie réelle soit minimisée. On rétro-propage l'erreur commise vers l'arrière jusqu'à la couche d'entrée pour calculer la contribution de cette erreur sur la modification de

chacun des poids. Dans les types d'apprentissage supervisé, ce processus est défini comme un problème d'optimisation qui vise à trouver les coefficients du réseau minimisant une fonction d'erreur Globale.

4.2 Les fonctions d'erreur

La fonction d'erreur permet de mesurer l'écart entre les sorties réelles présentées dans le vecteur d'entraînement Y et les sorties prédites par le réseaux Y' . Il existe plusieurs fonctions d'erreurs possibles qui sont classées en deux classes selon le type de problème à résoudre : Classification ou Régression. Dans la classification, la tâche consiste à prédire les probabilités respectives de toutes les classes auxquelles le problème est confronté. Dans la régression, au contraire, la tâche consiste à prédire la valeur continue concernant un ensemble donné de caractéristiques indépendantes pour l'algorithme d'apprentissage (Sparsh Gupta, 2020 [65])

La fonction Coût et la fonction Perte (ou erreur) se réfèrent au même contexte. La fonction de coût est une fonction qui est calculée comme la moyenne de toutes les valeurs de la fonction de perte. Alors que la fonction de perte est calculée pour chaque échantillon de sortie par rapport à sa valeur réelle.

La fonction de perte est directement liée aux prédictions du modèle construit. Ainsi, si la valeur de la fonction de perte est inférieure, le modèle fournira de bons résultats. La fonction de perte, ou plutôt la fonction de coût qui est utilisée pour évaluer la performance du modèle, doit être réduite au minimum afin d'améliorer sa performance.

4.2.1 Erreurs de classification

4.2.1.1 La perte d'entropie croisée (Binary Cross-Entropy Loss / Log Loss) :

La perte d'entropie croisée diminue à mesure que la probabilité prévue converge vers l'étiquette réelle. Elle mesure la performance d'un modèle de classification dont la sortie prédite est une valeur de probabilité comprise entre 0 et 1. Quand on est face à un problème binaire (2 Classes à prédire), cette fonction est définie comme suit :

$$L = -\frac{1}{m} \sum_{i=1}^m (y_i \cdot \log(y'_i) + (1 - y_i) \cdot \log(1 - y'_i)) \quad (4.1)$$

Et quand le nombre de classe est supérieur à 2, l'équation devient :

$$L = -\frac{1}{m} \sum_{i=1}^m Y_i \cdot \log(Y'_i) \quad (4.2)$$

ou m est le nombre d'échantillons d'entraînement.

4.2.1.2 La perte de charnière (Hinge Loss) :

La deuxième fonction de perte utilisée pour les problèmes de classification et une alternative à la fonction de perte d'entropie croisée est la perte de charnière (Sparsh Gupta, 2020 [65]), principalement développée pour l'évaluation du modèle de machine à vecteur de supports (Support Vector Machine - SVM, en Anglais) de l'équation suivante :

$$L = \max(0, 1 - Y * Y') \quad (4.3)$$

La perte de charnière pénalise non seulement les mauvaises prédictions, mais aussi les bonnes prédictions qui ne sont pas fiables. Elle est principalement utilisée avec les classificateurs SVM dont les étiquettes de classe sont -1 et 1.

4.2.2 Erreurs de Regression

4.2.2.1 La fonction de perte quadratique moyenne (Mean Square Error / Quadratic Loss / L2 Loss) :

La fonction de perte quadratique moyenne (MSE) est définie comme la moyenne des différences au carré entre la valeur réelle et la valeur prédite, définie comme suite :

$$MSE = \frac{1}{m} \sum_{i=1}^m (Y'_i - Y_i)^2 \quad (4.4)$$

La fonction de coût correspondante est la moyenne de ces erreurs au carré (MSE). La fonction de perte MSE pénalise le modèle pour les erreurs importantes en les élevant au carré et cette propriété rend la fonction de coût de la MSE moins robuste aux valeurs aberrantes. Par conséquent, elle ne doit pas être utilisée si les données sont sujettes à de nombreuses aberrations (données distantes des autres observations).

4.2.2.2 La fonction de perte absolue moyenne (Mean Absolute Error / L1 Loss) :

La fonction de perte absolue moyenne (MAE) est définie comme la moyenne des différences absolues entre la valeur réelle et la valeur prévue. Elle mesure l'ampleur moyenne des erreurs dans un ensemble de prédictions, sans tenir compte de leurs directions. Elle est définie comme suit :

$$MAE = \frac{\sum_{i=1}^m |Y'_i - Y_i|}{m} \quad (4.5)$$

La fonction de coût correspondante est la moyenne de ces erreurs absolues (MAE). La fonction de perte MAE est plus robuste aux valeurs aberrantes que la fonction de perte

MSE. Elle doit donc être utilisée si les données sont sujettes à de nombreuses valeurs aberrantes (Sparsh Gupta, 2020[65])

Nous illustrons les différents graphes de ces fonctions dans la [Figure 4.1].

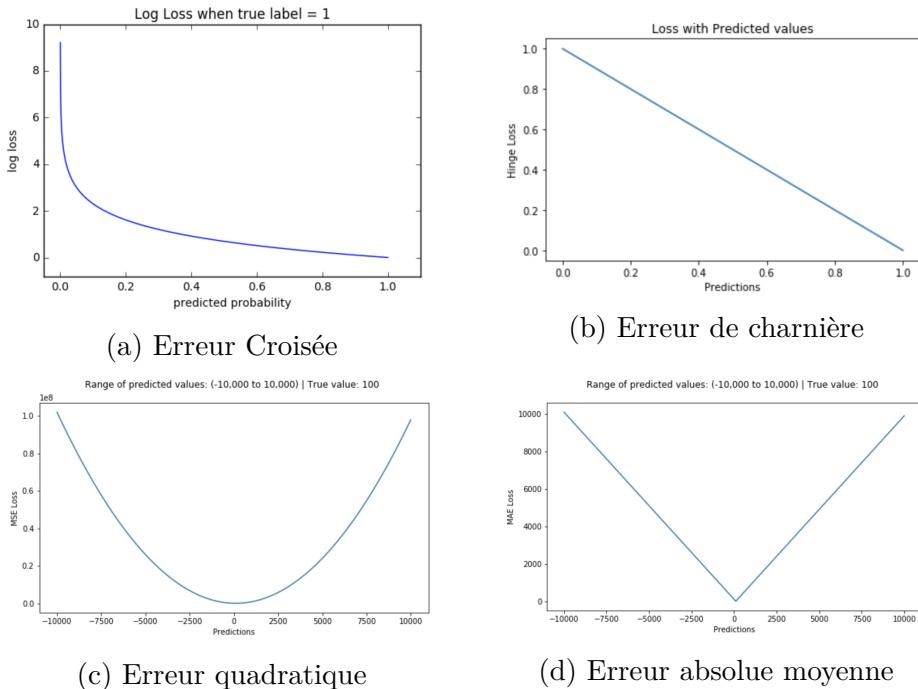


FIGURE 4.1 – Différentes formes de la fonction d’erreur pour la classification et la régression.

Maintenant que nous avons définis les différentes fonctions d’erreurs, nous avons besoins de choisir l’algorithme d’optimisation qui permet d’estimer les poids du réseau pour lesquels la fonctions de coûts est minimale.

4.3 Algorithmes d’optimisation

Les algorithmes d’optimisation utilisés nécessitent que la fonction de coût soit dérivable par rapport aux poids. Le principe de ces algorithmes est de se placer en un point initial, de trouver une direction de descente du coût dans l’espace des paramètres, puis de se déplacer d’un pas suivant cette direction. On atteint un nouveau point et l’on recommence la procédure.

Les techniques d’optimisation deviennent la pièce maîtresse des algorithmes d’apprentissage profond lorsque l’on attend des résultats meilleurs et plus rapides des réseaux de neurones. Le choix entre ces techniques d’optimisation peut faire une énorme différence sur le temps passé pour obtenir un réseau performant.

Dans cette section, nous allons présenter quelques algorithmes d'optimisation utilisés les réseaux de neurones profonds. Il existe plusieurs algorithmes d'apprentissage bien connus (Somnath Paul, 2020[66]), que nous pouvons regrouper comme suit :

- **Algorithme d'apprentissage basé sur le moment présent**
 - Descente en gradient vanille (GD)
 - Descente en gradient basée sur le momentum
 - Descente accélérée du gradient de Nesterov (NAG)
- **Algorithme d'apprentissage basé sur l'apprentissage par lots**
 - Mise à jour stochastique
 - Mise à jour du mini-batch
- **Algorithme d'apprentissage basé sur le taux d'apprentissage adaptatif**
 - AdaGrad
 - Adadelta
 - RMS Prop
 - Adam (un mélange de RMS Prop et de GD basé sur Momentum)

Nous allons maintenant présenter l'un des algorithmes les plus utilisés et plus simple à mettre en place, la descente du gradient.

4.3.1 Algorithme de descente du gradient

La descente en gradient est un algorithme d'optimisation qui est utilisé lors de la formation d'un modèle d'apprentissage (P.J. Werbos, 1990 [67]). Il est basé sur une fonction convexe et ajuste ses paramètres de manière itérative pour réduire une fonction donnée à son minimum local. C'est un algorithme d'optimisation permettant de trouver un minimum local d'une fonction différentiable, il est simplement utilisé pour trouver les valeurs des paramètres d'une fonction (coefficients) qui minimisent une fonction de coût autant que possible.

Les valeurs du paramètre initial sont définies et, à partir de là, la descente de gradient utilise le calcul pour ajuster itérativement les valeurs afin qu'elles minimisent la fonction de coût donnée. Le poids est initialisé en utilisant certaines stratégies d'initialisation et est mis à jour à chaque époque selon l'équation de mise à jour (M. Minoux, 1984 [68]).

L'équation suivante calcule le gradient de la fonction de coût $J(\theta)$, par rapport aux paramètres/poids θ pour l'ensemble des données de formation :

$$\theta'_j = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j} \quad (4.6)$$

Où α est le taux d'apprentissage, θ' est le nouveau paramètre , θ est l'ancien paramètre, et $\frac{\partial J(\theta)}{\partial \theta_j}$ est le dérivé d'ordre 1 qui définit le gradient (l'orientation) de la fonction coût J à la valeur du paramètre θ_j , qui mesure combien la sortie d'une fonction change si les entrées sont un peu modifiées.

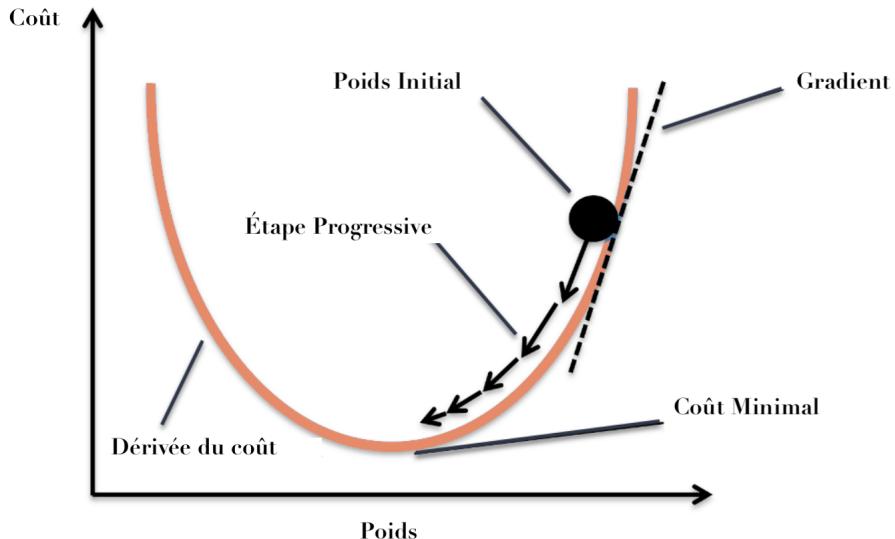


FIGURE 4.2 – Algorithme de la descente du gradient.

L'objectif ici est d'arriver au bas de notre graphe (Coût vs Poids), ou à un point où nous ne pouvons plus descendre, un minimum local [Figure 4.2].

Un paramètre important dans la descente du gradient, le taux d'apprentissage α , est un hyper-paramètre à bien déterminer. L'importance des pas de descente dans la direction du minimum local est déterminée par le taux d'apprentissage, qui détermine la vitesse ou le ralentissement de notre progression vers les poids optimaux [Figure 4.3].

Pour que la descente de la pente atteigne le minimum local, nous devons fixer le taux d'apprentissage à une valeur appropriée, qui ne soit ni trop basse ni trop élevée. C'est important car si les pas qu'il fait sont trop grands, il risque de ne pas atteindre le minimum local car il rebondit entre la fonction convexe de la descente de la pente (image de gauche [Figure 4.3]). Si nous réglons le taux d'apprentissage à une valeur très faible, la descente de la pente finira par atteindre le minimum local mais cela peut prendre un certain temps (image de droite [Figure 4.3]). Le taux d'apprentissage ne doit donc jamais être trop élevé ou trop bas pour cette raison. Nous pouvons vérifier si notre taux d'apprentissage est bon en le traçant sur un graphique (Nagesh Singh, 2020 [69]).

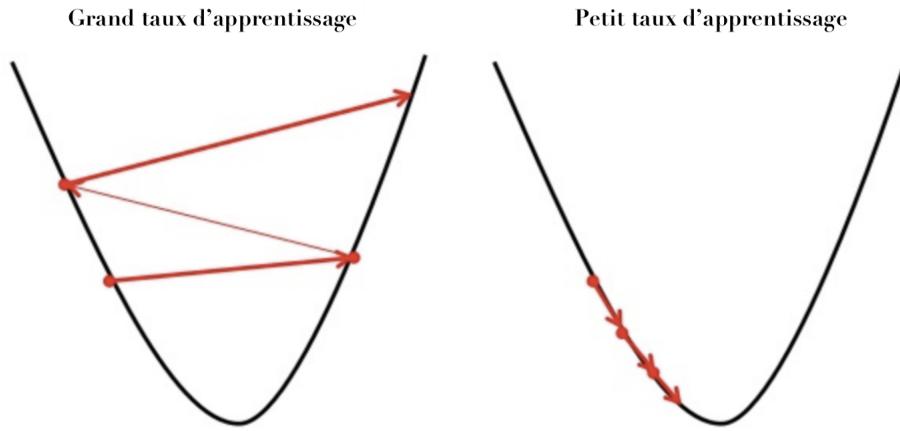


FIGURE 4.3 – Influence du taux d'apprentissage dans la descente du gradient.

4.3.2 Autres algorithmes d'optimisation

Il existe d'autres types d'algorithme que nous avons cités précédemment, nous allons donner une présentation brève pour les plus utilisés dans les réseaux de neurones profonds (Algorithmia, 2018 [70])

4.3.2.1 AdaGrad

Adagrad (Duchi et al. 2011 [71]) est un algorithme d'optimisation par gradient qui fait exactement cela : Il adapte le taux d'apprentissage aux paramètres, en effectuant des mises à jour plus petites (c'est-à-dire des taux d'apprentissage faibles) pour les paramètres associés à des fonctionnalités fréquentes, et des mises à jour plus importantes (c'est-à-dire des taux d'apprentissage élevés) pour les paramètres associés à des fonctionnalités peu fréquentes. Pour cette raison, il est bien adapté au traitement de données peu nombreuses. (Dean et al. 2012 [72]) ont constaté qu'Adagrad a considérablement amélioré la robustesse de la descente de gradient stochastique et l'a utilisée pour former les réseaux neuronaux à grande échelle de Google, qui ont notamment appris à reconnaître les chats dans les vidéos Youtube. De plus, (Pennington et al. 2014 [73]) ont utilisé Adagrad pour entraîner l'incorporation de mots dans GloVe, car les mots peu fréquents nécessitent des mises à jour beaucoup plus importantes que les mots fréquents.

4.3.2.2 Adadelta

Adadelta (Zeiler, 2014 [74]) est une extension d'Adagrad qui cherche à réduire son taux d'apprentissage agressif et monotone. Au lieu d'accumuler tous les gradients passés au carré, Adadelta limite la fenêtre des gradients passés accumulés à une taille fixe. Au lieu de stocker de manière inefficace les gradients carrés précédents, la somme des gradients est définie récursivement comme une moyenne décroissante de tous les gradients carrés

passés. La moyenne courante dépend alors que sur la moyenne précédente et le gradient actuel.

4.3.2.3 RMS Prop

RMSprop est une méthode inédite de taux d'apprentissage adaptatif (Geoff Hinton, [76]). RMSprop et Adadelta ont tous deux été développés indépendamment à peu près à la même époque, en raison de la nécessité de résoudre le problème de la diminution radicale des taux d'apprentissage d'Adagrad. RMSprop est en fait identique au premier vecteur de mise à jour d'Adadelta. RMSprop divise également le taux d'apprentissage par une moyenne de gradients carrés qui décroît de façon exponentielle.

4.3.2.4 Adam

L'estimation du moment adaptatif (Adam) est une autre méthode qui calcule les taux d'apprentissage adaptatif pour chaque paramètre (Kingma et al. 2015 [77]). En plus de stocker une moyenne à décroissance exponentielle des gradients carrés passés comme Adadelta et RMSprop, Adam conserve également une moyenne des gradients passés qui décroît de façon exponentielle.

La situation est similaire à celle de l'élan. Alors que l'élan peut être vu comme une balle descendant une pente, Adam se comporte comme une balle lourde avec frottement, qui préfère donc des minima plats dans la surface d'erreur (Heusel et al. 2017 [78]).

Après avoir définis les algorithmes d'optimisation qui permettent d'entraîner un réseaux de neurone en ajustant ses poids en se basant sur le calcul du gradient, nous allons présenter les façons dont ses gradients sont calculés efficacement au sein du processus d'apprentissage, et comment les variables sont stockées au sein d'un RNA.

4.4 Propagation Avant

La propagation avant (ou passage direct) fait référence au calcul et au stockage de variables intermédiaires (y compris les sorties) pour un réseau de neurones afin de passer de la couche d'entrée à la couche de sortie (Zhang et al. 2019 [79]). Nous traiterons un exemple d'un réseau de neurones avec une couche cachée, ne contenant pas le biais, et $x \in \mathbb{R}^d$ (espace de d-uplets réels), alors notre variable intermédiaire est comme suite :

$$z = W^{(1)}x, \quad (4.7)$$

où $W^{(1)} \in \mathbb{R}^{h \times d}$ est le paramètre de poids de la couche cachée. Après avoir exécuté la variable intermédiaire $z \in \mathbb{R}^h$ par la fonction d'activation ϕ , nous obtenons notre vecteur

d'activation caché de longueur h ,

$$h = \phi(z). \quad (4.8)$$

La variable cachée h est également une variable intermédiaire. En supposant que les paramètres de la couche de sortie ne possèdent qu'un poids de $W^{(2)} \in \mathbb{R}^{q \times h}$, nous pouvons obtenir une variable de la couche de sortie avec un vecteur de longueur q :

$$o = W^{(2)}h. \quad (4.9)$$

En supposant que la fonction de perte est l et que l'étiquette d'exemple est y , nous pouvons alors calculer le terme de perte pour un seul exemple de données,

$$L = l(o, y). \quad (4.10)$$

Selon la définition de la régularisation¹ L_2 (Zhang et al. 2019 [79]), compte tenu de l'hyperparamètre λ , le terme de régularisation est

$$s = \frac{\lambda}{2}(\|W^{(1)}\|_F^2 + \|W^{(2)}\|_F^2) \quad (4.11)$$

où la norme de la matrice est simplement la norme L_2 appliquée après avoir aplati la matrice en un vecteur. Enfin, la perte régularisée du modèle sur un exemple de données est

$$J = L + s. \quad (4.12)$$

ou J fait référence à la fonction objective utilisée lors de la phase d'optimisation.

Le tracé de graphiques de calcul nous aide à visualiser les dépendances des opérateurs et des variables dans le calcul. La [Figure 4.4] contient le graphique associé au réseau simple décrit ci-dessus, où les carrés représentent les variables et les cercles les opérateurs. Le coin inférieur gauche représente l'entrée et le coin supérieur droit, la sortie. Notons que les directions des flèches (qui illustrent le flux de données) sont principalement vers la droite et vers le haut.

1. La régularisation peut être définie comme toute modification que nous apportons à un algorithme d'apprentissage qui vise à réduire son erreur de généralisation mais pas son erreur d'entraînement. Cette régularisation est souvent effectuée en imposant des contraintes supplémentaires à un modèle d'apprentissage machine, comme l'ajout de restrictions sur les valeurs des paramètres ou l'ajout de termes supplémentaires dans la fonction objectif qui peuvent être considérés comme correspondant à une contrainte souple sur les valeurs des paramètres. Il existe plusieurs types de régularisation dont on a cité « la régularisation de paramètres L2 ».

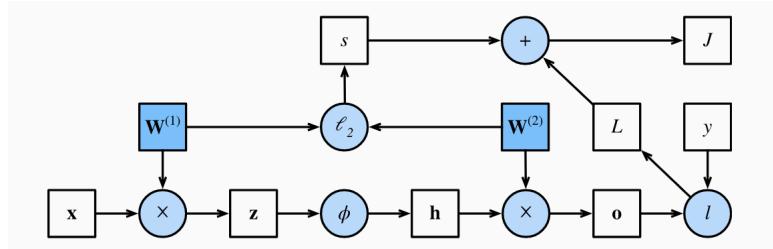


FIGURE 4.4 – Graphique de calcul de la propagation avant.

4.5 Rétropropagation

La rétropropagation est la méthode de calcul du gradient des paramètres des réseaux de neurones (Zhang et al. 2019 [79]). La méthode parcourt le réseau dans l’ordre inverse, de la couche de sortie à la couche d’entrée, selon la règle de la chaîne du calcul. L’algorithme stocke toutes les variables intermédiaires (dérivées partielles) nécessaires au calcul du gradient pour certains paramètres. Supposons que nous ayons les fonctions $Y = f(X)$ et $Z = g(Y)$, dans lesquelles l’entrée et la sortie X, Y, Z sont des tenseurs de formes arbitraires. En utilisant la règle de la chaîne (formule de dérivation d’une composition de deux fonctions dérivables), nous pouvons calculer la dérivée de Z par rapport à X via

$$\frac{\partial Z}{\partial X} = \text{prod}\left(\frac{\partial Z}{\partial Y}, \frac{\partial Y}{\partial X}\right) \quad (4.13)$$

Ici, nous utilisons l’opérateur *prod* pour multiplier ses arguments après que les opérations nécessaires, telles que la transposition et l’échange des positions d’entrée, ont été effectuées. Pour les vecteurs, il s’agit simplement d’une multiplication matricielle. Pour les tenseurs de dimension supérieure, nous utilisons la contrepartie appropriée.

Rappelons que les paramètres du réseau simple avec une couche cachée, dont le graphe de calcul se trouve à la [Figure 4.4], sont $W^{(1)}$ et $W^{(2)}$. L’objectif de la rétropropagation est de calculer les gradients $\frac{\partial J}{\partial W^{(1)}}$ et $\frac{\partial J}{\partial W^{(2)}}$. Pour ce faire, nous appliquons la règle de la chaîne et calculons, à tour de rôle, le gradient de chaque variable et paramètre intermédiaire. L’ordre des calculs est inversé par rapport à ceux effectués en propagation avant, puisque nous devons commencer par le résultat du graphe de calcul et progresser vers les paramètres. La première étape consiste à calculer les gradients de la fonction objectif $J = L + s$ par rapport au terme de perte L et au terme de régularisation s .

$$\frac{\partial J}{\partial L} = 1 \text{ et } \frac{\partial J}{\partial s} = 1 \quad (4.14)$$

Ensuite, nous calculons le gradient de la fonction objectif par rapport à la variable de la couche de sortie o selon la règle de la chaîne :

$$\frac{\partial J}{\partial o} = \text{prod}\left(\frac{\partial J}{\partial L}, \frac{\partial L}{\partial o}\right) = \frac{\partial L}{\partial o} \in \mathbb{R}^q \quad (4.15)$$

Ensuite, nous calculons les gradients du terme de régularisation par rapport aux deux paramètres :

$$\frac{\partial s}{\partial W^{(1)}} = \lambda W^{(1)} \text{ et } \frac{\partial s}{\partial W^{(2)}} = \lambda W^{(2)} \quad (4.16)$$

Nous sommes maintenant en mesure de calculer le gradient $\frac{\partial J}{\partial W^{(2)}} \in \mathbb{R}^{(q \times h)}$ des paramètres du modèle les plus proches de la couche de sortie. L'utilisation de la règle de la chaîne donne des résultats :

$$\frac{\partial J}{\partial W^{(2)}} = prod\left(\frac{\partial J}{\partial o}, \frac{\partial o}{\partial W^{(2)}}\right) + prod\left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial W^{(2)}}\right) = \frac{\partial J}{\partial o} h^T + \lambda W^{(2)} \quad (4.17)$$

Pour obtenir le gradient par rapport à $W^{(1)}$, nous devons continuer la rétropropagation le long de la couche de sortie jusqu'à la couche cachée. Le gradient par rapport aux sorties de la couche cachée $\frac{\partial J}{\partial h} \in \mathbb{R}^h$ est donné par

$$\frac{\partial J}{\partial h} = prod\left(\frac{\partial J}{\partial o}, \frac{\partial o}{\partial h}\right) = W^{(2)^T} \frac{\partial J}{\partial o} \quad (4.18)$$

Puisque la fonction d'activation θ s'applique élément par élément, le calcul du gradient $\frac{\partial J}{\partial z} \in \mathbb{R}^h$ de la variable intermédiaire z exige que nous utilisions l'opérateur de multiplication élément par élément, que nous désignons par \odot :

$$\frac{\partial J}{\partial z} = prod\left(\frac{\partial J}{\partial h}, \frac{\partial h}{\partial z}\right) = \frac{\partial J}{\partial h} \odot \theta'(z) \quad (4.19)$$

Enfin, nous pouvons obtenir le gradient $\frac{\partial J}{\partial W^{(1)}} \in \mathbb{R}^{h \times d}$ des paramètres du modèle les plus proches de la couche d'entrée. Selon la règle de la chaîne, nous obtenons

$$\frac{\partial J}{\partial W^{(1)}} = prod\left(\frac{\partial J}{\partial z}, \frac{\partial z}{\partial W^{(1)}}\right) + prod\left(\frac{\partial J}{\partial s}, \frac{\partial s}{\partial W^{(1)}}\right) = \frac{\partial J}{\partial z} x^T + \lambda W^{(1)} \quad (4.20)$$

4.6 Relation Propagation Avant et Arrière

Lors de la formation des réseaux de neurones, la propagation vers l'avant et vers l'arrière (rétropropagation) dépend l'une de l'autre (Zhang et al. 2019 [79]). En particulier, pour la propagation vers l'avant, nous parcourons le graphe de calcul dans la direction des dépendances et calculons toutes les variables sur son chemin. Celles-ci sont ensuite utilisées pour la rétropropagation où l'ordre de calcul sur le graphe est inversé.

Prenons l'exemple du réseau simple [Figure 4.4]. D'une part, le calcul du terme de régularisation (4.11) pendant la propagation vers l'avant dépend des valeurs actuelles des paramètres du modèle $W^{(1)}$ et $W^{(2)}$. Ces valeurs sont données par l'algorithme d'optimisation en fonction de la rétropropagation dans la dernière itération. D'autre part, le calcul du gradient du paramètre (4.17) pendant la rétropropagation dépend de la valeur

actuelle de la variable cachée h , qui est donnée par la propagation vers l'avant.

Par conséquent, lors de la formation des réseaux de neurones, après l'initialisation des paramètres du modèle, nous alternons la propagation vers l'avant et la rétropropagation, en mettant à jour les paramètres du modèle à l'aide des gradients donnés par la rétropropagation. Il est à noter que la rétropropagation réutilise les valeurs intermédiaires stockées de la propagation avant pour éviter les calculs en double. Une des conséquences est que nous devons conserver les valeurs intermédiaires jusqu'à ce que la rétropropagation soit terminée. C'est également l'une des raisons pour lesquelles l'entraînement nécessite beaucoup plus de mémoire que la simple prédiction. En outre, la taille de ces valeurs intermédiaires est à peu près proportionnelle au nombre de couches du réseau et à la taille du lot. Ainsi, l'entraînement de réseaux plus profonds utilisant des lots de taille plus importante conduit plus facilement à des erreurs de mémoire (Zhang et al. 2019 [79]).

Nous pouvons alors faire un résumé comme suit :

- La propagation directe calcule et stocke séquentiellement des variables intermédiaires dans le graphe de calcul défini par le réseau de neurones. Elle va de la couche d'entrée à la couche de sortie.
- La rétropropagation calcule et stocke séquentiellement les gradients des variables et paramètres intermédiaires dans le réseau de neurones dans l'ordre inverse.
- Lors de l'apprentissage des modèles d'apprentissage profond, la propagation vers l'avant et la propagation vers l'arrière sont interdépendantes.
- L'apprentissage nécessite beaucoup plus de mémoire que la prédiction.

4.7 Conclusion

Dans ce chapitre nous avons abordé le principe de l'apprentissage des RNA les notions qui lui sont nécessaires, les fonctions d'erreurs qui nous permettent de mesurer l'écart entre les prédictions du RNA et les sorties réelles présentées lors de l'entraînement, les algorithmes d'optimisations qui permettent de minimiser la fonction d'erreurs pour atteindre une bonne précision de notre réseau à base de calcul du gradient, et les deux principes fondamentaux de l'apprentissage (propagation avant et arrière) qui font références au calcul et au stockage des variables intermédiaires ainsi que la façon de calcul des gradients dans un RNA pour sa bonne formation.

Systèmes à boites noires

Bien que les algorithmes d'apprentissage automatique (AA) semblent puissants en termes de résultats et de prévisions, ils ont leurs propres limites et pièges. La plus importante est l'opacité ou le manque de transparence (Mengnan et al. 2019 [13]), qui caractérise par nature les systèmes à boîtes noires de l'AA. Cela signifie que la logique interne de ces systèmes et le fonctionnement interne sont cachés à l'utilisateur, ce qui est un désavantage sérieux car il empêche un humain, expert, ou non expert, parce qu'il est en mesure de vérifier, d'interpréter et de comprendre le raisonnement du système et la façon dont des décisions particulières sont prises (Montavon et al., 2017 [14]). En d'autres termes, tout système suffisamment complexe agit en tant qu'une boîte noire lorsqu'il devient plus facile d'expérimenter que de comprendre (Golovin et al., 2017 [15]). De nombreux modèles d'AA, y compris les modèles les plus performants dans divers domaines, appartiennent à ce groupe de systèmes à boîtes noires (Rudin, 2018 [16]), tels que les réseaux de neurones artificiels profonds.

5.1 Modèles paramétrique et non-paramétriques

L'approche la plus directe de l'inférence statistique consiste à considérer que nous pouvons modéliser les données en utilisant une forme fermée de fonction pouvant contenir un certain nombre de paramètres (poids) ajustables que nous pouvons estimer, de sorte que le modèle peut nous donner la meilleure explication possible de nos données (STATISTICA, 2016 [20]).

Prenons l'exemple d'une problématique de régression linéaire dans laquelle nous cherchons à approcher une variable cible, t à l'aide d'une fonction linéaire de la variable d'entrée, x . La fonction mathématique utilisée pour modéliser cette relation est donnée par une transformation linéaire f à deux paramètres, l'ordonnée à l'origine, a et la pente, b , comme suit :

$$t = f(x) = a + bx \tag{5.1}$$

Notre tâche consiste à trouver des valeurs de a et de b qui vont permettre de faire le lien entre une entrée x et la variable t .

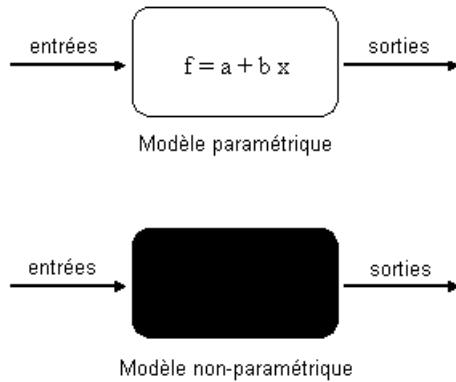


FIGURE 5.1 – Modèle paramétrique et modèle non-paramétrique.

la [Figure 5.1] montre la différence entre les modèles paramétriques et les modèles non-paramétriques. Dans les modèles paramétriques, la relation entre les entrées et les sorties s’exprime par une fonction mathématique de forme fermée. Par opposition, dans les modèles non-paramétriques, la relation entre les entrées et les sorties est pilotée par un approximateur (comme un réseau de neurones artificiels) que nous ne pouvons pas représenter par une fonction mathématique standard (STATISTICA, 2016 [20]).

Les modèles paramétriques reposent au sens strict sur l’hypothèse selon laquelle t est lié à x de manière connue a priori, ou peuvent être suffisamment bien approchées par une forme mathématique fermée, par exemple, une droite ou une fonction quadratique. Une fois que nous avons choisi la fonction mathématique, nous allons ajuster les paramètres du modèle supposé afin qu’ils prévoient au mieux t pour une instance donnée de x (STATISTICA, 2016 [20]).

Par opposition, les modèles non-paramétriques supposent que la véritable fonction sous-jacente qui gouverne la relation entre x et t n’est pas connue a priori, d’où le terme de *boîte noire (BN)*. Ils vont donc chercher une fonction mathématique (qui généralement ne possède pas une forme fermée) capable d’approcher convenablement la représentation de x et de t . Les réseaux de neurones artificiels et les fonctions polynomiales font partie de la famille des modèles non-paramétrique (STATISTICA, 2016 [20]).

5.2 Explication ou interprétation des modèles à BN

Des études sur les méthodes d’explication des modèles à boîtes noires ont mis en évidence une taxonomie permettant de classer les principaux problèmes posés par les algorithmes opaques. La plupart des méthodes étudiées sont appliquées à des algorithmes

basés sur des réseaux de neurones (Leilani et al., 2019 [19]). Les auteurs donnent un aperçu des méthodes qui expliquent les systèmes décisionnels basés sur des modèles d'apprentissage automatique opaques et obscurs. Leur classification examine quatre caractéristiques pour chaque méthode d'explication :

1. Le type de problème rencontré.
2. La capacité explicative utilisée pour ouvrir la boîte noire.
3. Le type de modèle de boîte noire qui peut être expliqué.
4. Le type de données d'entrée fournies au modèle de la boîte noire.

Ils divisent principalement les méthodes d'explication en fonction des types de problèmes rencontrés et identifient quatre groupes de méthodes d'explication :

- Les méthodes pour expliquer les modèles de boîtes noires.
- Les méthodes pour expliquer les résultats de la boîte noire.
- Les méthodes pour inspecter les boîtes noires.
- Les méthodes pour concevoir des boîtes transparentes.

À l'aide de leurs caractéristiques de classification et de ces définitions des problèmes, ils discutent et catégorisent les méthodes selon le type de capacité explicative adopté, le modèle de boîte noire "ouverte" et les données d'entrée. Leur but est d'examiner et de classer les principales architectures d'explication des boîtes noires, afin que leurs classifications puissent servir de guide pour identifier des problèmes et des approches similaires (Leilani et al., 2019 [19]).

Nous pouvons donner la formulation d'un problème de classification (Guidotti et al., 2018[18]), comme suit :

Un prédicteur, également appelé modèle ou classificateur, est une fonction $b : \mathcal{X}^m \rightarrow \mathcal{Y}$ où \mathcal{X}^m est l'espace des caractéristiques avec m correspondant au nombre de caractéristiques, et \mathcal{Y} est l'espace cible qui contient les différentes étiquettes (classes ou résultats) et identifie un concept sémantique. \mathcal{X} et \mathcal{Y} peuvent être un ensemble de booléens, d'entiers ou de chaînes de caractères. Autrement, un prédicteur b est la sortie d'une fonction d'apprentissage \mathcal{L}_b telle que $\mathcal{L}_b : (\mathcal{X}^{n \times m} \times \mathcal{Y}^n) \rightarrow (\mathcal{X}^m \rightarrow \mathcal{Y})$. L'apprenant \mathcal{L}_b prend comme entrée un ensemble de données $D = \{X, Y\}$ avec n échantillons où $X \in \mathcal{X}^{n \times m}$ et $Y \in \mathcal{Y}^n$ et renvoie le prédicteur b . Étant donné un ensemble de données dans l'espace de caractéristiques $x \in \mathcal{X}^m$, le prédicteur b peut être utilisé pour prédire la valeur cible \hat{y} , c'est-à-dire $b(x) = \hat{y}$.

5.2.1 Explication d'un modèle à boîte noire

Étant donné qu'un modèle à boîte noire résout un problème de classification, le problème d'explication de boîte noire consiste à fournir un modèle interprétable et transparent

capable d’imiter le comportement de la boîte noire et qui est également compréhensible par l’utilisateur. En d’autres termes, le modèle interprétable se rapprochant de la boîte noire doit être interprétable globalement (Guidotti et al., 2018[18]). En conséquence, nous dénions le problème d’explication du modèle de boîte noire comme suit :

Définition : étant donné un prédicteur à boîte noire b et un dataset $D = \{X, Y\}$, le problème d’explication d’un modèle à boîte noire consiste à trouver une fonction $f : (X^m \rightarrow Y) \times (X^{n \times m} \times Y^n) \rightarrow (X^m \rightarrow Y)$ qui prend comme entrée la boîte noire b et le dataset D , et renvoie un prédicteur compréhensible global, c_g i.e., $f(b, D) = c_g$, de telle sorte que c_g est capable d’imiter le comportement de b , et existe une fonction d’explication globale $\varepsilon_g : (X^m \rightarrow Y) \rightarrow \xi$, où ξ est l’ensemble regroupant les explications compréhensibles, qui peut dériver de c_g un ensemble d’explications $E \in \xi$ modélisant d’une manière compréhensible la logique derrière c_g i.e., $\varepsilon_g(c_g) = E$, (Guidotti et al., 2018[18]).

5.2.2 Explication des résultats d’une boîte noire

Le problème d’explication du résultat de la boîte noire consiste à fournir un résultat interprétable, c’est-à-dire une méthode pour fournir une explication du résultat de la boîte noire. En d’autres termes, le modèle interprétable doit renvoyer la prédiction accompagnée d’une explication des raisons de cette prédiction, c’est-à-dire que la prédiction ne peut être interprétée que localement. Il n’est pas nécessaire d’expliquer toute la logique derrière la boîte noire, mais seulement les raisons du choix d’un cas particulier (Guidotti et al., 2018[18]). Par conséquent, nous dénions le problème d’explication des résultats de la boîte noire comme suit :

Définition : étant donné un prédicteur à boîte noire b et un dataset $D = \{X, Y\}$, le problème d’explication du résultat d’un modèle à boîte noire consiste à trouver une fonction $f : (X^m \rightarrow Y) \times (X^{n \times m} \times Y^n) \rightarrow (X^m \rightarrow Y)$ qui prend comme entrée la boîte noire b et le dataset D , et renvoie un prédicteur compréhensible local , c_l i.e., $f(b, D) = c_l$, de telle sorte que c_l est capable d’imiter le comportement de b , et existe une fonction d’explication locale $\varepsilon_l : (X^m \rightarrow Y) \rightarrow \xi$ qui prend en entrée la boîte noire b , le modèle compréhensible local c_l , et une instance de donnée x avec des caractéristiques dans X^m , et renvoie une explication compréhensible $e \in \xi$ pour l’instance x i.e., $\varepsilon_l(b, c_l, x) = e$, (Guidotti et al., 2018[18]).

5.2.3 Problème d’inspection d’une boîte noire

Le problème d’inspection de la boîte noire consiste à fournir une représentation (visuelle ou textuelle) pour comprendre comment fonctionne le modèle à boîte noire ou pourquoi la boîte noire renvoie certaines prédictions plus susceptibles que d’autres (Guidotti et al., 2018[18]).

Définition : étant donné un prédicteur à boite noire b et un dataset $D = \{X, Y\}$, le problème d'inspection d'une boite noire consiste à trouver une fonction $f : (X \rightarrow Y) \times (X^{n \times n} \times Y^n) \rightarrow V$ qui prend en entrée la boite noire b et un dataset D , et renvoie une représentation visuelle du comportement de la boite noire, $f(b, D) = v$ avec V étant l'ensemble de toutes les représentations possibles, (Guidotti et al., 2018[18]).

5.2.4 Problème de conception de la boîte transparente

Compte tenu d'un problème de classification, le problème de conception de la boîte transparente consiste à fournir un modèle qui peut être interprété localement ou globalement de manière autonome (Guidotti et al., 2018[18]).

Définition : étant donné un dataset $D = \{X, Y\}$, le problème de conception de la boîte transparent consiste à trouver une fonction d'apprentissage $\mathcal{L}_c : (X^{n \times m} \times Y) \rightarrow (X^m \rightarrow Y)$ qui prend en entrée le dataset D et renvoie un prédicteur compréhensible (Locallement ou globallement) c , i.e., $\mathcal{L}_c(D) = c$. Cela implique qu'il existe une fonction d'explication locale ε_l ou une fonction d'explication globale ε_g qui prend comme entrée le prédicteur compréhensible c et renvoie une explication compréhensible par l'humain $e \in \xi$ ou un ensemble d'explications E , (Guidotti et al., 2018[18]).

Ainsi, selon nos dénis problématiques, dans cette étude, lorsque nous disons qu'une méthode est capable d'ouvrir la boîte noire, nous faisons référence à l'une des affirmations suivantes :

- i) elle explique le modèle,
- ii) elle explique le résultat,
- iii) elle peut inspecter la boîte noire en interne,
- iv) elle fournit une solution transparente.

5.3 Techniques d'interprétation des systèmes à BN

Les techniques d'apprentissage automatique interprétables peuvent généralement être regroupées en deux catégories : **l'interprétabilité intrinsèque** et **l'interprétabilité post-hoc**, selon le moment où l'interprétabilité est obtenue. L'interprétabilité intrinsèque est obtenue en construisant des modèles auto-explicatifs qui incorporent l'interprétabilité directement à leurs structures. En revanche, le modèle post-hoc nécessite la création d'un deuxième modèle pour expliquer un modèle existant (Mengnan et al., 2018 [13]).

La principale différence entre ces deux groupes réside dans le compromis entre l'exactitude du modèle et la fidélité de l'explication. Des modèles intrinsèquement interprétables

pourraient fournir des explications précises et non faussées, mais pourraient sacrifier dans une certaine mesure les performances de prédiction. Les modèles post-hoc sont limités dans leur nature approximative tout en conservant la précision du modèle sous-jacent intacte (Mengnan et al., 2018 [13]).

Sur la base de la catégorisation ci-dessus, nous distinguons deux types d’interprétabilité : l’**interprétabilité globale** et l’**interprétabilité locale**. L’interprétabilité globale signifie que les utilisateurs peuvent comprendre le fonctionnement global du modèle en inspectant les structures et les paramètres d’un modèle complexe, tandis que l’interprétabilité locale examine localement une prédiction individuelle d’un modèle, essayant de comprendre pourquoi le modèle prend la décision (Mengnan et al., 2018 [13]).

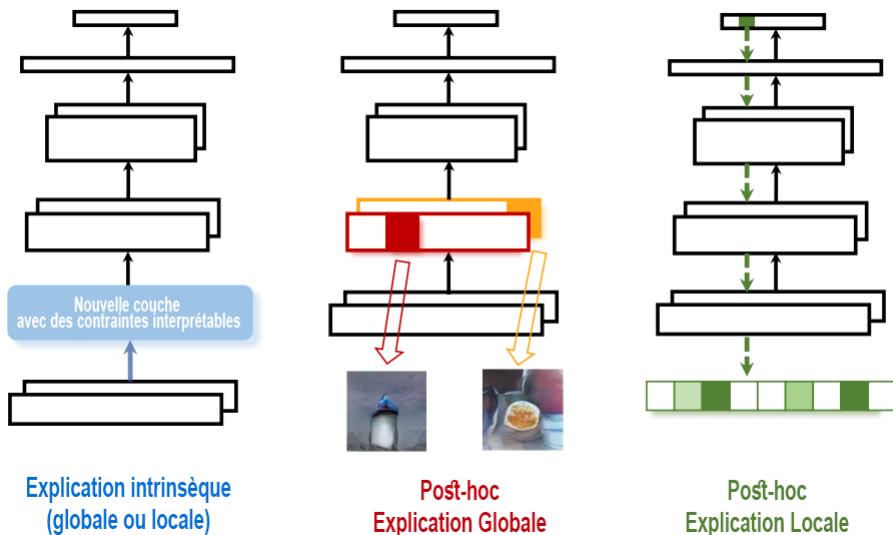


FIGURE 5.2 – Illustration de trois lignes de techniques d’apprentissage automatique interprétables : Explication intrinsèque, explication globale post-hoc d’un modèle et explication locale post-hoc d’une prédition.

Si l’on prend l’exemple d’un réseau de neurones profond de la [Figure 5.2], l’interprétabilité globale est obtenue en comprenant les représentations capturées par les neurones d’une couche intermédiaire, tandis que l’interprétabilité locale est obtenue en identifiant les contributions de chaque caractéristique dans une entrée spécifique à la prévision faite par le réseau. Ces deux types présentent des avantages différents. L’interprétabilité globale pourrait mettre en lumière les mécanismes de fonctionnement internes des modèles d’apprentissage automatiques et peut donc accroître leur transparence. L’interprétabilité locale aidera à découvrir les relations causales entre une entrée spécifique et la prédition correspondante du modèle. Ces deux méthodes aident les utilisateurs à faire confiance à un modèle et à une prédiction, respectivement (Mengnan et al., 2018 [13]).

5.3.1 Modèle interprétable intrinsèque

L'interprétabilité intrinsèque peut être obtenue en concevant des modèles auto-explicatifs qui incorporent directement l'interprétabilité dans les structures du modèle. Ces modèles interprétables construits sont interprétables globalement ou pourraient fournir des explications lorsqu'ils font des prédictions individuelles (Mengnan et al., 2018 [13]).

Modèle d'interprétation globale : Les modèles d'interprétation globale peuvent être construits de deux façons : directement formés à partir de données mais avec des contraintes d'interprétabilité, et extraits d'un modèle complexe et opaque (Mengnan et al., 2018 [13]).

Contraintes d'interprétabilité : L'interprétabilité d'un modèle pourrait être favorisée par l'intégration de contraintes d'interprétabilité. Parmi les exemples représentatifs, mentionnons l'application de termes clairsemés ou l'imposition de contraintes de monotonie sémantique dans les modèles de classification. Dans ce cas, la rareté signifie qu'un modèle est encouragé à utiliser relativement moins de caractéristiques pour la prédiction, tandis que la monotonie permet aux caractéristiques d'avoir des relations monotones avec la prédiction. Ces contraintes simplifient le modèle et pourraient en améliorer l'intelligibilité pour les utilisateurs (A. Freitas, 2014 21).

Modèle d'interprétation locale : Les modèles d'interprétation locale sont généralement obtenus en concevant des architectures de modèles plus justifiées qui pourraient expliquer pourquoi une décision spécifique est prise. Contrairement aux modèles d'interprétation globale qui offrent un certain degré de transparence sur ce qui se passe à l'intérieur d'un modèle, les modèles d'interprétation locale fournissent aux utilisateurs une justification compréhensible pour une prévision spécifique (Mengnan et al., 2018 [13]).

5.3.2 Explication globale post-hoc

Les modèles d'apprentissage automatique apprennent automatiquement des modèles utiles à partir d'une énorme quantité de données d'entraînement et conservent les connaissances acquises dans les structures et les paramètres du modèle. L'explication globale post-hoc vise à fournir une compréhension globale des connaissances acquises par ces modèles pré-entraînés et à éclairer les paramètres ou les représentations apprises d'une manière intuitive pour les humains (Mengnan et al., 2018 [13]).

Explication agnostique au modèle : L'importance des caractéristiques agnostiques du modèle est largement applicable à divers modèles d'apprentissage automatique. Il traite un modèle comme une boîte noire et n'inspecte pas les paramètres internes du modèle (Mengnan et al., 2018 [13]).

Explication spécifique au modèle : Il existe également des méthodes d'explication spécialement conçues pour différents modèles. Les méthodes d'explication spécifique du modèle en tirent habituellement des explications en examinant les structures et les

paramètres du modèle interne (Mengnan et al., 2018 [13]).

5.3.3 Explication locale post-hoc

Les explications locales visent à identifier les contributions de chaque caractéristique de l'entrée à une prévision de modèle spécifique. Comme les méthodes locales attribuent généralement la décision d'un modèle à ses caractéristiques d'entrée, on les appelle aussi méthodes d'attribution (Mengnan et al., 2018 [13]).

Explication agnostique du modèle : Les méthodes agnostiques permettent d'expliquer les prédictions de modèles arbitraires d'apprentissage automatique indépendamment de leur implémentation. Elles comportent en même temps certains risques, car nous ne pouvons garantir que l'explication reflète fidèlement le processus décisionnel d'un modèle (Mengnan et al., 2018 [13]).

Explication spécifique du modèle : Il existe également des approches explicatives exclusivement conçues pour un type de modèle spécifique. Les méthodes spécifiques aux réseaux de neurones profonds, qui traitent les réseaux comme des boîtes blanches et utilisent explicitement la structure intérieure pour en tirer des explications, sont divisées en trois grandes catégories : les méthodes basées sur la rétropropagation de haut en bas (K. Simonyan et al., 2014 [31]), les méthodes basées sur la perturbation de bas en haut (K. Simonyan et al., 2014 [31]), l'étude des représentations profondes dans les couches intermédiaires (Mengnan et al., 2018 [13]).

5.4 Travaux existants

Les méthodes d'explication des systèmes à boîte noire peuvent être caractérisées comme spécifique au modèle (ou model-specific en anglais) *vs* modèle-agnostique (ou model-agnostic en anglais), et locale *vs* globale. Les méthodes spécifiques au modèle peuvent être utilisées seulement pour expliquer des réseaux de neurones profonds. Les méthodes des modèles agnostiques peuvent expliquer le résultats de n'importe quel système à boite noire (Riccardo et al., 2019 [23]).

5.4.1 Méthode ABELE

(Riccardo Guidotti et al., 2019 [23]) ont proposés la méthode ABELE (Adversarial Black box Explainer generating Latent Exemplars, en anglais), une approche, pour expliquer la décision d'un système à boite noire pour la classification d'images. Ils ont mis en place une approche pour expliquer les décisions des modèles à boîtes noires pour la classification des images. Tout en utilisant la boîte noire pour étiqueter les images, cette méthode d'explication exploite l'espace des caractéristiques latentes appris par un auto-

encodeur contradictoire (Advesial Autoencoder en Anglais, AAE)[Figure 5.3].

La méthode proposée génère d'abord des images modèles dans l'espace des traits latents et apprend un arbre de décision classificateur. Ensuite, elle sélectionne et décode des exemplaires respectant les règles de décision locales. Enfin, elle les visualise de manière à montrer à l'utilisateur comment les exemples peuvent être modifiés pour rester dans leur classe, ou pour devenir des contrefactuels en se "transformant" en une autre classe. L'explication obtenue à partir des exemples fournit également une carte des points saillants mettant en évidence les zones de l'image qui contribuent à sa classification, et les zones de l'image qui la poussent dans une autre classe.

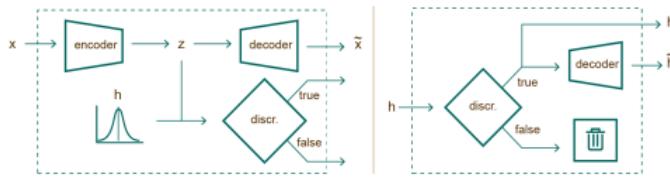


FIGURE 5.3 – A gauche : Architecture d'auto-encodeur contradictoire : l'encodeur (encoder) transforme l'image x en sa représentation latente z , le décodeur (decoder) reconstruit une approximation \tilde{x} de x à partir de z , et le discriminateur identifie si une instance latente h générée aléatoirement peut être considérée comme valide ou non. A droite : Module discriminateur et décodeur (discr.) : l'entrée est une instance latente h générée aléatoirement et, si elle est considérée comme valide par le discriminateur, celui-ci la renvoie avec sa version décompressée \tilde{h} .

ABELE est un explicateur local agnostique au modèle pour les classificateurs d'images résolvant le problème d'explication des résultats. Étant-donné une image x à expliquer et une boîte noire b , l'explication fournie par ABELE est composée par (i) un ensemble d'exemplaires et contre-exemplaires d'images qui illustrent, respectivement, les instances classifiées avec la même étiquette et avec une étiquette différente de celle de l'instance à expliquer x , qui peuvent être analysées visuellement pour comprendre les raisons de la classification, et (ii) une carte des points saillants (a saliency map, en anglais) mettant en évidence les zones de l'image x à expliquer qui contribuent à sa classification, et les zones de l'image qui la poussent vers une autre classe (Riccardo et al., 2019 [23]).

Le processus d'explication comprend les étapes suivantes. Tout d'abord, ABELE génère un voisinage dans l'espace de la caractéristique latente en exploitant l'Auto-encodeur, puis il apprend un arbre de décision sur ce voisinage latent fournissant une décision locale et des règles contre-factuelles. Enfin, ABELE sélectionne et décode les exemples et contre-exemples satisfaisant ces règles et en extrait une carte des points saillants [Figure 5.4].

En conclusion, (Riccardo et al., 2019 [23]) ont présenté les résultats d'une évaluation expérimentale sur trois jeux de données et deux modèles de boîte noire, en plus de fournir

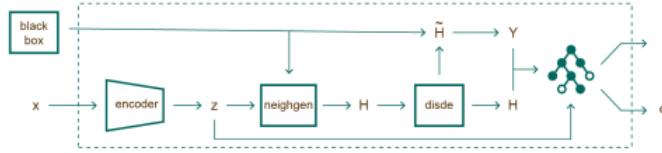


FIGURE 5.4 – Extracteur de règles locales latentes. Il prend en entrée l'image x à expliquer et la boîte noire b (black box). Avec l'encodeur (encoder) formé par l'AAE, il transforme x en sa représentation latente z . Ensuite, le module (neighgen) utilise z et b pour générer le voisinage local latent H . Les instances valides sont décodées dans \tilde{H} par le module (disde). Les images dans \tilde{H} sont étiquetées avec la boîte noire $Y = b(\tilde{H})$. H et Y sont utilisés pour apprendre un classificateur d'arbre de décision. Enfin, une règle de décision r et les règles contre-factuelles Θ pour z sont retournées.

les explications les plus utiles et les plus faciles à interpréter, dans leurs article [23].

5.4.2 Méthode LIME

(Marco Tulio Ribeiro et al., [24]) ont proposés la méthode LIME (Local Interpretable Model-Agnostic Explanations, en Anglais),une technique d'explication qui explique les prédictions de tout classificateur d'une manière interprétable et fidèle, en apprenant un modèle interprétable localement autour de la prédiction.

Étant donné que le modèle est agnostique, certaines données en entrées sont perturbées autour de leurs voisnages pour apprendre le comportement du modèle sous-jacent afin de voir comment les prédictions changent (Marco Tulio Ribeiro, 2016 [25]). Une illustration de ce processus est donnée dans la [Figure 5.5].

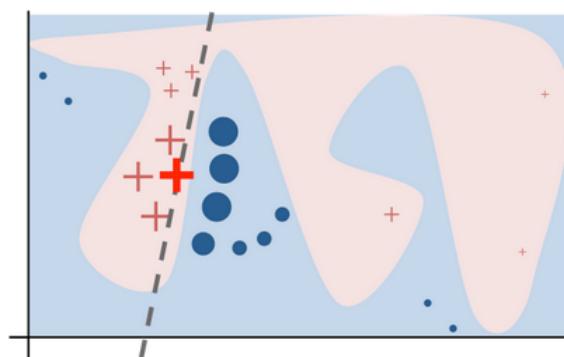


FIGURE 5.5 – Processus de LIME.

Comme illustré dans la [Figure 5.5], La fonction de décision du modèle original est représentée par le fond bleu/rose, et est clairement non linéaire. La croix rouge vif est l'instance expliquée (qu'elle soit x). Les instances perturbées autour de x sont échantillonées, et pondérées en fonction de leur proximité avec x (le poids est ici représenté par la taille). Le modèle original effectue une prédiction sur ces instance perturbées, puis

un modèle linéaire (ligne pointillée) qui se rapproche bien du modèle à proximité de x effectue l'apprentissage. L'explication dans ce cas n'est pas fidèle globalement, mais elle est fidèle localement autour de x (Marco Tulio Ribeiro et al., 2016 [24]).

Formellement, une explication est définie comme un modèle $g \in G$, où G est une classe de modèles potentiellement interprétables, c'est-à-dire qu'un modèle $g \in G$ peut être facilement présenté à l'utilisateur avec des artefacts visuels ou textuels. Le domaine de g est $\{0, 1\}^{d'}$, c'est-à-dire g agit sur l'absence/présence des composantes interprétables. Comme tous les $g \in G$ ne sont pas assez simples pour être interprétables, $\Omega(g)$ est donc une mesure de la complexité (par opposition à l'interprétabilité) de l'explication $g \in G$. Que le modèle expliqué soit dénoté $f : \mathbb{R}^d \rightarrow \mathbb{R}$. Dans la classification, $f(x)$ est la probabilité (ou l'indicateur binaire) que x appartient à une certaine classe. En outre, $\pi_x(z)$ est utilisé comme mesure de proximité entre une instance z et x , de manière à définir la localité autour de x . Finalement, que $\mathcal{L}(f, g, \pi_x)$ soit une mesure de l'infidélité de g dans l'approximation de f dans la localité définie par π_x . Afin d'assurer à la fois l'**interprétabilité et la fidélité locale**, $\mathcal{L}(f, g, \pi_x)$ est minimisée tout en ayant $\Omega(g)$ suffisamment bas pour être interprétable par l'utilisateur (Marco Tulio Ribeiro et al., [24]). L'explication produite par LIME est obtenue par ce qui suit :

$$\xi(x) = \underset{g \in G}{\operatorname{argmin}} \mathcal{L}(f, g, \pi_x) + \Omega(g) \quad (5.2)$$

Le modèle ξ pour l'instance x est le modèle g qui minimise la perte \mathcal{L} , qui mesure à quel point l'explication est proche de la prédiction du modèle original f , tandis que la complexité du modèle $\Omega(g)$ est maintenue à un niveau bas. La mesure de proximité π_x définit la taille du voisinage autour de l'instance x que nous considérons pour l'explication (Christoph Molnar, 2019 [27]).

Les modèles de substitution locaux (Local Surrogate, en Anglais) sont des modèles interprétables qui sont utilisés pour expliquer les prédictions individuelles des modèles d'apprentissage des systèmes à boîte noire. Les modèles de substitution sont entraînés à approximer les prédictions du modèle sous-jacent de la boîte noire. Au lieu de former un modèle de substitution global, le LIME se concentre sur la formation de modèles de substitution locaux pour expliquer les prédictions individuelles (Christoph Molnar, 2019 [27]). Nous résumons les étapes pour former des modèles de substitution locaux en ce qui suit :

- Sélectionner l'instance d'intérêt pour laquelle une explication de la prédiction en boîte noire est souhaitée.
- Perturber un jeu de données pour obtenir les prédictions de la boîte noire pour ces nouveaux points.

- Pondérer les nouveaux échantillons en fonction de leur proximité avec l’instance d’intérêt.
- Entraîner un modèle pondéré et interprétable sur l’ensemble de données avec les variations.
- Expliquer la prédiction en interprétant le modèle local.

Voir la [Figure 5.6] pour un exemple de la façon dont LIME fonctionne pour la classification des images. prenons exemple d’expliquer un classificateur qui prédit la probabilité que l’image contienne une rainette. L’image de gauche est divisées en composantes interprétables (superpixels contigus) (Marco Tulio Ribeiro et al., 2016 [26]).

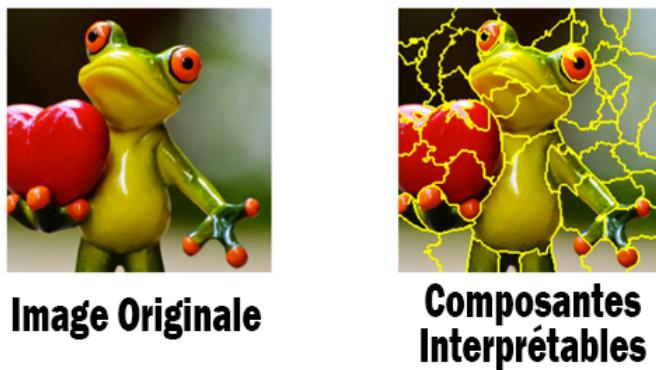


FIGURE 5.6 – Transformation d’une image en composantes interprétables.

Comme l’illustre la [Figure 5.7], un ensemble de données d’instances perturbées est généré en désactivant certaines des composantes interprétables (dans ce cas, en les rendant grises). Pour chaque instance perturbée, la probabilité qu’une rainette soit dans l’image est obtenue selon le modèle. Ensuite un modèle simple (linéaire) apprend sur cet ensemble de données, qui est pondéré localement, c’est-à-dire que s’est soucié davantage de faire des erreurs dans les instances perturbées qui sont plus semblables à l’image originale. A la fin, les superpixels ayant les poids positifs les plus élevés comme explication sont présentées en grisant tout le reste (Marco Tulio Ribeiro et al., 2016 [26]).

En conclusion, *Marco Tulio Ribeiro, Sameer Singh et Carlos Guestrin* dans leurs article [24], ont détaillé LIME, une technique qui explique les prédictions de tout classificateur de manière interprétable et fidèle, en apprenant un modèle interprétable localement autour de la prédiction. Ils ont également proposé une méthode pour expliquer les modèles en présentant des prédictions individuelles représentatives et leurs explications de manière non redondante, en encadrant la tâche comme un problème d’optimisation sous-modulaire. Ils ont démontré la flexibilité de ces méthodes en expliquant différents modèles pour la classification de textes (par exemple, les forêts aléatoires) et d’images (par exemple, les réseaux de neurones). Ils ont montré l’utilité des explications par de nouvelles expériences, à la fois simulées et avec des sujets humains, sur divers scénarios qui nécessitent de la

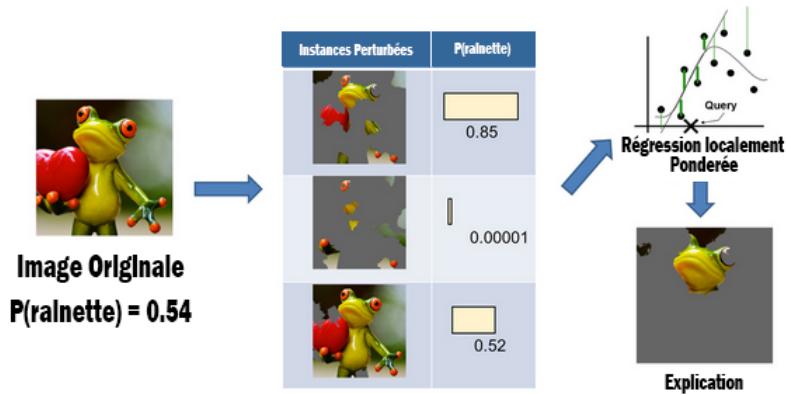


FIGURE 5.7 – Explication d'une prédition avec LIME.

confiance : décider si l'on doit faire confiance à une prédition, choisir entre les modèles, améliorer un classificateur non fiable, et déterminer pourquoi on ne devrait pas faire confiance à un classificateur.

5.4.3 Méthodes à base de IC & UC

L'utilisation des réseaux de neurones est encore difficile dans de nombreux domaines d'application en raison du manque de possibilités d'explication (problème de systèmes à boîtes noires). Les concepts d'importance contextuelle (Contextual Importance en Anglais, CI) et d'utilité contextuelle (Contextual Utility en Anglais, CU) permettent d'expliquer les résultats des réseaux de neurones de manière compréhensible pour l'utilisateur. Dans les explications obtenues le module de raisonnement et le module d'explication sont complètement séparés (Främling, 1996 [28]). Cette méthode permet d'expliquer les résultats des prédictions sans transformer le modèle en un modèle interprétable (Anjomshare et al., [29]). Plusieurs études ont suggéré de modéliser des installations d'explication basées sur des concepts théoriques pertinents sur le plan pratique, tels que les justifications contrastives (contrastive justifications, en Anglais), afin de produire des explications compréhensibles par l'homme, ainsi que des explications complètes (complete justifications, en Anglais) (Miller, 2018 [30]). Les explications complètes présentent la liste des causes d'une prédition individuelle, tandis que les explications contrastives justifient pourquoi une certaine prédition a été faite au lieu d'une autre (Christoph Molnar, 2019 [27]).

IC et UC sont des valeurs numériques qui peuvent être représentées sous forme visuelle et en langage naturel pour présenter des explications pour des cas individuels (Främling, 1996 [28]). (Anjomshae et al., [29]), ont visé à fournir des explications complètes ainsi que des explications contrastives en utilisant les méthodes IC et UC pour les systèmes à boîte noire. Cette approche peut généralement être utilisée avec des modèles d'apprentissage à la fois linéaires et non linéaires. Les méthodes basées sur les IC et UC offrent la possibilité

d'expliquer directement les résultats (et non la connaissance elle-même) des réseaux de neurones sans transformer la connaissance du réseau de neurones. Elles sont basées sur les concepts d'importance et d'utilité d'une entrée sur la variable de sortie, qui sont des indicateurs pouvant être calculés directement à partir du réseau de neurones.

Dans la pratique, l'importance des critères et l'utilité de leurs valeurs changent en fonction du contexte courant. Par temps froid, l'importance et l'utilité des vêtements chauds augmentent par rapport à la chaleur de l'été, alors que l'importance de l'indice de protection solaire qui pourrait être utilisé devient faible. C'est la raison pour laquelle le mot contextuel a été choisi pour décrire l'IC et l'UC. Cette approche peut généralement être utilisée avec des modèles d'apprentissage linéaires et non linéaires. Elle est basée sur l'explication des prédictions du modèle sur l'importance et l'utilité individuelles de chaque caractéristique (Anjomshae et al., [29]). IC et UC sont définis comme :

$$IC = \frac{Cmax_x(C_i) - Cmin_x(C_i)}{absmax - absmin} \quad (5.3)$$

$$UC = \frac{y_{i,j} - Cmin_x(C_i)}{Cmax_x(C_i) - Cmin_x(C_i)} \quad (5.4)$$

où

- C_i est le contexte étudié (qui définit les valeurs d'entrée fixes du modèle),
- x est la ou les entrées pour lesquelles le IC et le UC sont calculés, il peut donc également s'agir d'un vecteur,
- $y_{i,j}$ est la valeur de sortie de la sortie j étudiée lorsque les entrées sont celles définies par C_i ,
- $Cmax_x(C_i)$ et $Cmin_x(C_i)$ sont les valeurs de sortie les plus élevées et les plus basses observées en faisant varier la ou des entrées x ,
- $absmax$ et $absmin$ spécifient la plage de valeurs pour la sortie j étudiée.

IC correspond à la fraction de la plage de sortie couverte par la variation de la ou des valeurs des entrées x et de la plage de sortie maximale. UC reflète la position de $y_{i,j}$ dans la gamme de sortie couverte ($Cmax_x(C_i) - Cmin_x(C_i)$). Chaque caractéristique x avec prédiction $y_{i,j}$ a ses propres valeurs de IC et de UC.

(Främling, 1996 [28]) a mentionné que IC et UC sont des valeurs numériques continues, faciles à trouver pour une entrée en utilisant la simulation de Monte-Carlo ou d'autres méthodes. Elles peuvent ensuite être transformées en valeurs symboliques afin de produire des explications. La transformation peut être faite simplement en définissant des plages de valeurs pour " bon ", " mauvais ", " important ", " peu important ", ..., ou en utilisant des ensembles flous. Les notions de IC et de UC sont illustrées par l'exemple de la [Figure

5.8]. La fonction de préférence apprise par le réseau de neurones est celle du choix d'une voiture, où les entrées du réseau de neurones sont les caractéristiques de la voiture et la valeur de sortie est la valeur de préférence (ou " score ") obtenue. Le contexte est défini par la voiture étudiée, la " SAAB 900 S 2.0-16 ", qui donne les valeurs de toutes les entrées sauf celle étudiée. La valeur de préférence est indiquée en fonction du prix de la voiture, la croix indiquant la valeur actuelle. L'échelle indiquée sur la droite montre certaines limites pour traduire UC en mots. Le même type d'échelle est utilisé pour IC (Främling, 1996 [28]).

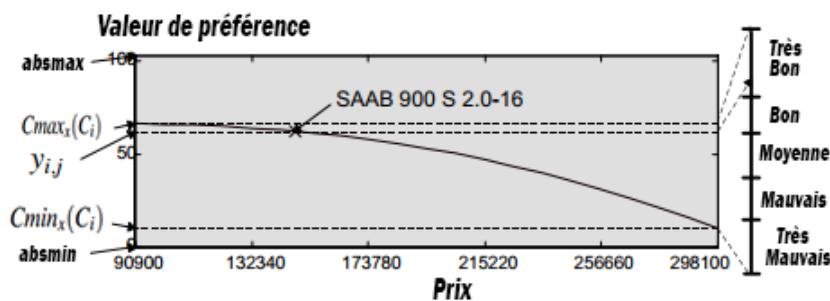


FIGURE 5.8 – Illustration des notions d'importance contextuelle et d'utilité contextuelle à partir de la problématique du choix d'une voiture .

L'IC et l'UC peuvent également être calculés pour plus d'une entrée ou même pour toutes les entrées, ce qui signifie que des concepts arbitraires de niveau supérieur qui sont des combinaisons de plus d'une entrée peuvent être utilisés dans les explications. Comme les concepts et les vocabulaires utilisés pour produire les explications sont externes à la boîte noire, les vocabulaires et les explications visuelles peuvent être adaptés en fonction de l'utilisateur auquel ils sont destinés. La [Figure 5.9] illustre comment les explications sont générées en utilisant la méthode de l'importance contextuelle et de l'utilité. Un autre point important est que les humains demandent généralement des explications sur la raison pour laquelle une certaine prédiction a été faite au lieu d'une autre. Cela donne une meilleure idée de ce qui se serait passé si l'entrée avait été différente. Créer des explications contrastives et comparer les différences à une autre instance peut souvent être plus utile que l'explication complète seule pour une prédiction particulière. Puisque l'importance contextuelle et les valeurs d'utilité peuvent être produites pour toutes les combinaisons possibles de valeurs d'entrée et de sortie, cela permet d'expliquer pourquoi une certaine instance C_i est préférable à une autre, ou pourquoi une classe (sortie) est plus probable qu'une autre (Anjomshae et al., [29]).

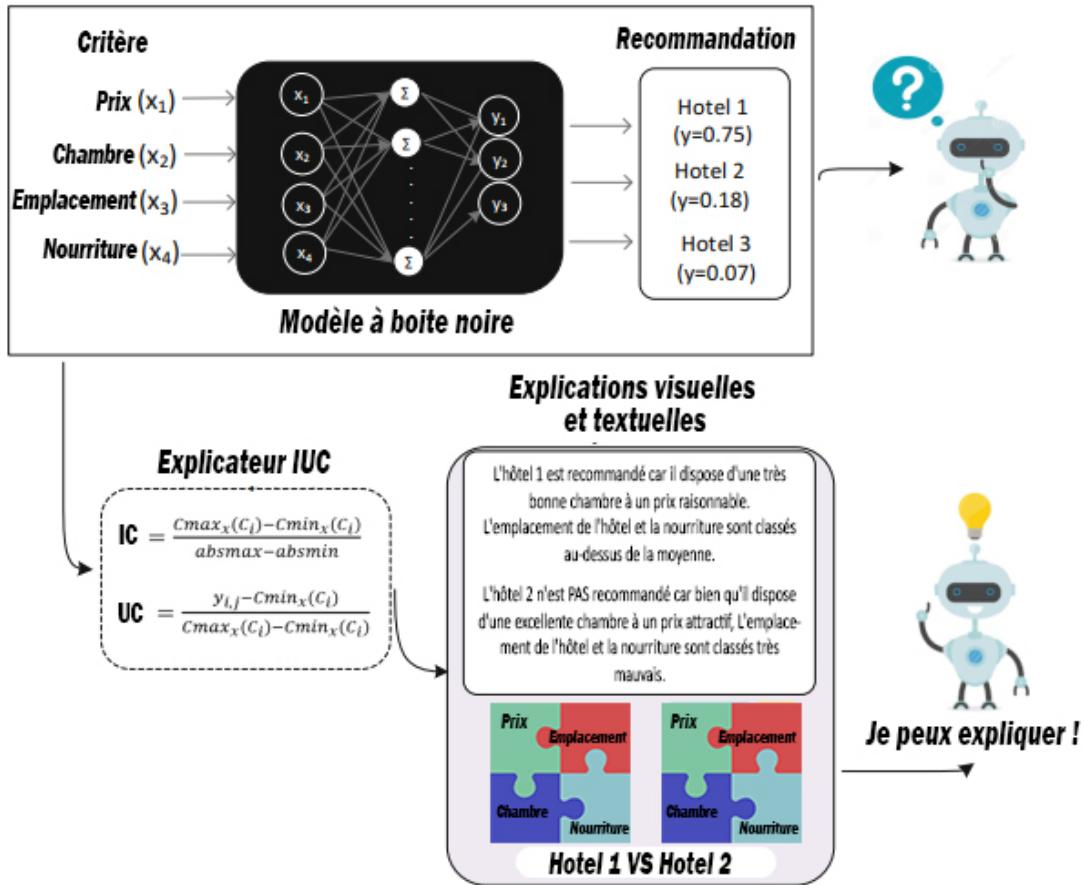


FIGURE 5.9 – Fournir des explications pour les cas individuels utilisant IC et UC.

En conclusion, (Anjomshae et al., [29]) ont présenté un exemple d'explication de modèles linéaires et non linéaires pour démontrer la généralisabilité de la méthode. Ils ont montré l'utilité des explications dans l'exemple de la sélection des voitures et de la classification des fleurs d'iris en présentant une explication complète et contrastive.

5.5 Méthode basée sur le Raisonnement à partir de Cas

Il existe actuellement plusieurs architectures d'apprentissage profond et, ces dernières années, les réseaux de neurones convolutifs sont l'architecture utilisée pour effectuer des opérations complexes à l'aide de filtres convolutifs (Zhang, Yu-Dong, et al, 2015 [79]). Pour la classification d'une image, les CNNs sont les architectures de réseau communément utilisées pour l'apprentissage profond (Srinivas et Sasibhushana 2019, [46]). La nature non linéaire de ce type de réseaux de neurones les rend difficiles à interpréter et difficiles à expliquer (Olden et Jackson, 2002 [47] ; Selvaraju et al., 2017 [48]). Les CNNs peuvent être encore plus opaques en ce qui concerne leurs caractéristiques et leurs contributions res-

pectives aux classifications. Les tentatives d'amélioration de l'interprétabilité des CNNs se sont largement concentrées sur la visualisation de ce que des neurones spécifiques du réseau ont appris. Toutefois, on peut soutenir que ces visualisations doivent encore être contextualisées (Kenny et Keane, 2019 [49]).

Le RàPC est une technique d'IA qui a été appliquée dans une grande variété de domaines pour de nombreuses tâches différentes, par exemple, diagnostic, conception, classification, recommandation et droit,(De Mantaras et al., 2005 [52]). Dans sa forme la plus simple, lorsqu'un cas d'interrogation est présenté, les cas les plus similaires à celui-ci sont récupérés avant d'être utilisés directement (ou adaptés) pour faire une prédiction. En général, l'étape de récupération permet de trouver des cas en faisant correspondre les caractéristiques de la requête à la base de cas en utilisant l'algorithme des k-plus proche voisins (k-NN). La précision de l'extraction peut dépendre fortement du poids accordé à ces caractéristiques, qui reflète leur importance dans le domaine (Kenny et Keane, 2019 [49]). Dans les algorithmes d'apprentissage automatiques, le plus simple est l'algorithme de classification k-NN (Suguna et al., 2010 [53]). Il s'agit d'une méthode de classification des images utilisant un espace de caractéristiques construit sur les données d'entraînement les plus proches. Dans le processus d'apprentissage, l'algorithme ne stocke que les étiquettes et les vecteurs de caractéristiques des paramètres nécessaires à l'apprentissage des images. Dans la tâche de classification, les k-plus proches voisins sont affectés au point de requête non étiqueté. Ensuite, la classification de l'image est basée sur les étiquettes de ses k-plus proches voisins. Si $k=1$, cela signifie que l'image est classée comme la classe la plus proche. Ensuite, chaque image est convertie en un vecteur de longueur fixe avec des nombres réels en utilisant la distance euclidienne comme fonction de distance commune pour k-NN (Srinivas et Sasibhushana 2019, [46]).

5.5.1 Les systèmes jumeaux RNA-RàPC

Les systèmes RNA-RàPC combinés, souvent nommés "RNA-RàPC jumeaux" , sont un cas particulier de système hybride, qui combine des modules RNA et RàPC, lorsque la précision et l'interprétabilité sont les exigences principales du système global. Bien que les modules RNA et RàPC aient été couplés comme les premiers systèmes hybrides, ce n'est qu'à la fin des années 1990 que de "vrais" jumeaux sont apparus (Keane et kenny, 2019 [54]).

Les systèmes hybrides combinent généralement plusieurs techniques d'IA pour répondre à une tâche donnée. Par exemple, (Reategui et al., 1997) ont combiné un système RNA et un système RàPC pour le diagnostic médical, où le RNA a généré des hypothèses pour orienter la recherche de cas dans un système RàPC. Cependant, ces systèmes ne sont

pas des systèmes jumeaux dans notre sens, car la motivation pour combiner les techniques est de construire un pipeline de traitement pour réaliser une certaine tâche, où chaque technique est généralement insuffisante en soi. En revanche, l'idée des systèmes jumeaux utilise deux modèles l'un à côté de l'autre où le but de l'utilisation des deux techniques est, spécifiquement, l'explication (Kenny et Keane, 2019 [49]).

Définition du couple RNA-RàPC : En conséquence, nous pouvons définir un couple RNA-RàPC (Keane et kenny, 2019 [54]), précisément, comme un système avec :

- **Deux techniques :** Un système hybride dans lequel (au moins) deux techniques, RNA (MLP ou DNN) et techniques RàPC (notamment, k-NN), sont combinées pour répondre aux exigences de précision et d'interprétabilité du système.
- **Modules séparés :** où ces techniques sont exécutées comme des modules séparés, indépendamment mais, pour ainsi dire, côte à côté.
- **Ensemble de données commun :** les deux techniques sont exécutées sur le même ensemble de données (c'est-à-dire qu'elles sont jumelées par cet usage commun).
- **Cartographie des poids des caractéristiques (ou Feature-Weight Mapping en Anglais) :** une description des fonctionnalités du RNA, généralement décrite comme les poids des caractéristiques, qui "reflètent" ce que le RNA a appris, est mise en correspondance avec l'étape de récupération des k-NN du système RàPC.
- **Division bipartite du travail :** dans les modules RNA et RàPC, le premier fournit des prédictions et le second assure l'interprétabilité, en expliquant les résultats du RNA (pour la classification ou la régression).

Comme (Keane et kenny, 2019 [54]) ont cité, le succès de toute cette entreprise dépend d'un certain nombre de facteurs :

- (a) le RNA doit être raisonnablement précise dans ses prévisions,
- (b) les pondérations des caractéristiques extraites du RNA ont une grande fidélité à sa fonction,
- (c) les voisins les plus proches trouvés n'ont pas un rapport trop complexe avec la requête,
- (d) et l'utilisateur dispose d'une expertise suffisante pour établir facilement un lien avec ces voisins proches explicatifs au cas de la requête.

Dans le travail de (Kenny et Keane, 2019 [49]) plusieurs méthodes de pondération des caractéristiques sont testées de manière concurrentielle dans le cadre de deux expériences, dont leurs méthode basée sur les contributions (appelée COLE) qui s'avère la plus performante. Les essais portent sur le "jumelage" de réseaux perceptron multicouches (MLP) traditionnels et de réseaux neuronaux convolutionnels (CNN) avec des systèmes CBR. Pour les CNN formés sur des données d'images, les preuves qualitatives montrent que les cas fournissent des explications plausibles pour les classifications des CNN.

5.5.2 Autres travaux

Il existe plusieurs autres méthodes et travaux pour expliquer les résultats obtenus par les systèmes à boites noires. Au-delà des méthodes basées sur les gradients¹ et les perturbations mentionnées ici [31, 32, 33, 34, 35], il existe diverses autres méthodes similaires [36]. Ces méthodes ont en commun de ne pas modifier les architectures existantes, mais de s'appuyer sur des calculs a posteriori pour inverser les valeurs d'importance ou les sensibilités des entrées. (Li et al., 2018 [37]) proposent une architecture de réseau de neurones interprétable dont les prédictions sont basées sur la similarité de l'entrée à un petit ensemble de prototypes, qui sont appris pendant la formation. (Kim et al, 2018 [38]) proposent une technique d'apprentissage de vecteurs d'activation de concepts représentant des concepts d'intérêt conviviaux pour l'homme, en s'appuyant sur un ensemble d'exemples annotés caractérisant ces derniers. En calculant des dérivées directionnelles le long de ces vecteurs, ils évaluent la sensibilité des prédicteurs par rapport aux changements sémantiques dans la direction du concept. (David Alvarez-Melis et Tommi Jaakkola, 2018 [42]) conçoivent des modèles auto-explicatifs par étapes, en généralisant progressivement les classificateurs linéaires à des modèles complexes mais explicites sur le plan architectural. (Lei et al., 2017 [39]) proposent une architecture de réseau de neurones pour la classification des textes qui " justifie " ses prédictions en sélectionnant les jetons pertinents dans le texte de saisie. Mais ce qui est interprétable est ensuite exploité par un réseau de neurones complexe, de sorte que la méthode est transparente comme à quel aspect de les données d'entrées sont utilisées pour la prédiction. (Osbert Bastani et al., 2019 [40]) proposent de construire des explications globales de modèles complexes de boîte noire sous la forme d'un arbre de décision approchant le modèle original, tant que l'arbre de décision est une bonne approximation, il reflète alors le calcul effectué par le modèle de boîte noire. (Rajiv Khanna et al., 2018 [41]) ont conçu une méthode d'interprétation en utilisant les noyaux de Fisher (Fisher Kernels, en Anglais), qui indique quels exemples d'entraînement sont les plus responsables pour un ensemble donné de prédictions. (Paolo Tamagnini et al., 2017 [43]) ont proposé Rivelio, une interface d'analyse visuelle qui permet aux analystes de comprendre les causes derrière les prédictions des classificateurs binaires en explorant de manière interactive un ensemble d'explications au niveau des instances. Ces explications sont agnostiques au modèle, traitant un modèle comme une boîte noire, et elles aident les analystes à sonder interactivement l'espace de données binaires en haute dimension pour détecter les caractéristiques pertinentes aux prédictions.

1. L'algorithme du gradient désigne un algorithme d'optimisation différentiable destiné à minimiser une fonction réelle différentiable définie sur un espace euclidien.

5.6 Synthèse des travaux

Dans cette section nous allons faire une synthèses sur des méthodes d’explications des résultats obtenus par les systèmes à boite noire.

La classification des méthodes proposée par (Guidotti et al., 2018[18]) est basée sur le type de problème rencontré et sur l’explicatif adopté pour ouvrir la boîte noire. En particulier, nous prenons en compte les caractéristiques suivantes :

- Le type de problème rencontré,
- le type d’explication adopté pour ouvrir la boîte noire,
- le type de modèle de boîte noire que la méthode d’explication est en mesure d’ouvrir,
- le type de données utilisées en entrée par le modèle de boîte noire.

Nous avons vu dans ce chapitres les différentes façons d’expliquer les résultats des boites noires. Il existe des méthodes qui ouvre ces systèmes qui fournissent des modèles globalement interprétables qui sont capables d’imiter le comportement des boîtes noires et qui sont également compréhensibles par l’utilisateur. Des méthodes permettant de résoudre le problème de l’explication des résultats de boites noires qui fournissent un modèle interprétable localement, capable d’expliquer la prédition de la boîte noire en termes compréhensibles pour l’utilisateur pour une instance ou un enregistrement spécifique. Des méthodes d’inspection de la boîte noire qui fournissent une représentation permettant de donner une explication locale ou globale du modèle. Les méthodes conçues pour résoudre le problème de la classification en utilisant une méthode transparente qui est localement ou globalement interprétable par elle-même, c’est-à-dire en résolvant le problème de la conception de la boîte transparente.

Nous allons catégoriser les différentes méthodes utilisé pour effectuer l’explication des systèmes à boites noires (Guidotti et al., 2018[18]), comme suit :

- **Decision Tree (DT) ou Single Tree** : l’un des modèles les plus interprétables et les plus facilement compréhensibles, principalement pour les explications globales, mais aussi locales. En effet, une technique très répandue pour ouvrir la boîte noire est ce que l’on appelle "l’approximation à arbre unique".
- **Decion Rules (DR) ou Rule Based Explanator** : Les règles de décision font partie des techniques les plus compréhensibles pour l’utilisateur. Elles sont utilisées pour expliquer le modèle, le résultat et aussi pour la conception transparente.
- **Features Importance (FI)** : Une solution très simple mais effective agissant comme une explication globale ou locale consiste à renvoyer comme explication le poids et l’ampleur des caractéristiques utilisées par la boîte noire.

- **Saliency Mask (SM)** : Une manière efficace de mettre en évidence ce qui provoque un certain résultat, en particulier lorsque des images ou des textes sont traités, consiste à utiliser des "masques" mettant en évidence visuellement les aspects déterminants du document analysé.
- **Sensitivity Analysis (SA)** : Elle consiste à évaluer l'incertitude du résultat d'une boîte noire par rapport à différentes sources d'incertitude dans ses entrées. Elle est généralement utilisée pour développer des outils visuels pour l'inspection des modèles.
- **Partial Dependence Plot (PDP)** : Ces tracés aident à visualiser et à comprendre la relation entre le résultat d'une boîte noire et l'entrée dans un espace de caractéristiques réduit.
- **Prototype Selection (PS)** : Cet explicatif consiste à renvoyer, avec le résultat, un exemple très similaire à l'enregistrement classé, afin d'indiquer clairement quels critères la prédiction a été renvoyée.
- **Activation Maximization (AM)** : L'inspection des réseaux neuronaux et du réseau neuronal profond peut également être effectuée en observant quels sont les neurones fondamentaux activés par rapport à des enregistrements d'entrée particuliers, c'est-à-dire en recherchant des modèles d'entrée qui maximisent l'activation d'un certain neurone dans une certaine couche. Elle peut également être considérée comme la génération d'une image d'entrée qui maximise l'activation de sortie.

Plusieurs de types de boîtes noires ont été expliquées :

- **Réseaux de neurones (Neural Networks, NN)** : Inspirés des réseaux de neurones biologiques, les réseaux de neurones artificiels sont formés par un ensemble de neurones connectés. Chaque lien entre les neurones peut transmettre un signal. Le neurone récepteur peut traiter le signal puis le transmettre aux neurones en aval qui lui sont connectés. En règle générale, les neurones sont organisés en couches. Les couches différentes effectuent des transformations différentes sur leurs entrées. Les signaux voyagent de la couche d'entrée à la couche de sortie, en passant plusieurs fois par la ou les couches cachées du milieu. Les neurones et les connexions peuvent également avoir un poids qui varie au fur et à mesure de l'apprentissage, ce qui peut augmenter ou diminuer la force du signal.
- **Ensemble d'arbres (Tree Ensemble, TE)** : Les méthodes d'ensemble combinent plus d'un algorithme d'apprentissage pour améliorer la puissance prédictive de n'importe lequel des algorithmes d'apprentissage individuels qu'elles combinent. Les forêts aléatoires, les arbres renforcés et l'ensachage d'arbres sont des exemples d'ensembles d'arbres. Ils combinent les prédictions des arbres de décision différents, chacun étant formé sur un sous-ensemble indépendant (en ce qui concerne les caractéristiques et les enregistrements) des données d'entrée.

- **Machines à vecteurs de support (Support Vector Machine, SVM)** : Les machines à vecteurs de support utilisent un sous-ensemble de données de formation, appelé vecteurs de support, pour représenter la limite de décision. Un SVM est un classificateur qui, en utilisant un ensemble de noyaux différents disponibles, recherche les hyperplans ayant la plus grande marge pour la frontière de décision.
- **Réseau neuronal profond (Deep Neural Networks, DNN)** : Un DNN est un NN qui peut modéliser une relation non linéaire complexe avec plusieurs couches cachées. Une architecture DNN est formée par une composition de modèles exprimés sous la forme d'une combinaison en couches d'unités de base. Les réseaux neuronaux récurrents (Recurrent Neural Networks, RNN) sont un type de DNN largement utilisé. Une composante particulière des RNN sont les noeuds à mémoire longue et courte durée (Long Short-Term Memory, LSTM), qui sont notamment effectives pour la modélisation du langage. Cependant, dans le traitement d'images, les réseaux neuronaux conventionnels (CNN) sont généralement utilisés. Nous faisons une distinction entre NN et DNN uniquement sur le fait que le DNN peut être plus profond que le NN et qu'il peut utiliser une architecture de noeuds plus sophistiquée (par exemple, RNN, CNN).
- **Modèles non linéaires (Non-Linear Models, NLM)** : La fonction utilisée pour modéliser les observations est une combinaison non linéaire des paramètres du modèle et dépend d'une ou de plusieurs variables indépendantes du mode.

Les types de données utilisées comme entrée des boîtes noires analysées sont les suivants :

- **Tabulaire (TAB)** : Avec les données tabulaires, nous indiquons tout ensemble de données classique dans lequel chaque enregistrement partage le même ensemble de caractéristiques et chaque caractéristique est soit numérique, soit catégorique, soit booléenne.
- **Image (IMG)** : De nombreuses boîtes noires fonctionnent avec des images étiquetées. Ces images peuvent être traitées telles quelles par la boîte noire ou peuvent être pré-traitées (par exemple, redimensionnées pour avoir toutes les mêmes dimensions).
- **Texte (TXT)** : Comme la modélisation du langage est l'une des tâches les plus largement évaluées de nos jours avec la reconnaissance d'images, les ensembles de données étiquetés de texte sont généralement utilisés pour des tâches telles que la détection du spam ou la classification des sujets.

Nous allons lister dans la [Table 5.1] quelques solutions utilisées pour expliquer les résultats de la boite noire.

Nous avons résumé quelques méthodes d'explication des boites noires cité dans (Guidotti et al., 2018[18]). Le contenu des champs du tableau sont signifiants comme suit :

Nom	Auteur	Année	Problème	Boite noire	Explication	Type	Données	Code Source	Dataset
Trepan	Craven et al.	1996	EBN	NN	DT	TAB		X	
LIME	Ribeiro et al.	2016	ERBN	AGN	FI	ANY	X	X	
GDP	Baehrens	2010	IBN	AGN	SA	TAB	X	X	
BCM	Kim et al.	2014	CBT	-	PS	ANY		X	
STA	Zhou et al.	2016	EBN	TE	DT	TAB			
LORE	Guidotti et al.	2018	ERBN	AGN	DR	TAB	X	X	
DGM-AM	Nguyen et al.	2016	IBN	DNN	AM	IMG	X	X	
-	Lei et al.	2016	ERBN	DNN	SM	TXT		X	
FRL	Wang et al.	2015	CBT	-	DR	TAB	X	X	

TABLE 5.1 – Résumé des méthodes d’explication des boites noires.

- **Nom** : Il s’agit du nom de la méthode donnée par le ou les auteur(s).
- **Auteur** : S’agit de ou des auteurs(s) qui ont proposé la solution.
- **Année** : S’agit de l’année de la publication de la solution et l’article associé.
- **Problème** : Consiste du type de problème de l’explication qui peut être :
 - **EBN** pour le problème d’explication de la boite noire,
 - **ERBN** pour le problème d’explication des résultats de la boite noire,
 - **IBN** pour le problème d’inspection de la boite noire,
 - **CBT** pour le problème de la conception de la boite transparente.
- **Boite noire** : S’agit de la catégorie du modèle de boite noire qui s’adapte avec la méthode, détaillée précédemment, AGN pour Agnostique signifie que la méthode s’applique sur toute catégorie de boite noire.
- **Explication** : S’agit de la méthode utilisée pour élaborer l’explication, détaillé précédemment.
- **Type de données** : S’agit du type de données d’entrée utilisées par la boite noire.
- **Code source** : Indique si le code source est disponible au grand public ou non.
- **Dataset** : Indique si les ensembles de données utilisés dans les expériences sont disponibles.

Ces méthodes diffèrent l’une de l’autre selon le problème traité, le type de la boite noire traitée, les données présentées en entrées, ainsi le modèle utilisé pour l’explication. Il existe des méthodes qui sont adaptées à un seul type de boite noire (Spécifiques) et d’autres qui s’adapte sur n’importe quel type de boite noire (Agnostiques), qui peuvent offrir des explications sur le fonctionnement interne de la boite noire (Globale) ou sur les prédictions elles mêmes (Locale), et aussi des méthodes qui nécessitent un deuxième modèle pour expliquer le modèle à boite noire (Post-hoc) ou qui proposent un modèle qui s’explique lui même, c-a-d donne la prédition accompagnée de l’explication.

5.7 Conclusion

Dans ce chapitre nous avons détaillé les concepts reliés à la notion de boite noire et l’explication de leurs résultats, nous avons traité quelques méthodes proposées pour traité ce problème, et nous avons fait une comparaison entre ces méthodes. Dans le chapitre suivant nous allons proposer notre solution et détailler les concepts qui la composent.

Conception

Nous détaillons dans ce chapitre notre solution "Explication locale post-hoc spécifique au modèle", qui consiste en l'explication des résultats obtenus par un réseau de neurones convolutif en utilisant le raisonnement à partir de cas.

Explication locale : obtenue en identifiant les contributions des caractéristiques des entrées à la prévision faite par le RNA.

Explication post-hoc : pour assurer l'interprétabilité nous avons besoin de créer un deuxième modèle qui explique les résultats du RNA, le RàPC dans notre cas.

Explication spécifique au modèle : l'interprétabilité obtenue est spécifique à un type de RNA, CNN pour reconnaissance d'objets d'ImageNet.

6.1 Architecture de la solution

Nous allons présenter une conception de haut niveau où nous identifierons les éléments composant notre solution. Par la suite, nous détaillerons chaque élément en présentant l'architecture détaillée et les algorithmes mis en jeu.

La [Figure 6.1] montre le schéma de l'architecture de haut niveau de la solution proposée pour l'explication des résultats des RNA en utilisant le RàPC.

Notre système est composé principalement de quatre modules dont les fonctionnalités sont les suivantes :

- Acquisition des données en entrée (Requête).
- Extraction de caractéristiques (Boîte noire).
- Génération d'un modèle de classification (Prédiction).
- Explication des résultats obtenus de la classification (Explication).

Dans ce qui suit nous présentons les modules du système. Nous donnons leurs principales composantes et nous détaillons leur fonctionnement.

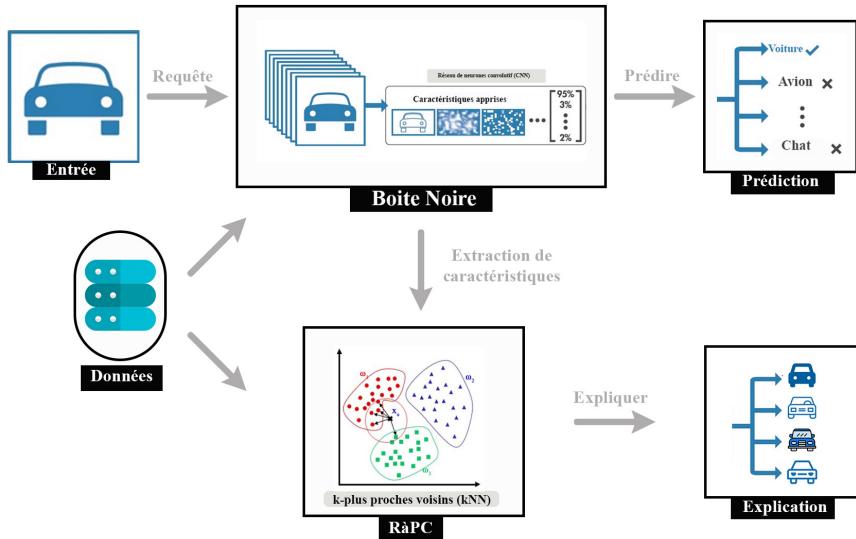


FIGURE 6.1 – Architecture du système jumelé RNA-RàPC : Une requête à un RNA produit une prédiction, mais inexpliquée, de la classe d'une image. Le RNA est jumelé avec un système RàPC, ce qui permet à ce dernier de retrouver le cas du voisin le plus proche, en utilisant les poids des caractéristiques du RNA, pour expliquer la prédiction.

6.2 Acquisition des données en entrée

La tâche consiste en la prédiction de la classe d'une image en entrée (Requête) [Figure 6.2], lorsque l'on dispose d'un ensemble de données d'exemples d'apprentissage (c'est-à-dire une base de cas antérieurs) décrivant les images et les classes auxquelles elles appartiennent. Le RNA apprend avec précision à prévoir la classe des images non vues (c'est-à-dire des cas de requêtes) après avoir calculé les entrées-sorties, la cartographie des caractéristiques de l'image et de sa classe à l'aide des données d'entraînement (training-set ou data-set en Anglais). Pour expliquer les prédictions du RNA ses poids caractéristiques (ou feature-weights en Anglais) sont extraits et utilisés dans l'étape de "Remémoration" du RàPC, pour identifier le ou les cas les plus similaires afin d'*"expliquer"* la prédiction du RNA. Dans la phase de remémoration qui consiste en la recherche des cas similaires nous allons utiliser l'algorithme KNN. En substance, l'étape de l'explication consiste à affirmer : l'image x est de la classe de voiture, car les autres images, qui ont des caractéristiques très similaires, appartiennent à cette même classe (qui sont proches de celle qui a été prédictée).

Ce module se charge de la réception des données d'entrée de la requête. Cette entrée est une image de ($largeur = 32px$, $hauteur = 32px$, $profondeur = 3$). La profondeur, représente les canaux R, V, B. La représentation matricielle de l'image en entrée du RNA 32x32x3 consiste en 3 matrice de 32 lignes et 32 colonnes où chaque valeur i d'une cellule est une valeur qui varie entre 0 et 255 [Figure 6.3].

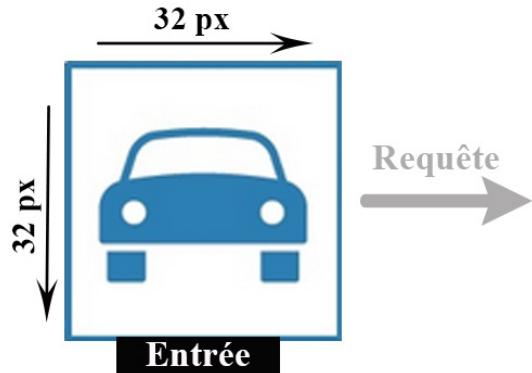


FIGURE 6.2 – Module d’acquisition des données en entrée.

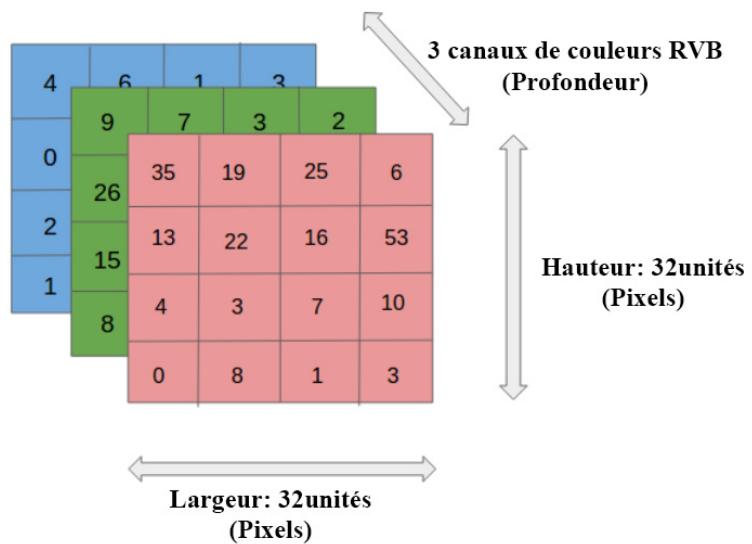


FIGURE 6.3 – Format de la requête.

6.2.1 Choix de notre base de données

L’objet final du système consiste à la reconnaissance d’objets appartenant à notre base de données *cifar-10*. L’ensemble de données CIFAR-10 comprend 60000 images couleur 32x32 dans 10 classes, avec 6000 images par classe [Figure 6.4]. Il y a 50000 images d’entraînement et 10000 images de test. L’ensemble de données est divisé en cinq lots d’entraînement et un lot de test, chacun contenant 10 000 images. Le lot de test contient exactement 1000 images sélectionnées au hasard dans chaque classe. Les lots d’entraînement contiennent les images restantes dans un ordre aléatoire, mais certains lots d’entraînement peuvent contenir plus d’images d’une classe que d’une autre. Entre eux, les lots d’entraînement contiennent exactement 5000 images de chaque classe.



FIGURE 6.4 – CIFAR-10 dataset.

6.3 Module d'extraction de caractéristiques

Dans cette partie, nous allons détailler l'objet principal de notre projet, inspecter la boite noire qui s'agit du réseau de neurones convolutif. Pour l'explication des résultats de prédiction de la boite noire nous avons opté pour la méthode d'extraction de caractéristiques des images dans un CNN.

Le module d'extraction de caractéristiques [Figure 6.5] est composé d'un réseau de neurones convolutif, qui se chargera de l'extraction des caractéristiques apprises (en utilisant l'apprentissage par transfert), avant de les faire passer au MLP pour établir la prédiction finale et au RàPC pour expliquer la prédiction.

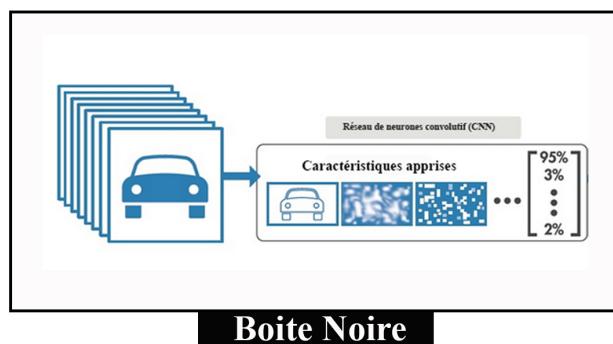


FIGURE 6.5 – Module d'extractions de caractéristiques.

On réalise ainsi un transfert de connaissances acquises par le CNN vers le KNN, d'où le nom. Les connaissances dans notre cas s'agit des caractéristiques apprises par le CNN, on

effectue une extraction de ces caractéristiques pour les transférer à un réseau MLP pour la prédiction et les transférer vers le KNN pour établir l'interprétabilité de la prédiction. Dans notre cas nous avons tiré parti des connaissances du modèle VGG16 entraîné sur 1000 catégories d'objets, et l'avons adapté à un cas d'études plus simple, un modèle qui reconnaîtra 10 catégories d'objets.

6.3.1 Technique d'extraction de caractéristiques

Un exemple d'extraction de caractéristique , d'un CNN entraîné sur l'ensemble de données *ImageNet*, une base de données d'images annotées, nous donnons ce que les filtres de chaque couche du CNN ont appris à reconnaître, ou ce par quoi chaque filtre est activé, dans les [Figures 6.6 & 6.7] :

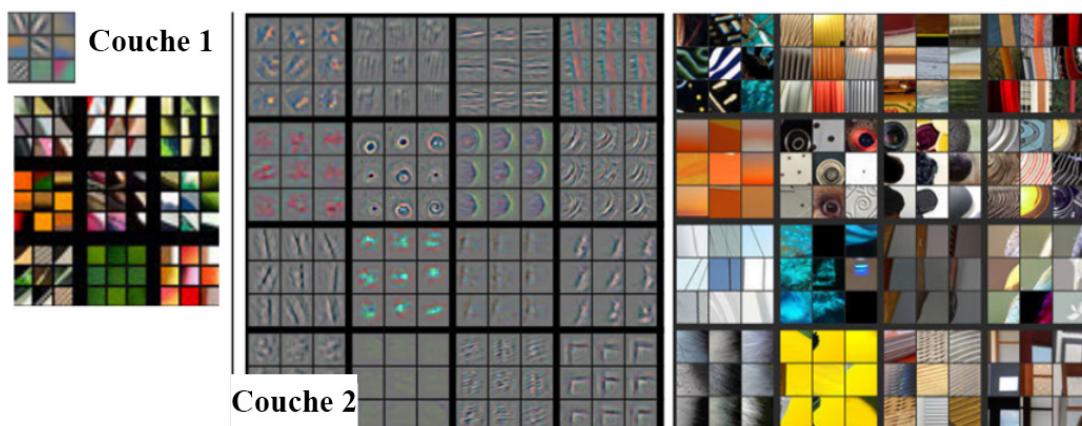


FIGURE 6.6 – Visualisation des couches 1 et 2.

Chaque couche illustre deux images, une qui montre les filtres eux-mêmes et une qui montre quelle partie de l'image est la plus fortement activée par le filtre donné. Par exemple, dans l'espace étiqueté couche 2, nous avons des représentations des 16 filtres différents (à gauche).

- Les filtres des premières couches de convolutions apprennent à reconnaître les couleurs et certaines lignes horizontales et verticales.
- Les couches suivantes apprennent lentement à reconnaître des formes triviales en utilisant les lignes et les couleurs apprises dans les couches précédentes.
- Puis les couches suivantes apprennent à reconnaître des textures, puis des parties d'objets comme les jambes, les yeux, le nez, etc.
- Enfin, les filtres des dernières couches sont activés par des objets entiers tels que des chiens, des voitures, etc.

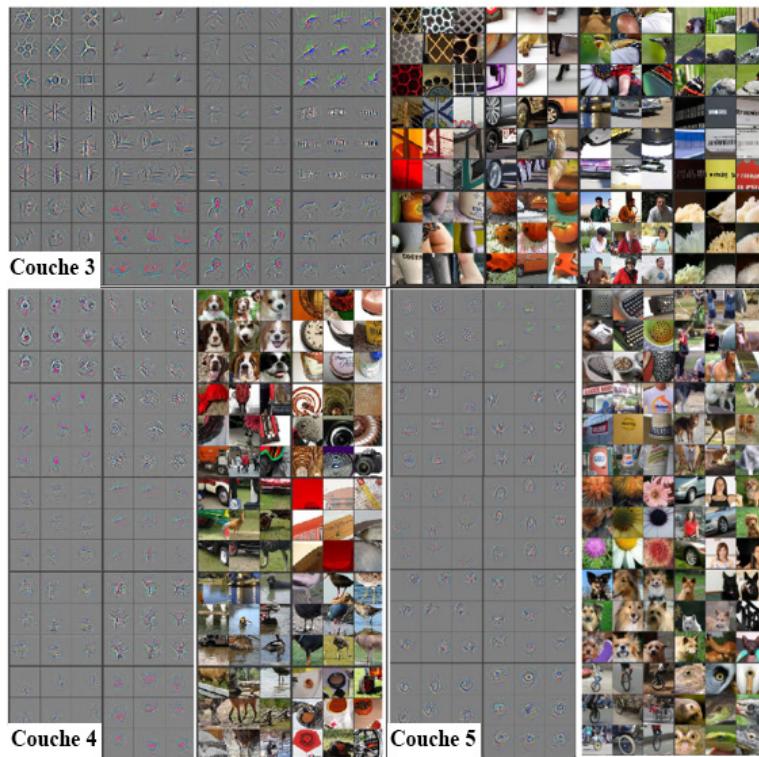


FIGURE 6.7 – Visualisation des couches 3,4 et 5.

Nous utilisons un réseau qui est préformé sur l'ensemble de données *imagenet*. Ce réseau est entraîné pour reconnaître les formes triviales et les petites parties de différents objets. En utilisant Le VGG-16 pour faire l'apprentissage par transfert pour la prédiction, nous ajoutons quelques couches denses à la fin du réseau préformé et nous apprenons quelle combinaison de ces caractéristiques déjà apprises aide à reconnaître les objets de notre nouvel ensemble de données. Pour l'explication des résultats des prédictions, ces caractéristiques extraites sont transférées comme entrées pour entraîner notre modèle d'explication.

Nous entraînons donc que quelques couches denses dans le RNA. De plus, nous utilisons les caractéristiques déjà apprises pour entraîner modèle d'explication. Tout cela contribue à rendre le processus d'apprentissage très rapide et nécessite très peu de données d'apprentissage par rapport à l'apprentissage d'un réseau de convolution à partir de zéro, ainsi qu'il garantie une certaine homogénéité et concomitance entre les deux systèmes de prédiction et d'explication.

Ce processus est effectué grâce à l'extraction de caractéristiques, qui consiste à se servir des caractéristiques du réseau pré-entraîné pour représenter les images du nouveau

problème. Pour cela, on retire la dernière couche fully-connected et on fixe tous les autres paramètres. La partie effectuant l'extraction des caractéristiques va ainsi calculer la représentation de chaque image en entrée à partir des caractéristiques déjà apprises lors du pré-entraînement. On entraîne alors un classificateur, initialisé aléatoirement, sur ces représentations pour résoudre le nouveau problème. Nous récupérons les caractéristiques apprises par le CNN [Figure 6.8] avant de les transmettre aux couches supérieures du réseau.

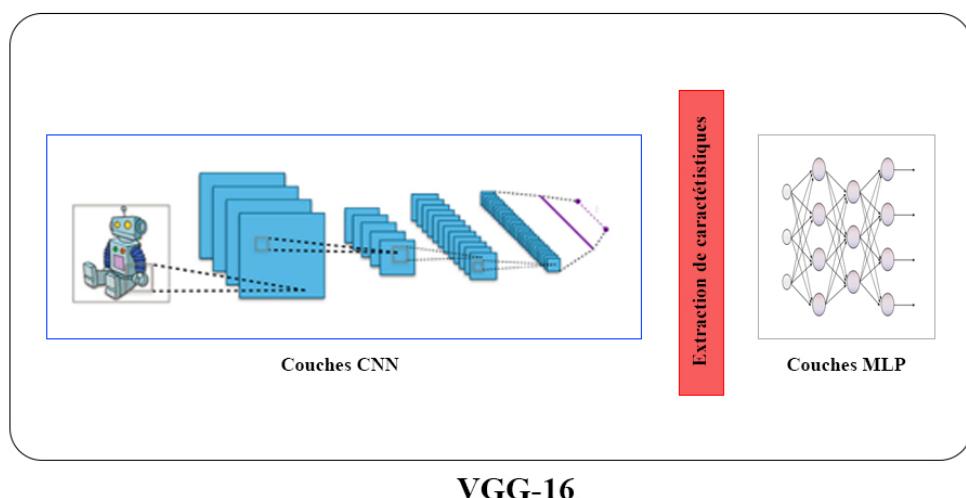


FIGURE 6.8 – Technique d'extraction de caractéristiques.

Cette stratégie doit être utilisée lorsque la nouvelle collection d'images est petite et similaire aux images de pré-entraînement. En effet, entraîner le réseau sur aussi peu d'images est dangereux puisque le risque du surapprentissage¹ est important. De plus, si les nouvelles images ressemblent aux anciennes, elles peuvent alors être représentées par les mêmes caractéristiques.

On résume cette stratégies en 4 étapes :

1. On enlève la dernière couche, i.e. fully connected, du réseau,
2. on « gèle » les poids du modèle et nous les utilisons comme une variable fixe d'extrait,
3. on extrait alors les caractéristiques grâce à la variable fixée pour toutes les images,
4. on entraîne une classification linéaire pour le nouveau dataset.

1. Surapprentissage ou overfitting en Anglais, en statistique, le surapprentissage, ou sur-ajustement, ou encore surinterprétation, est une analyse statistique qui correspond trop étroitement ou exactement à un ensemble particulier de données. Ce phénomène dégrade la performance des algorithmes de l'apprentissage automatique.

6.3.2 Réalisation du réseau de neurones convolutif VGG16

Le VGG-16 est un modèle de réseau neuronal convolutif proposé par (K.Simonyan et A.Zisserman, 2014 [60]), pour la classification et la détection. Le modèle atteint une précision de 92,7 % dans le top 5 des tests d’ImageNet, qui est un ensemble de données de plus de 14 millions d’images appartenant à 1000 classes. Il s’agit de l’un des célèbres modèles soumis à l’ILSVRC-2014. Il apporte l’amélioration par rapport à AlexNet en remplaçant les grands filtres de la taille d’un noyau (11 et 5 dans la première et la deuxième couche convolutionnelle, respectivement) par plusieurs filtres de la taille d’un noyau 3×3 , l’un après l’autre. Le VGG16 a été formé pendant des semaines et utilisait les GPU NVIDIA Titan Black.

6.3.2.1 Architecture globale du réseau VGG-16

L’entrée de la couche conv1 est une image RVB de taille fixe de 224×224 [Figure 6.9]. L’image passe à travers une pile de couches convolutionnelles (conv.), où les filtres ont été utilisés avec un champ de réception 3×3 . Le pas de convolution est fixée à 1 pixel, le padding est de 1 pixel pour les couches convolutionnelles 3×3 . Le pooling est effectuée par cinq couches de pooling, qui suivent certaines des couches convolutionnelles. Le max-pooling est effectuée sur une fenêtre de 2×2 pixels, avec un pas de 2.

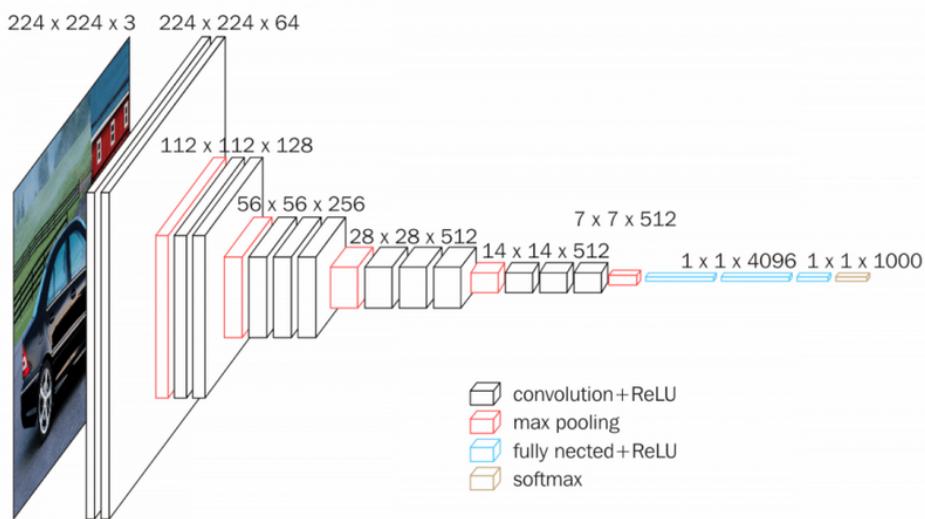


FIGURE 6.9 – Architecture du VGG16.

Trois couches entièrement connectées suivent une pile de couches convolutionnelles : les deux premières couche du MLP ont 4096 neurones chacune, la troisième effectue une classification et contient donc 1000 neurones en sortie (un pour chaque classe). La dernière couche est la couche softmax. La configuration des couches entièrement connectées est la

même dans tous les réseaux. Toutes les couches cachées appliquent la fonction d'activation, ReLU. Dans notre cas, les couches entièrement connectées n'ont pas été chargées, ceci pour effectuer l'extraction des caractéristiques.

Nous allons détailler l'architecture que nous avons adaptée à notre cas d'étude en citant toutes les couches composant le réseaux ainsi que ses paramètre [Figure 6.10]. La taille de l'image en entrée est de 32 x 32 car nous allons utiliser plus tard des images de la base de données Cifar-10 de taille 32 x 32.

Model: "vgg16"		
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 32, 32, 3)	0
block1_conv1 (Conv2D)	(None, 32, 32, 64)	1792
block1_conv2 (Conv2D)	(None, 32, 32, 64)	36928
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	0
block2_conv1 (Conv2D)	(None, 16, 16, 128)	73856
block2_conv2 (Conv2D)	(None, 16, 16, 128)	147584
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	0
block3_conv1 (Conv2D)	(None, 8, 8, 256)	295168
block3_conv2 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv3 (Conv2D)	(None, 8, 8, 256)	590080
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	0
block4_conv1 (Conv2D)	(None, 4, 4, 512)	1180160
block4_conv2 (Conv2D)	(None, 4, 4, 512)	2359808
block4_conv3 (Conv2D)	(None, 4, 4, 512)	2359808
block4_pool (MaxPooling2D)	(None, 2, 2, 512)	0
block5_conv1 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv2 (Conv2D)	(None, 2, 2, 512)	2359808
block5_conv3 (Conv2D)	(None, 2, 2, 512)	2359808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0
<hr/>		
Total params: 14,714,688		
Trainable params: 14,714,688		
Non-trainable params: 0		

FIGURE 6.10 – Architecture du VGG16, sur Keras.

Le VGG16 est composé de cinq blocs de convolution et un bloc de classification [Figure 6.11].

Dans les réseaux de neurones convolutifs disposent d'un nombre fixe de poids régis par

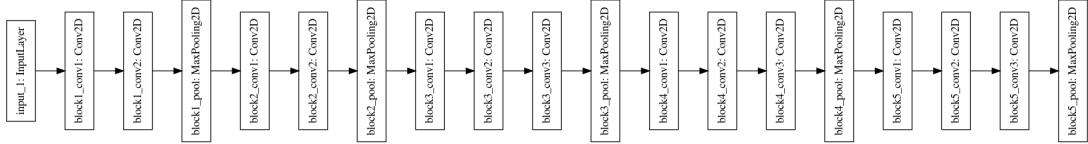


FIGURE 6.11 – Le graphe représentant les couches de convolution du VGG16.

le choix de la taille et du nombre de filtres, mais indépendants de la taille de l'entrée. les filtres appris sont simplement des poids, mais en raison de la structure bidimensionnelle spécialisée des filtres, les valeurs de poids ont une relation spatiale les unes avec les autres et le fait de représenter chaque filtre comme une image bidimensionnelle est significatif. Nous pouvons examiner les filtres dans le modèle, pour voir avec quoi nous devons travailler.

La [Figure 6.10] résume la sortie de chaque couche, et le nombre total de poids par couche.

```

block1_conv1 (3, 3, 3, 64)
block1_conv2 (3, 3, 64, 64)
block2_conv1 (3, 3, 64, 128)
block2_conv2 (3, 3, 128, 128)
block3_conv1 (3, 3, 128, 256)
block3_conv2 (3, 3, 256, 256)
block3_conv3 (3, 3, 256, 256)
block4_conv1 (3, 3, 256, 512)
block4_conv2 (3, 3, 512, 512)
block4_conv3 (3, 3, 512, 512)
block5_conv1 (3, 3, 512, 512)
block5_conv2 (3, 3, 512, 512)
block5_conv3 (3, 3, 512, 512)

```

FIGURE 6.12 – La forme des filtres de chaque bloc de couches de convolutions.

Chaque couche de convolution utilise des filtres en couleurs de taille 3×3 px, déplacés avec un pas de 1 pixel [Figure 6.12]. Le zero-padding vaut 1 pixel. Le nombre de filtres varie selon le "bloc" dans lequel la couche se trouve. De plus, un paramètre de biais est introduit dans le produit de convolution pour chaque filtre. Chaque couche de convolution a pour fonction d'activation une ReLU. L'opération de pooling est réalisée avec des cellules de taille 2×2 px et un pas de 2 px.

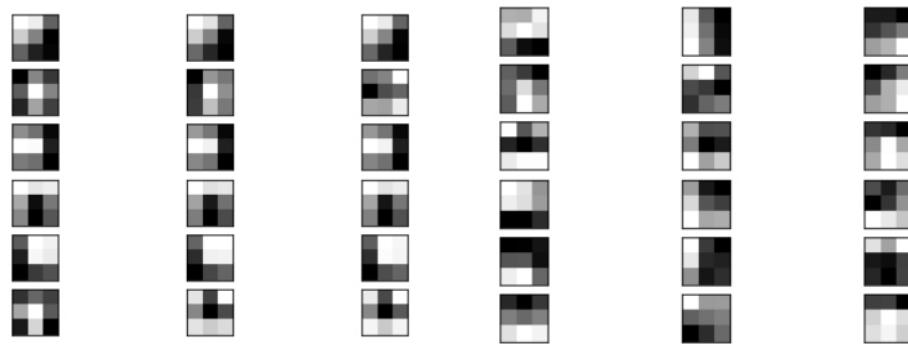
6.3.2.2 Base de donnée d'entraînement du VGG16

VGG16 a été entrainé sur la base de donnée ImageNet qui est un ensemble de données de plus de 15 millions d'images haute résolution étiquetées appartenant à environ 22 000 catégories. Les images ont été collectées sur le web et étiquetées par des étiqueteurs humains à l'aide de l'outil d'Amazon "Mechanical Turk", un outil d'approvisionnement

par la foule. Depuis 2010, dans le cadre du Pascal Visual Object Challenge, un concours annuel appelé ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) a été organisé. L'ILSVRC utilise un sous-ensemble d'ImageNet avec environ 1000 images dans chacune des 1000 catégories. Au total, il y a environ 1,2 million d'images de formation, 50 000 images de validation et 150 000 images de test. ImageNet est constitué d'images à résolution variable. Par conséquent, les images ont été réduites à une résolution fixe de 256×256 . Dans le cas d'une image rectangulaire, l'image est redimensionnée et la zone centrale de 256×256 est recadrée par rapport à l'image résultante.

6.3.2.3 Visualisation des filtres des couches de convolutions

Nous pouvons énumérer les six premiers filtres sur les 64 d'un bloc et tracer les trois canaux de chaque filtre. Nous traçons chaque filtre comme une nouvelle rangée de sous-tracés, et chaque canal ou profondeur de filtre comme une nouvelle colonne [Figure 6.13]. Les valeurs de poids seront probablement de petites valeurs positives et négatives centrées autour de 0,0. Nous pouvons normaliser leurs valeurs dans la plage 0-1 pour les rendre faciles à visualiser.



(a) Filtres de la première couche. (b) Filtres de la dernière couche.

FIGURE 6.13 – Les six premiers filtres sur les 64 filtres de la première et dernière couches du modèle VGG16.

Dans la [Figure 6.13a] nous traçons les six premiers filtres de la première couche convolutionnelle cachée dans le modèle VGG16 et dans la [Figure 6.13b] nous traçons ceux de la dernière couche. Pour chacune nous créons une figure avec six rangées de trois images, soit 18 images, une rangée pour chaque filtre et une colonne pour chaque canal. Nous pouvons voir que dans certains cas, le filtre est le même pour tous les canaux (la première ligne), et dans d'autres, les filtres diffèrent (la dernière ligne). Les carrés foncés indiquent les poids faibles ou inhibiteurs et les carrés clairs les poids élevés. Nous allons détailler plus tard le rôle des filtres de chaque couche.

6.3.2.4 Visualisations des caractéristiques apprises dans le VGG16

Nous pouvons visualiser l'évolution des caractéristiques d'une image en entrée dans le VGG-16. L'idée de visualiser une carte de caractéristiques pour une image d'entrée spécifique serait de comprendre quelles caractéristiques de l'entrée sont détectées ou préservées dans les cartes de caractéristiques. Les cartes de caractéristiques proches de l'entrée détectent des détails de petite taille ou à grain fin, tandis que les cartes de caractéristiques proches de la sortie du modèle capturent des caractéristiques plus générales.

Afin d'explorer la visualisation des cartes de caractéristiques, nous avons besoin d'entrées pour le modèle VGG16 qui peuvent être utilisées pour créer des activations. Nous utiliserons une simple photographie d'une voiture [Figure 6.14].



FIGURE 6.14 – L'image en entrée du VGG-16.

Ensuite, nous devons avoir une idée plus précise de la forme des cartes de caractéristiques produites par chacune des couches convolutives. La [Figure 6.15] permet d'énumérer toutes les couches du modèle et d'imprimer la taille de la sortie ou de la carte des caractéristiques pour chaque couche convulsive ainsi que l'index de la couche dans le modèle. Nous avons les mêmes formes de sortie que celles que nous avons dans le résumé du modèle [Figure 6.11], mais dans ce cas uniquement pour les couches convolutives.

```
1 block1_conv1 (None, 32, 32, 64)
2 block1_conv2 (None, 32, 32, 64)
4 block2_conv1 (None, 16, 16, 128)
5 block2_conv2 (None, 16, 16, 128)
7 block3_conv1 (None, 8, 8, 256)
8 block3_conv2 (None, 8, 8, 256)
9 block3_conv3 (None, 8, 8, 256)
11 block4_conv1 (None, 4, 4, 512)
12 block4_conv2 (None, 4, 4, 512)
13 block4_conv3 (None, 4, 4, 512)
15 block5_conv1 (None, 2, 2, 512)
16 block5_conv2 (None, 2, 2, 512)
```

FIGURE 6.15 – Forme des cartes de caractéristiques du VGG16.

Nous allons maintenant produire une carte des caractéristiques de l'image de la voiture en entrée à partir de la première couche convolutive [Figure 6.16].

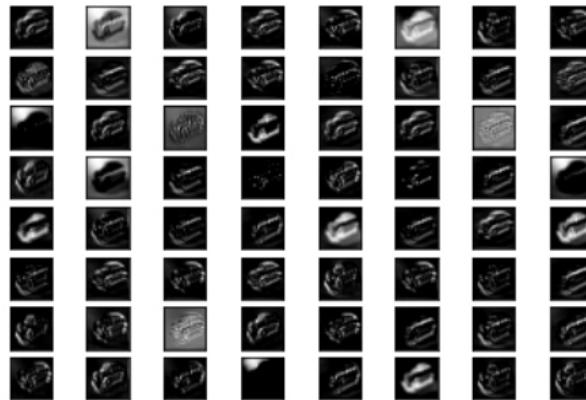
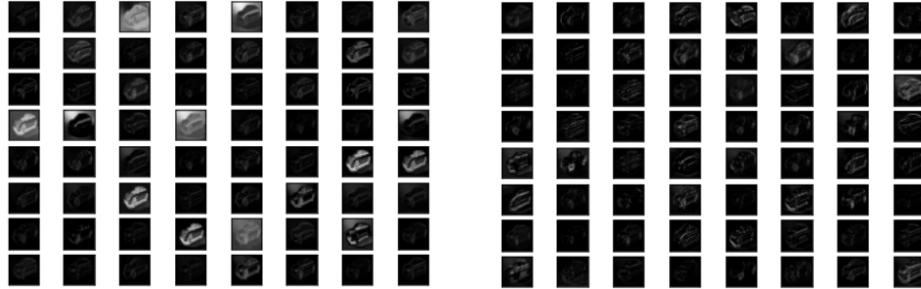


FIGURE 6.16 – Cartes de caractéristiques de la voiture dans la première couche du VGG-16.

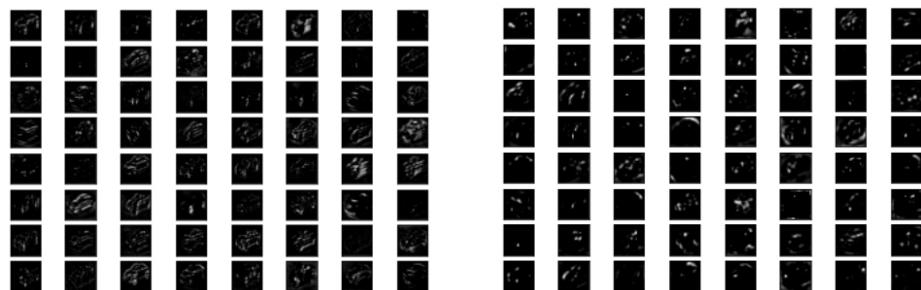
Le résultat de l'application des filtres dans la première couche convolutionnelle est un grand nombre de versions de l'image de la voiture avec différentes caractéristiques mises en évidence. Ainsi, chaque filtre appliqué à une image dans une couche de convolution spécifique est significatif au caractéristiques de cette image.

Les 64 premières cartes de caractéristiques issues de chaque bloc du modèle sont collectées. L'image comporte cinq blocs principaux (par exemple, le bloc 1, le bloc 2, etc.) qui se terminent par une couche de pooling. Les index de la dernière couche convolutionnelle de chaque bloc sont "2, 5, 9, 13, 17" [Figure 6.15].



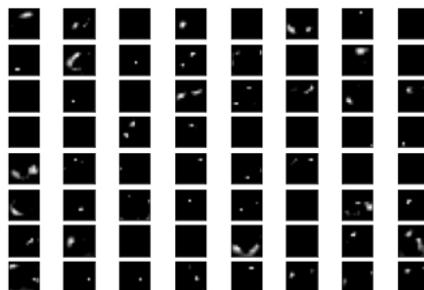
(a) Carte de caractéristiques du bloc 1.

2.



(b) Carte de caractéristiques du bloc 3.

4.



(c) Carte de caractéristiques du bloc 5.

FIGURE 6.17 – Visualisation des 64 premières cartes des caractéristiques extraites dans les cinq blocs de convolutions du VGG-16.

Le modèle extrait les caractéristiques de l'image en concepts plus généraux qui peuvent être utilisés pour faire une classification. Bien que l'image finale ne montre pas clairement que le modèle a vu une voiture, nous perdons généralement la capacité d'interpréter ces cartes de caractéristiques plus profondes [Figure 6.17].

Nous pouvons de même visualiser les 64 premières caractéristiques des images de Cifar-

10 apprises par le VGG-16 dans chaque bloc de convolution [Figure 6.18]. Nous rappelons que les caractéristiques de sortie, se trouvant à la dernière couche de convolution du CNN, seront des entrées d'entraînement du MLP d'un côté et des entrées d'entraînement du KNN.

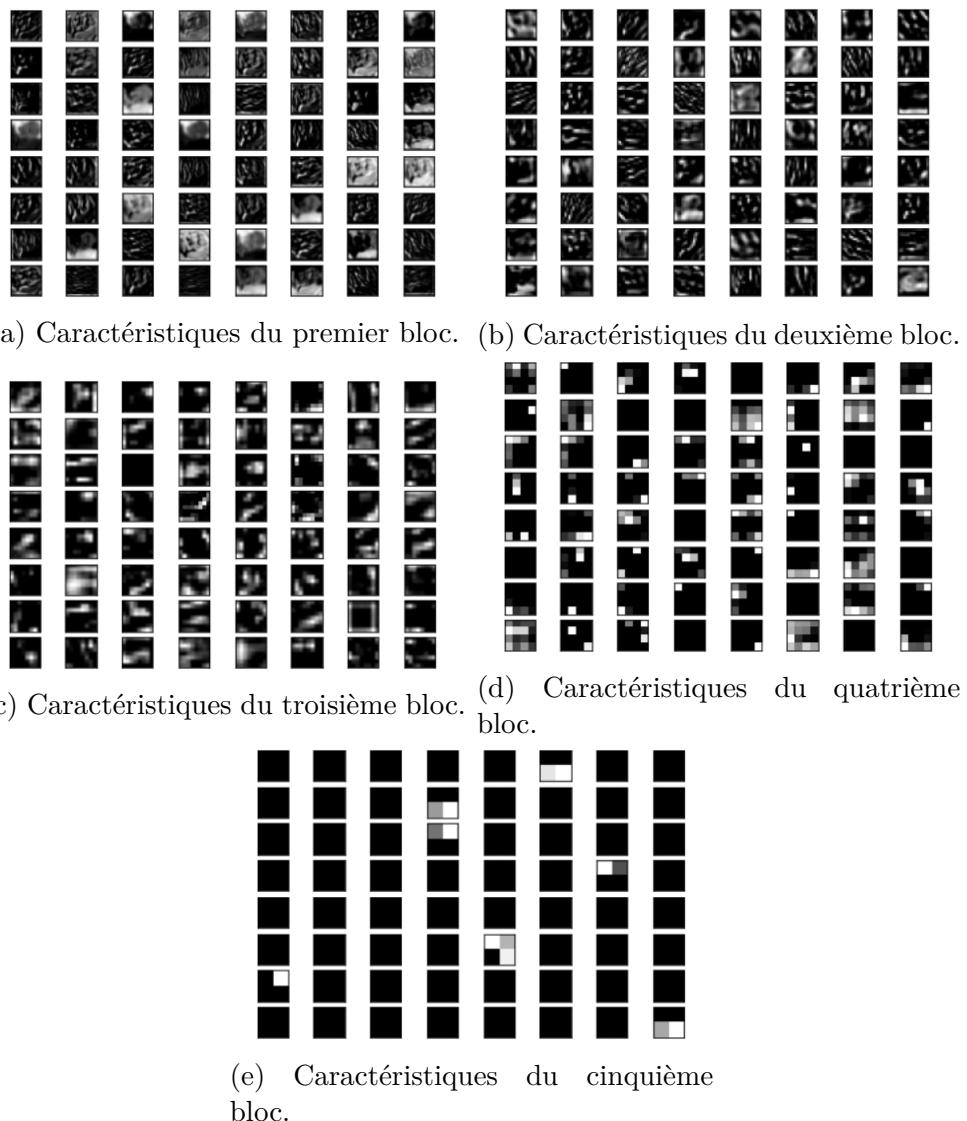


FIGURE 6.18 – Les caractéristique des images de cifar10 apprises dans les blocs de convolution du VGG-16.

6.3.2.5 Calcul de paramètres d'une couche de convolution

Les paramètres dans un réseau de neurones sont des poids qui sont appris pendant l'entraînement et les biais associés à chaque filtre. Ce sont des matrices de poids qui contribuent à la prédiction efficace du modèle, modifié au cours du processus d'apprentissage.

Le nombre de paramètres dans une couche de convolution donnée dans un CNN est le nombre d'éléments "appris" pour un filtre, c'est-à-dire les paramètres du filtre pour cette

couche. Autrement, c'est la somme des nombres de poids plus le nombre du biais.

Le nombre de paramètres dans les couches de convolution dépend du type de la couche :

- **Couche d'entrée** : La couche d'entrée n'a rien à apprendre, elle ne fait que donner la forme de l'image d'entrée. Il n'y a donc pas de paramètres à apprendre ici. Donc $N_{param} = 0$.
- **Couche de convolution** : C'est là que le CNN apprend, donc nous aurons certainement des matrices de poids. Pour calculer les paramètres d'apprentissage, il suffit de faire la multiplication de la largeur m , la hauteur n , la profondeur d du filtre de la couche et de tenir compte du nombres de ces filtres k avec le biais pour chacun des filtres. Le nombre de paramètres dans une couche de convolution serait : $N_{param} = ((m * n * d) + 1) * k$, le 1 représente le biais pour chaque filtre.
- **Couche de Pooling** : Cette couche n'a pas de paramètres à apprendre parce qu'elle ne fait que calculer un nombre spécifique, sans aucun apprentissage. Ainsi, $N_{param} = 0$.
- **Couche entièrement connectée (FC)** : Cette couche a certainement des paramètres à apprendre, en fait, par rapport aux autres couches, cette catégorie de couches a le plus grand nombre de paramètres, car chaque neurone est connecté à tous les autres neurones. Alors, le nombre de paramètres est le produit du nombre de neurones de la couche actuelle c et du nombre de neurones de la couche précédente p en prenant aussi compte du biais. Voici donc le nombre de paramètres : $N_{param} = ((c * p) + 1) * c$.

Prenons exemple de la première couche de convolution "block1_conv1" du premier bloc du VGG-16 [Figure 6.12]. Le filtre correspondant à cette couche est de forme $3 \times 3 \times 3 \times 64$, qui signifie de gauche à droite que le réseau doit apprendre 64 filtres de 3 canaux (RGB) de 3 pixel de hauter et 3 pixel de largeur. Dans ce cas le nombre de paramètre serra $64 \text{ filtre} * 3 \text{ canaux} * \text{hauteur de } 3 * \text{largeur de } 3 + 64 \text{ biais pour chaque filtre}$ on aura $NB_{param} = 3 * 3 * 3 * 64 + 64 = 1792$. On peut vérifier ceci dans la [Figure 6.10]. Pareil pour la dernière couche de convolution "bloc5_conv3", troisième couche du troisième bloc de convolution, le nombre de paramètres serra : $NB_{param} = 3 * 3 * 512 * 512 + 512 = 2359808$.

6.4 Module de classification

Le module de classification s'agit d'un réseau de neurone multicouches, ou MLP, qui se chargera de l'élaboration de la prédiction de l'image en entrée, acquise par le premier module de notre système [Figure 6.19].

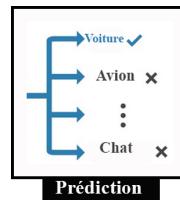


FIGURE 6.19 – Module de classification.

Ce réseau MLP aura comme entrées les caractéristiques extraites par le module précédent, module d'extraction de caractéristiques, pour la phase d'entraînement.

Model: "sequential"		
Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 512)	262656
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 10)	5130

Total params: 267,786
Trainable params: 267,786
Non-trainable params: 0

FIGURE 6.20 – Architecture du MLP.

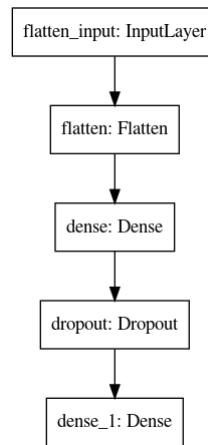


FIGURE 6.21 – Le graphe représentant les couches du MLP.

Dans la [Figure 6.20] est représentée l'architecture de notre réseau MLP à entraîner sur les caractéristiques extraites du CNN. Il est composé de :

- **Une couche entrée** : cette couche recevra un vecteur obtenu de la dernière couche de pooling, comme cette sortie est une matrice une opération d'aplatissement est

effectuée, de la dernière couche de Pooling du CNN qui contient les caractéristiques extraites. Elle est composée de 512 neurones qui correspond au nombre de caractéristiques extraites.

- **Une couche cachée :** Cette couche est composée de 512 neurones et une fonction d'activation **ReLU**.
- **Une couche Dropout :** Le Dropout consiste à ignorer des unités (c'est-à-dire des neurones) pendant la phase d'entraînement de certains ensembles de neurones qui sont choisis au hasard. Il oblige un réseau de neurones à apprendre des caractéristiques plus robustes. Il double le nombre d'itérations nécessaires pour converger. C'est est une approche de régularisation dans les réseaux de neurones qui contribue à réduire l'apprentissage interdépendant entre les neurones et qui évite un surajustement des données de formations (Srivastava et al, 2014 [64]). Le paramètre taux d'une couche Dropout spécifie la probabilité à laquelle les sorties de la couche sont abandonnées, ou inversement, la probabilité à laquelle les sorties de la couche sont retenues. Une valeur commune est une probabilité de 0.5 pour la rétention de la sortie de chaque nœud dans une couche cachée et une valeur proche de 1.0, comme 0.8, pour la rétention des entrées de la couche visible.
- **Une couche de sortie :** Cette couche s'agit de la dernière couche du réseau qui effectue la prédiction. Elle est composée de 10 neurones correspondants aux nombre de classe de prédiction de note dataset, et une fonction d'activation **Softmax**.

Comme nous pouvons constater dans la [Figure 6.20], les deux couches Dense contiennent un nombre de paramètres qui s'agit du nombre de poids et biais appris le réseau. Pour calculer le nombre de paramètres dans une couche il nous suffit de faire le produits du nombre de neurones dans la couche qui la précède et le nombre de ses propres neurones et on ajoute le nombre de biais. pour la première couche Dense le nombre de paramètres est : $NB_{params} = 512 * 512 + 512 = 262656$. Pour la dernière couche de prédiction le nombre de paramètres est : $NB_{params} = 512 * 10 + 10 = 5130$.

6.5 Module d'explication

Nous allons maintenant détailler le modèle d'explication des résultats obtenus par le RNA. Il s'agit du module contenant le système de RàPC, il a comme entrées les caractéristiques extraites du CNN pour entraîner son algorithme afin d'établir une certaine homogénéité avec le MLP de prédiction [Figure 6.22].

Ce module est composé principalement d'une base de cas qui englobe les cas ou expériences du système, un algorithme de mesure de similarité entre un cas présenté et une

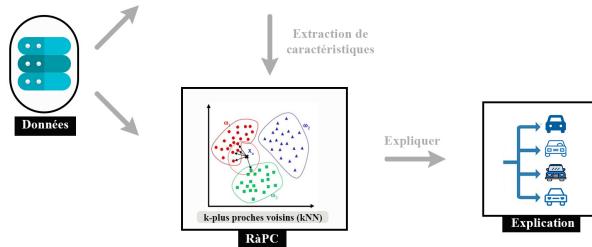


FIGURE 6.22 – Module RàPC.

expérience vue par le système, et une explication visuelle d'un cas.

6.5.1 Base de cas

La base de cas représente la collection des images de la base de donnée Cifar-10 dans le concept de la reconnaissance d'objets. Elle est composée des valeurs des cartes des caractéristiques extraites par le CNN. Chaque carte est dotée d'un index qui correspond à l'image initiale de la base de données Cifar-10.

```

Out[6]: array([[[[38.7392      , 96.80645     , 5.194617     , ..., 0.
                  0.        , 0.        , 111.],
                  [[[22.418543   , 0.          , 0.          , ..., 0.
                  0.        , 0.        , 111.],
                  [[[12.937457   , 0.          , 0.          , ..., 0.
                  0.        , 0.23915878]],],
                  ...,
                  [[[ 0.        , 0.        , 0.        , ..., 0.
                  0.        , 0.        , 111.],
                  [[[ 0.        , 0.        , 0.        , ..., 0.
                  0.        , 0.        , 111.],
                  [[[11.946317   , 0.          , 0.          , ..., 0.
                  0.        , 0.        , 111.], dtype=float32)
  
```

FIGURE 6.23 – Forme de la base de cas.

La [Figure 6.23] montre que la base de cas est sous forme de 50000 vecteurs de 512 valeurs chacun. Elle est d'une structure plate où chaque cas est représenté par des valeurs numériques qui indiquent les parties de l'image qui contribuent le plus à son identité.

La [Table 6.1] montre l'organisation de notre base de cas. La colonne la plus à gauche

Attributs	ID_Cas						
	Grenouille	Camion	Camion	...	Camion	Voiture	Voiture
I1	38,7392	22,418543	12,937457	...	0,0	0,0	11,946317
I2	96,80645	0,0	0,0	...	0,0	0,0	0,0
I3	5,194617	0,0	0,0	...	0,0	0,0	0,0
.
.
.
I510	0,0	0,0	0,0	...	0,0	0,0	0,0
I511	0,0	0,0	0,0	...	0,0	0,0	0,0
I512	0,0	0,0	0,23915878	...	0,0	0,0	0,0

TABLE 6.1 – Structure de la base de Cas.

représente les index des vecteurs contenant les caractéristiques des cas. Les identifiants des cas correspondent aux noms des classes, ou labels. Sachant qu'il n'y a pas de relation entre les cas. A chaque fois qu'un cas est représenté au système, l'algorithme d'explication se charge de calculer la mesure de similarité entre le cas présenté et les valeurs des expériences dans la base de cas représentés dans les colonnes de notre table.

6.5.2 Description d'un cas

Le cas correspond à la requête présentée à notre système pour élaborer l'explication qui lui est associée. La forme des cas dans notre système est un vecteur à 1D de caractéristiques extraites par le CNN. Cette approche de représentation d'un cas est basée sur les caractéristiques de l'image (ou features-based approach, en Anglais). Elle décrit les propriétés les plus marquantes de l'image par des valeurs statistiques (Petra Perner, 2008 [62]).

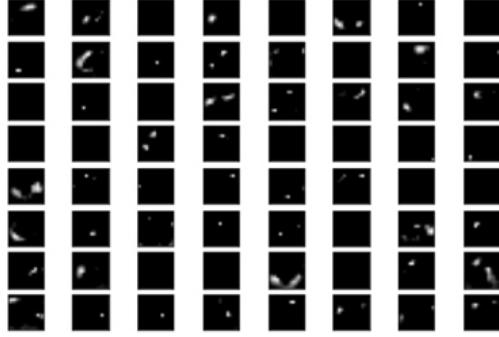
Dans la [Figure 6.24] nous avons illustré les valeurs d'un cas par l'approche basée sur les caractéristiques. La [Figure 6.24 à droite] montre les 64 premières caractéristiques parmi les 512 d'un cas. La [Figure 6.24 à gauche] montre une partie des valeurs correspondantes à ces caractéristiques obtenues par les opérations de convolution du module de la boîte noire.

6.5.3 Algorithme des plus proches voisins

Pour la détection des cas les plus similaires, nous utilisons l'algorithme des K plus proches voisins. Nous pouvons résumer le fonctionnement de l'algorithme KNN pour une tache de classification comme suit :

- Soit (X_i, C_i) où $i = (1, 2, \dots, n)$ étant des points de données.

```
array([[11.340129 ,  0.        ,  0.        ,  0.        ,  5.888249 ,
       34.36422 ,  0.        ,  0.        ,  0.        , 12.508782 ,
       0.        ,  0.        , 15.650603 ,  0.        ,  0.        ,
       0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
       0.        ,  0.        , 13.850389 ,  0.        ,  0.        ,
       0.        , 17.865393 ,  0.        ,  0.        ,  0.        ,
      39.923977 , 31.239395 ,  0.        ,  0.        ,  0.        ,
       0.        ,  0.        ,  0.        ,  0.        , 70.230034 ,
       0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
       0.        , 58.674603 ,  0.        , 43.50447 ,  0.        ,
       0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
       0.        ,  0.        ,  0.        ,  0.        , 0.45338455,
       0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
       0.        ,  0.        , 62.4943  ,  0.        ,  0.        ,
       0.        , 28.19247 ,  0.        ,  0.        , 14.089926 ,
     11.700225 ,  0.        ,  0.        ,  0.        ,  0.        ,
       0.        , 20.264854 ,  0.        ,  0.        ,  0.        ,
       0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
       0.        , 13.478817 ,  0.        ,  0.        , 44.2962  ,
       0.        ,  0.        ,  0.        ,  0.        ,  0.        ,
       0.        ,  0.        ,  0.        , 23.719486 ,  0.        ]]
```



Les 64 premières cartes de caractéristiques d'un cas.

Les valeurs des caractéristiques obtenues par le module de boîte noire.

FIGURE 6.24 – Représentation d'un cas basée sur les caractéristiques.

- X_i désigne les valeurs des caractéristiques et C_i désigne les étiquettes pour X_i pour chaque i .
- En supposant que le nombre de classes est " c ", on obtient $C_i \in (1, 2, 3, \dots, c)$ pour toutes les valeurs de i .

Soit x un point pour lequel l'étiquette n'est pas connue, et nous voudrions trouver la classe d'étiquette en utilisant l'algorithme des k plus proche voisin.

1. Calculer $d(x, x_i)$ avec $i = (1, 2, \dots, n)$, où d désigne la distance euclidienne entre les points.
2. Disposer les n distances euclidiennes calculées dans un ordre non décroissant.
3. Soit k est un entier, prendre les k premières distances de cette liste triée.
4. Trouver les k points correspondant à ces $k - distances$.
5. Soit k_i désigne le nombre de points appartenant à la $i^{\text{ème}}$ classe parmi les k points, c'est-à-dire $k \geq 0$.
6. Si $k_i > k_j, \forall i \neq j$ alors mettre x dans la classe i .

6.6 Conclusion

Dans ce chapitre, nous avons détaillé la conception de notre solution proposé au problème d'explication des résultats obtenus par un réseau de neurones artificiel. Nous avons utilisé notre base de données pour extraire les caractéristiques des images en entrée dans un CNN à travers le transfert de connaissance d'un réseau pré-entraîné vers un classificateur que nous avons conçu. Ainsi, nous avons transférer ces caractéristiques pour entraîner

l’algorithme qui retrouve les cas similaires. Dans ce qui suit, nous allons entamer l’implémentation de notre solution ainsi que les tests et les résultats obtenus.

Implémentation de la solution

Dans ce chapitre, nous allons entamer la partie réalisation de la solution. Nous allons mettre en oeuvre notre solution en utilisant des outils et technologies qui facilitent cette étape. Nous allons choisir un langage et les bibliothèques associées au domaine de l'intelligence artificielle et de l'apprentissage automatique. De plus, nous allons détailler toutes les étapes suivies pour réaliser notre système d'explication des préditions d'un RNA.

7.1 Technologies et outils

Pour implémenter notre solution, nous avons utilisé des technologies et outils relatifs au domaine d'apprentissage automatique. Notre choix est basé sur la nouveauté et réputation, comptabilité avec le domaine d'études, la maîtrise des langages, ainsi que l'accès à la documentation.

Le langage Python est opensource placé sous une licence libre proche de la licence BSD et fonctionne sur la plupart des plates-formes informatiques, des smartphones aux ordinateurs centraux, de Windows à Unix avec notamment GNU/Linux en passant par macOS, ou encore Android, iOS. Il est conçu pour optimiser la productivité des programmeurs en offrant des outils de haut niveau et une syntaxe simple à utiliser. Il est également apprécié par certains pédagogues qui y trouvent un langage où la syntaxe, clairement séparée des mécanismes de bas niveau, permet une initiation aisée aux concepts de base de la programmation. Il dispose d'un ensemble de bibliothèques mathématiques et statistiques (ex : Numpy) ainsi que celles de l'apprentissage automatique (ex : keras, Sickit-learn).



FIGURE 7.1 – Python.

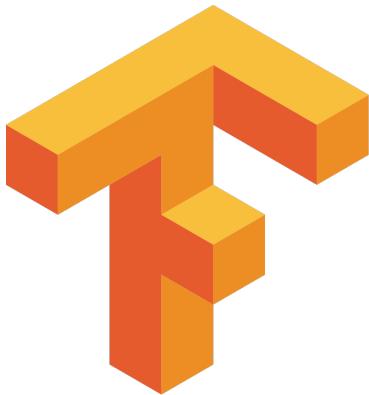


FIGURE 7.2 – TensorFlow.

TensorFlow est une plate-forme Open Source de bout en bout dédiée au machine learning. Elle propose un écosystème complet et flexible d'outils, de bibliothèques et de ressources communautaires permettant aux chercheurs d'avancer dans le domaine du machine learning, et aux développeurs de créer et de déployer facilement des applications qui exploitent cette technologie. TensorFlow est développé par Google. Le code source a été ouvert le 9 novembre 2015 par Google et publié sous licence Apache.

Il est fondé sur l'infrastructure DistBelief, initiée par Google en 2011, et est doté d'une interface pour Python et Julia. TensorFlow est l'un des outils les plus utilisés en IA dans le domaine de l'apprentissage automatique.

La bibliothèque Keras permet d'interagir avec les algorithmes de réseaux de neurones profonds et d'apprentissage automatique, notamment Tensorflow, Theano, Microsoft Cognitive Toolkit ou PlaidML. Conçue pour permettre une expérimentation rapide avec les réseaux de neurones profonds, elle se concentre sur son ergonomie, sa modularité et ses capacités d'extension. Keras suit les meilleures pratiques pour réduire la charge cognitive, elle offre des API cohérentes et simples, elle minimise le nombre d'actions de l'utilisateur nécessaires pour les cas d'utilisation courante et elle fournit des messages d'erreur clairs et exploitables.

Elle dispose également d'une vaste documentation et de guides du développeur. Construit sur TensorFlow 2.0, Keras est un cadre d'apprentissage solide qui peut s'adapter à de grandes grappes de GPU ou à un pod TPU entier. Keras est un élément central de l'écosystème TensorFlow 2.0, qui couvre chaque étape du processus d'apprentissage de la machine, de la gestion des données à l'entraînement aux hyperparamètres en passant par les solutions de déploiement.

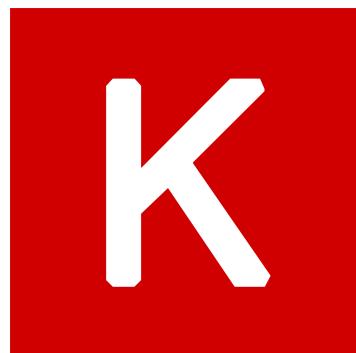


FIGURE 7.3 – Keras.



FIGURE 7.4 – Scikit-learn.

Scikit-learn est une bibliothèque libre Python destinée à l'apprentissage automatique. Elle est développée par de nombreux contributeurs notamment dans le monde académique par des instituts français d'enseignement supérieur et de recherche comme Inria. Elle comprend notamment des fonctions pour estimer des forêts aléatoires, des régressions logistiques, des algorithmes de classification, et les machines à vecteurs de support. Elle est conçue pour s'harmoniser avec d'autres bibliothèques libres Python, notamment NumPy et SciPy.

Joblib permet de sauvegarder des structures de données dans un fichier pour pouvoir les recharger plus tard ainsi de lancer des tâches en parallèle. Joblib est optimisé pour être rapide et robuste sur de grandes données en particulier et possède des optimisations spécifiques pour les tableaux numérisés. Il est sous licence BSD. L'objectif est de fournir des outils permettant d'obtenir facilement de meilleures performances et une meilleure reproductibilité lors de travaux de longue durée.

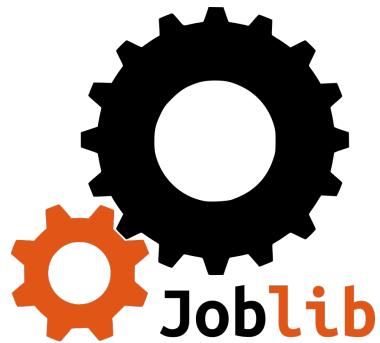


FIGURE 7.5 – Joblib.



FIGURE 7.6 – Matplotlib.

Matplotlib est une bibliothèque du langage de programmation Python destinée à tracer et visualiser des données sous formes de graphiques. Elle peut être combinée avec les bibliothèques python de calcul scientifique NumPy et SciPy. Matplotlib est distribuée librement et gratuitement sous une licence de style BSD. Sa version stable actuelle (la 2.0.1 en 2017) est compatible avec la version 3 de Python. Plusieurs points rendent cette bibliothèque intéressante : Export possible en de nombreux formats matriciels (PNG, JPEG...) et vectoriels (PDF, SVG...), Documentation en ligne en quantité, nombreux exemples disponibles sur internet, Forte communauté très active, Interface pylab qui reproduit fidèlement la syntaxe MATLAB, et Bibliothèque haut niveau idéale pour

le calcul interactif.

NumPy est une extension du langage de programmation Python, destinée à manipuler des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces tableaux. Plus précisément, cette bibliothèque logicielle libre et open source fournit de multiples fonctions permettant notamment de créer directement un tableau depuis un fichier ou au contraire de sauvegarder un tableau dans un fichier, et manipuler des vecteurs, matrices et polynômes. Distribué sous une licence BSD libérale, NumPy est développé et maintenu publiquement sur GitHub par une communauté dynamique, réactive et diversifiée.

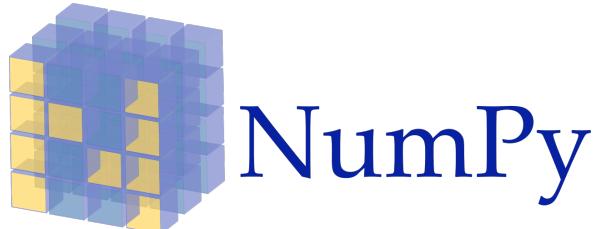


FIGURE 7.7 – NumPy.



FIGURE 7.8 – Pandas.

Pandas est une bibliothèque écrite pour le langage de programmation Python permettant la manipulation et l'analyse des données. Elle propose en particulier des structures de données et des opérations de manipulation de tableaux numériques et de séries temporelles. Pandas est un logiciel libre sous licence BSD. Elle est aussi utilisé pour la conversion et sauvegarde de structure de données.

Google Cloud Platform (GCP) est une plateforme de cloud computing fournie par Google, proposant un hébergement sur la même infrastructure que celle que Google utilise en interne pour des produits tels que son moteur de recherche. Cloud Platform fournit aux développeurs des produits permettant de construire une gamme de programmes allant de simples sites web à des applications complexes. Google Cloud Platform fait partie d'un ensemble de solutions pour les entreprises appelé Google Cloud,



Google Cloud Platform

FIGURE 7.9 – Google Cloud Plateform.

et fournit des services modulaires basés sur le cloud, tels que le stockage d'informations, le calcul, des applications de traduction et de prévision, etc. Elle offre des plates-formes de développement des algorithmes d'IA, ou nous pouvons louer des machines virtuelles avec des puissances de calculs considérables sans se soucier de l'installation ou mise à jour des bibliothèques telles que TensorFlow ou des langages tel que le Python.



FIGURE 7.10 – AI Platform Notebooks .

AI Platform Notebooks est un service géré offrant un environnement JupyterLab intégré et sécurisé au sein duquel les data scientists et les développeurs en machine learning peuvent expérimenter, développer et déployer des modèles en production. Les utilisateurs peuvent créer des instances exécutant JupyterLab en un seul clic. Les frameworks de machine learning et de science des données les plus récents y sont pré-installés. L'IDE intégré JupyterLab est une application web utilisée pour programmer dans plus de 40 langages de

programmation, dont Python, Julia, Ruby, R, ou encore Scala. Jupyter est une évolution du projet IPython. Jupyter permet de réaliser des calepins ou notebooks, c'est-à-dire des programmes contenant à la fois du texte en markdown et du code en Julia, Python, R... etc. Ces calepins sont utilisés en science des données pour explorer et analyser des données.

7.2 Étapes de réalisation de notre système

La [Figure 7.11] représente l'architecture technique de notre système proposé donné dans la [Figure 6.1], au niveau d'implémentation. Chaque étape de réalisation de notre système fait appel à plusieurs méthodes et techniques relatives à notre domaine d'application sur l'environnement de Python. Nous allons détailler dans cette section chaque étape de réalisation, allant du chargement des données d'entrée jusqu'à l'obtention du résultat. Nous allons montré la façon avec laquelle chaque bibliothèque contribue à la réalisation d'une tache bien particulière.

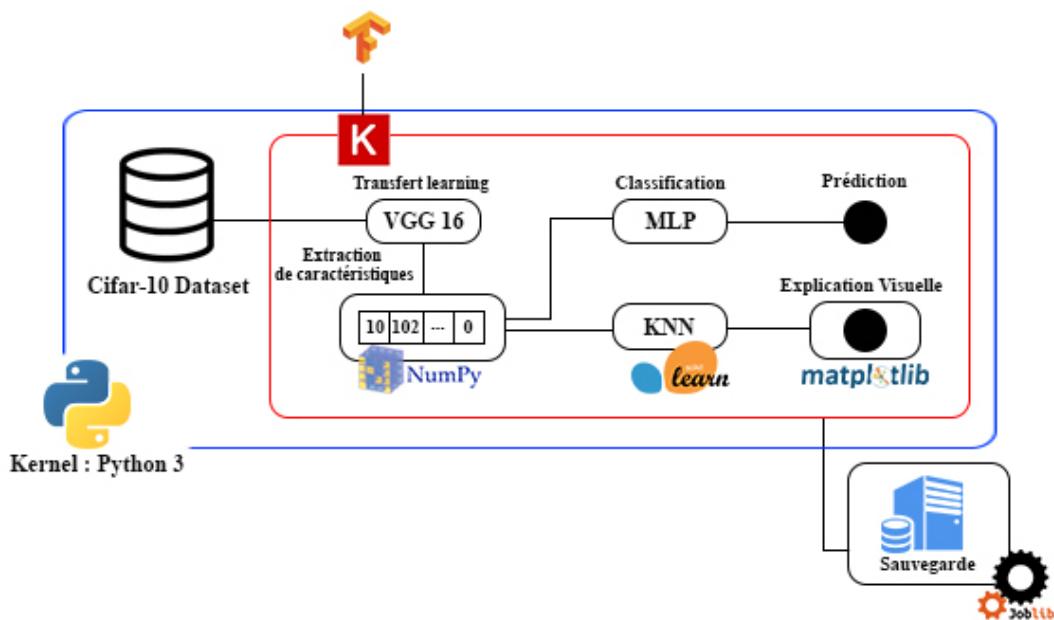


FIGURE 7.11 – Architecture technique.

7.2.1 Chargement des données

Comme tout autre modèle d'apprentissage automatique nous avons besoin d'une base de données d'entraînement. Dans notre cas la phase de chargement des données s'agit du chargement du dataset **Cifar-10** composé de 50000 images d'entraînement et 10000 images de test, de format 32 x 32 x 3. Chaque image est dotée d'un label, entier de 0 à 9, chaque label représente la classe à laquelle appartient cette image. Ce processus est réalisé en utilisant les modules et méthodes de keras suivants :

- ***keras.datasets*** : Ce module fournit quelques jeux de données (déjà vectorisés, au format Numpy) qui peuvent être utilisés pour déboguer un modèle ou créer des exemples de code simples.
- ***cifar10.load_data()*** : Cette fonction charge les données d'images dans deux couples de variables "données et labels" pour l'entraînement et le test.

7.2.2 Extraction de caractéristiques

Dans la phase d'extraction de caractéristiques nous avons besoin de charger les couches de convolution du modèle pré-entraîné **VGG16**. Après le chargement du modèle nous allons passer les données pour lesquelles nous effectuerons l'extraction de caractéristiques. Ce processus est effectué grâce aux modules et fonctions suivants :

- ***keras.applications*** : Les applications de Keras sont des modèles d'apprentissage profond qui sont mis à disposition à côté de poids préformés. Ces modèles peuvent être utilisés pour la prédiction, l'extraction de caractéristiques et le réglage fin.

- ***vgg16.preprocess_input()*** : cette fonction est destinée à adapter nos images au format requis par le modèle. Certains modèles utilisent des images dont les valeurs vont de 0 à 1. D'autres de -1 à +1. D'autres utilisent le style "caffe¹", qui n'est pas normalisé, mais centré.
- ***vgg16.VGG16()*** : Cette fonction charge une instance du model VGG16. Elle nous offre aussi la possibilité de charger que les couches de convolution sans inclure les 3 couches entièrement connectées au sommet du réseau.
- ***predict()*** : Cette fonction nous permet d'activer le processus de convolution du modèle pour extraire les caractéristiques des données en entrée. Les caractéristiques sont chargées dans un tableau NumPy de 50000 cellules où chaque cellule contient 512 valeurs.

7.2.3 Modèle MLP

Dans cette étape nous allons construire le modèle de classification composé des couches MLP qui sera entraîné avec les caractéristiques extraites du modèle précédent. Ce processus est effectué en utilisant les modules et fonctions suivantes :

- ***keras.models*** : Ce module contient plusieurs types de modèles. Il y a trois façons de créer des modèles Keras : Le modèle séquentiel, L'API fonctionnelle de Keras, et le sous-classement de modèle.
- ***keras.layers*** : Ce module contient plusieurs types de couches : Dense, Flatten, Dropout, Conv2D, ... etc.
- ***add()*** : Cette fonction nous permet d'ajouter une couche à notre modèle en précisant le type de cette couche, le nombre des noeuds, la fonction d'activation, ... etc.
- ***compile()*** : Cette fonction nous permet de configurer notre modèle en précisant la fonctions d'erreur, l'optimiseur, et les métriques.
- ***fit()*** : Cette fonction lance le processus d'apprentissage en ayant plusieurs paramètres : Données d'entraînement, le nombre de tours d'entraînement, données de validation ou de test, nombre de fois de validation, ... etc.

7.2.4 Modèle KNN

Dans cette phase nous allons implémenter l'algorithme qui regroupe les principes du RàPC. Cette phase consiste à récupérer les cas similaires d'une requête de prédiction. L'algorithme KNN est entraîné avec les caractéristiques extraites de notre dataset qui

1. Les images sont converties de RVB en BGR, puis chaque canal de couleur est centré sur zéro par rapport à l'ensemble de données ImageNet, sans mise à l'échelle

sont déjà utilisées pour entraîner le MLP ce qui offre une certaine homogénéité entre le modèle de prédiction et celui d'explication. Ce processus est effectué en utilisant les modules et fonctions de la bibliothèques Scikit-learn suivants :

- **`sklearn.neighbors`** : Ce module offre des fonctionnalités pour des méthodes d'apprentissage basées sur les voisins, supervisées ou non-supervisées, KDTree, BallTree, KNN Classifier, ... etc.
- **`KNeighborsClassifier()`** : Cette fonction donne une instance du classificateur qui implémente les k-plus proches voisins par vote. Elle admet plusieurs paramètres : nombre de voisins, le type des poids utilisés pour la prédiction, l'algorithme qui calcule les voisins, la métrique de distance, ... etc.
- **`fit()`** : Cette fonction lance le processus d'entraînement du KNN en ayant comme paramètres les données d'entraînement et les labels correspondants.
- **`kneighbors()`** : Cette fonction retrouve les "k" voisins pour un point, elle renvoie les indices des voisins et les distances entre le point et chaque voisin. Elle prend comme paramètre : le point de requête, nombre de voisins, et booléen pour retourner la distance ou non.

7.2.5 Visualisation et Sauvegarde

Pour ce qui concerne toutes les visualisation dans notre système, allant des graphes de tests, illustration des images, schéma et graphes des modèles, illustration des filtres et caractéristiques, aux explications visuelles des prédictions, nous avons exploiter le module **`matplotlib.pyplot`** contenant la fonction de traçage de graphe **`plot()`**, et la fonction d'affichage d'image **`imshow()`**.

Pour la sauvegarde et le chargement des structures des modèles, les poids, des caractéristiques et des labels, nous avons utilisé les fonctions **`dump()`**, et **`load()`**, de **Joblib** ainsi que les fonction **`DataFrame()`**, et **`to_json()`**, de **Pandas**. Pour le parcours de fichiers nous avons utilisé le module **`pathlib`** de **Python**.

7.3 Conclusion

Dans ce chapitre, nous avons exposé les principales techniques que nous avons utilisé pour réaliser notre système d'explication des prédictions des résultats obtenus par les RNA en utilisant le RàPC. Nous avons synthétisé l'architecture technique de notre solution en premier lieu. Puis, nous avons cité les différents langages et outils de programmation ainsi que les modules et fonctions principales pour implémenter chaque module composant notre solution. Dans le chapitre suivant, nous allons présenter les différents tests que nous avons effectués pour valider notre solution.

Test et Évaluation

Dans ce chapitre, nous allons exposer les différents tests effectués pour réaliser notre solution. Nous allons présenter pour chaque module présenté dans le chapitre précédent les métriques utilisées pour mesurer ses performances. Nous allons ensuite exposer les résultats obtenus après avoir fixé les hyper-paramètres.

8.1 Environnement de Test

Notre solution a été implémentée et testé sur **Google Cloud Console**, sur une machine virtuelle de types de ressources suivants :

- **Processur** : 4vCPUs,
- **RAM** : 15 GB,
- **Disque Dur** : 100 GB
- **Environnement** : Python3, Tensorflow 2.2.0.

8.2 Métriques d'évaluations

Afin de mesurer les performances d'un modèle de Machine Learning, on utilise généralement la matrice de confusion, aussi appelée tableau de contingence [Figure 8.1]. Une Matrice de confusion est un résumé des résultats de prédictions sur un problème de classification. Les prédictions correctes et incorrectes sont mises en lumière et réparties par classe. Les résultats sont ainsi comparés avec les valeurs réelles. De cette matrice on tire différentes mesures telles que l'aire sous courbe (ROC), la précision (P), le rappel (R) ou taux de détection (DR), et le score (F). Ces mesures sont calculées en utilisant quatre mesures principales :

- **Vrai positif (VP)** : les cas où la prédition est positive, et où la valeur réelle est effectivement positive.
- **Vrai négatif (VN)** : les cas où la prédition est négative, et où la valeur réelle est effectivement négative.
- **Faux positif (FP)** : les cas où la prédition est positive, mais où la valeur réelle est négative.

- **Faux négatif (FN)** : les cas où la prédiction est négative, mais où la valeur réelle est positive.

		Classe Prédite	
		Postif	Négatif
Classe Réelle	VRAI	VP	FN
	FAUX	FP	VN

FIGURE 8.1 – Matrice de confusion.

- **Taux de réussite (ACC)** : ou l'exactitude, est la proportion du nombre total de prédictions correctes par rapport à la taille réelle de l'ensemble de données.

$$ACC = \frac{VP + VN}{FP + FN + VP + VN} \quad (8.1)$$

- **Précision (P)** : c'est la proportion de prédictions correctes parmi les points que l'on a prédits positifs.

$$P = \frac{VP}{VP + FP} \quad (8.2)$$

- **Le rappel (R)** : ou le taux de vrais positifs (TVP), c'est à dire la proportion de positifs que l'on a correctement identifiés.

$$R \text{ (ou TVP)} = \frac{VP}{VP + FN} \quad (8.3)$$

- **F-score** : Définit la moyenne harmonique de la précision et le rappel, et représente le point où les deux sont égaux. Définit la qualité d'une classification en fonction d'une classe, mais ne tient pas compte de l'éventuel déséquilibre entre les classes.

$$F - score = 2 \times \frac{P \times R}{P + R} \quad (8.4)$$

- **Le taux de faux positifs (TFP)** : représente la proportion de nombre des positifs mal classés par rapport au nombre de négatifs correctement classés.

$$TFP = \frac{FP}{FP + VN} \quad (8.5)$$

- **Courbe ROC (Receiver Operating Characteristic)** : est un graphe représentant les performances d'un modèle de classification pour tous les seuils de classification. Cette courbe trace le taux de vrais positifs (ordonnée) en fonction du taux de faux positifs (abscisse).

- **Aire sous la courbe (Area Under Curve (AUC)) :** C'est l'aire sous la courbe ROC, elle est souvent calculée pour estimer la performance d'un modèle de détection indépendamment du seuil de détection, plus que sa valeur est proche de 1 plus que le modèle est performant [Figure 8.2].

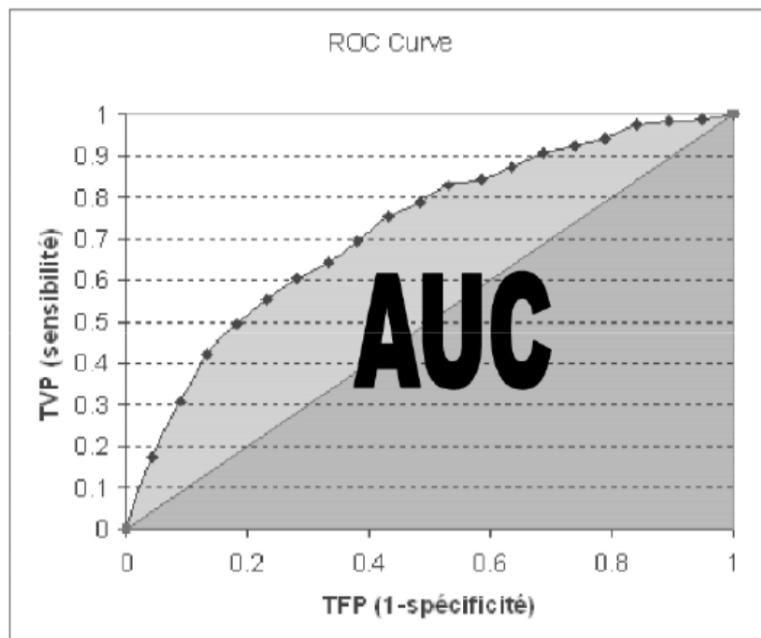


FIGURE 8.2 – Aire sous la courbe AUC.

8.3 Choix des hyper-paramètres

Dans les algorithmes d'apprentissage, il est nécessaire de trouver les bon paramètres d'un modèle qui feront de lui un modèle plus précis et plus robuste par rapport aux autres. Le choix de ces paramètres n'est pas une généralisation sur tous les modèles d'apprentissage, mais il est en relation avec le domaine d'études et les données dont nous disposons pour entraîner notre modèle. Dans cette section, nous allons présenter les différents tests effectués sur nos modèles et les résultats obtenus en variant leurs paramètres. Le tableau dans la [Figure 8.3] illustre pour chacun de nos modèles les paramètres qui lui sont associés et la métrique utilisée pour l'évaluation.

8.3.1 Évaluation du modèle MLP

Nous allons maintenant illustrer les valeurs de l'évaluation obtenues pour le modèle MLP en variant chaque paramètre, et nous allons expliquer la signification de chaque paramètre utilisé.

Modèle	Paramètres à ajuster	Métrique d'évaluation
MLP (pour la prédiction de classe)	Nombre de couches cachées Le taux de perturbation (Dropout) Nombre d'étapes d'apprentissage Nombre d'itérations par étape d'apprentissage Nombre d'étapes de validation	Précision d'entraînement Taux d'erreur d'apprentissage Taux d'erreur de validation
KNN (pour l'explication)	Nombre de voisins Algorithme de calcul de voisins Fonction de pondération Métrique de Distance	Score ROC AUC

FIGURE 8.3 – Paramètres d'évaluation.

Tout d'abord nous allons définir les configurations utilisées pour compiler notre modèle MLP :

- **Fonction d'activation** : les fonctions d'activation utilisées dans notre modèle sont :
 - **relu** : cette fonction est utilisée dans la couche cachée de notre modèle.
 - **softmax** : c'est une fonction qui est utilisée dans la dernière couche du modèle lorsqu'on dispose de plus de 2 neurones d'activation (dans notre cas 10 classes).
- **L'algorithme d'optimisation adam**¹ : L'optimisation d'Adam est une méthode stochastique de descente de gradient qui est basée sur l'estimation adaptative des moments du premier et du second ordre.
- **La fonction de perte categorical_crossentropy** : Calcul de la perte d'entropie croisée entre les étiquettes et les prédictions, utilisée lorsqu'il existe deux ou plusieurs classes d'étiquettes.
- **La métrique accuracy** : Calcule la fréquence à laquelle les prédictions égalent les étiquettes, similaire aux fonctions de perte, sauf que les résultats de l'évaluation d'une métrique ne sont pas utilisés lors de la formation du modèle, nous pouvons utiliser n'importe quelle métrique.

La phase suivante qui consiste à l'entraînement du modèle est alimentée par plusieurs paramètres à prendre en compte :

1. Selon Kingma et al. 2014, la méthode est "efficace sur le plan du calcul, nécessite peu de mémoire, est invariable avec le redimensionnement diagonal des gradients et convient bien aux problèmes qui sont importants en termes de données/paramètres".

- **Nombre de couches cachées** : le nombre de couches entre la couche d'entrée et la couche de sortie.
- **Le taux de perturbation (Dropout)** : consiste à enlever certains nœuds afin que le réseau ne soit pas trop lourd. L'idée est que nous ne voulons pas que notre réseau soit submergé d'informations, surtout si nous considérons que certains nœuds peuvent être redondants et inutiles. Elle a pour but de forcer l'entraînement du réseau à s'entraîner, et éviter le surajustement des données (overfitting). Le taux est la fraction des unités d'entrée à ignorer.
- **Nombre d'étapes d'apprentissage** : représente le nombre de fois que notre algorithme s'entraîne sur l'ensemble de notre jeu de données.
- **Nombre d'itération par étape d'apprentissage** : le nombre de lots nécessaires pour terminer une époque.
- **Nombre d'étapes de validation** : similaire à **Nombre d'itération par étape d'apprentissage** mais sur l'ensemble des données de validation au lieu des données de formation.

8.3.1.1 Résultats de tests

Nous allons maintenant présenter les résultats de métriques d'évaluation lors de la phase de choix des paramètres du modèle MLP. La procédure de nos test obtenus dans le tableau de la [Figure 8.4] est de varier un paramètre en fixant les autres aux valeurs indiquées en vert qui sont nos meilleurs paramètres trouvés.

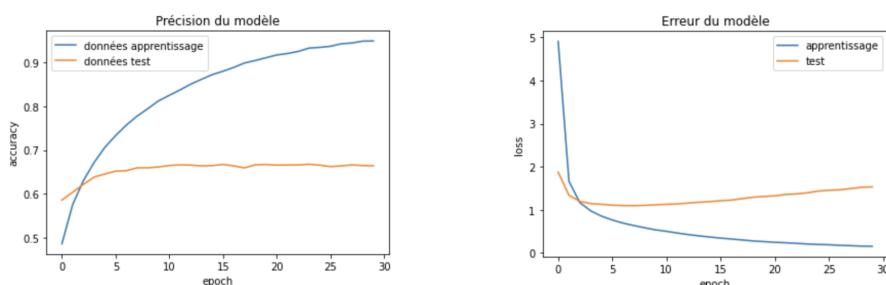
Dans la [Figure 8.4] nous voyons parfaitement l'effet de l'erreur de validation, elle est différente de l'erreur d'apprentissage, parfois elle conduit à ce qu'on appelle "le risque empirique", qui signifie la possibilité d'avoir un taux d'erreur d'apprentissage mais avec une erreur de validation considérable. Cette erreur consiste en l'erreur trouvée sur l'ensemble de test que notre réseau n'a pas utilisé dans l'apprentissage. Cette erreur est souvent plus considérée par rapport à l'erreur d'apprentissage qui quand à elle s'agit de l'erreur commise sur l'ensemble d'apprentissage. Nous avons alors détecté les meilleurs paramètres de notre modèle MLP qui nous donnent une meilleure précision du modèle avec une erreur d'apprentissage minimale en prenant en considération l'erreur de validation (paramètres en vert).

Les courbes de la [Figure 8.5] illustrent la variation de la précision du modèle ainsi que l'erreur durant les étapes d'apprentissage sur les ensembles d'entraînement et de tests. Dans la figure 8.5a nous voyons comment la précision sur les deux ensembles (entraînement et test) croient, ceci nous indique que notre modèle apprend durant chaque phase d'entraînement. Dans la figure 8.5b nous constatons le phénomène de minimisation du

Paramètre	Valeur	Précision d'entraînement	Taux d'erreur d'apprentissage	Taux d'erreur de validation
Nombre de couches cachées	1	0.94	0.15	1.53
	2	0.97	0.06	1.92
	3	0.98	0.04	2.56
Le taux de perturbation (Dropout)	0.1	0.97	0.08	1.75
	0.2	0.94	0.15	1.53
	0.5	0.83	0.45	1.13
Nombre d'étapes d'apprentissage	20	0.91	0.25	1.31
	30	0.94	0.15	1.53
	60	0.97	0.07	2.15
Nombre d'itérations par étape d'apprentissage	30	0.92	0.22	1.31
	60	0.94	0.15	1.53
	90	0.95	0.14	1.76
Nombre d'étapes de validation	1000	0.95	0.15	1.55
	2000	0.94	0.15	1.53

FIGURE 8.4 – Résultats d'évaluation.

coût de perte de notre modèle, l'erreur sur les deux ensembles (entraînement et test) décroissent à chaque évolution de la phase d'apprentissage, nous remarquons une légère croissance sur l'erreur de validation durant les dernières phases d'entraînement qui est due à la quantité de données de tests que nous avons présenté à notre modèle, mais qui n'influe pas la qualité de notre modèle.



(a) Précision du modèle. (b) Erreur du modèle.

FIGURE 8.5 – Courbes d'évaluations du MLP.

8.3.2 Évaluation du modèle KNN

Nous allons maintenant définir les hyper-paramètres de l'algorithme KNN qui nous permet d'obtenir les cas les plus similaires à une requête en entrée. Pour ce faire nous

allons d'abord présenter la signification de chaque paramètres listé dans la [Figure 8.3] à régler.

- **Nombre de voisins** : c'est le nombre de voisins à utiliser par défaut pour les requêtes sur les voisins.
- **Algorithme de calcul de voisins** : Algorithme utilisé pour calculer les voisins les plus proches, plusieurs options possibles :
 - **brute-force** : L'utilisation de la force brute implique le calcul des distances entre toutes les paires de points de données dans l'ensemble de données.
 - **kd-tree** : L'arbre KD est une structure de données arborescente qui aide à surmonter le coût de calcul de l'approche de la force brute. L'idée de base est que si le point A est très éloigné du point B, et que le point B est très proche du point C, alors nous savons que les points A et C sont très éloignés, sans avoir à calculer explicitement leur distance.
 - **ball-tree** : L'arbre à boules aide à surmonter les inefficacités de l'arbre KD pour les dimensions supérieures. Le ball tree est beaucoup plus rapide pour le stockage de données multidimensionnelles. L'arbre à boules divise les données en hypersphères imbriquées appelées boules. Dans l'arbre à boules, chaque noeud contient une hypersphère. Et chaque hypersphère contient un sous-ensemble de points qui doivent être recherchés. L'arbre à boules assume les données dans un espace multidimensionnel et crée les hypersphères emboîtées.
 - **auto** : tentera de décider de l'algorithme le plus approprié en se basant sur les valeurs passées à la méthode d'entraînement.
- **Fonction de pondération** : c'est la fonction de pondération utilisée dans la prédiction, deux cas de figures :
 - **uniform** : Tous les points de chaque voisinage sont pondérés de manière égale.
 - **distance** : Pondérer les points par l'inverse de leur distance. Dans ce cas, les voisins les plus proches d'un point de requête auront une plus grande influence que les voisins les plus éloignés.
- **Métrique de Distance** : la métrique de distance à utiliser pour l'arbre, que nous avons détaillé dans l'état de l'art.

Pour le choix entre ces paramètres nous avons utilisé un autre modèle appelé "Grid-SearchCV". C'est une méthode d'optimisation (hyperparameter optimization) qui va nous permettre de tester une série de paramètres et de comparer les performances pour en déduire le meilleur paramétrage. Il existe plusieurs manières de tester les paramètres d'un modèle et le Grid Search est une des méthodes les plus simples. Pour chaque paramètre, on détermine un ensemble de valeurs que l'on souhaite tester. Le Grid Search croise simplement chacune de ces hypothèses et va créer un modèle pour chaque combinaison de

paramètres. Dans notre cas nous aurons 80 modèles à tester, ceci nécessite une bonne puissance de la machine, d'où notre implémentation sur le cloud.

La méthode consiste à découper le data set en k échantillons. On sélectionne x échantillons pour constituer l'échantillon d'apprentissage. Les $k - x$ échantillons restants permettront d'évaluer la performance du modèle. Pour construire le modèle suivant on sélectionne les échantillons différemment de manière à ne jamais avoir les mêmes échantillons d'apprentissage et de validation. Une fois que chaque modèle a pu être entraîné et évalué, il ne reste plus qu'à comparer la performance pour choisir le meilleur modèle. Dans notre cas nous avons passé ces paramètres à ce modèle et il a retourné les meilleurs pour nous [Figure 8.6].

```
[73]: #Réglages des hyper-paramètres
params = {'n_neighbors': [3, 2, 1, 4, 5],
          'weights': ['distance', 'uniform'],
          'algorithm': ['kd_tree', 'ball_tree', 'auto', 'brute'],
          'metric': ['euclidean', 'manhattan']}
#Construcion d'un model qui cherchera les meilleurs hyperparametres
grid_search_cv = GridSearchCV(KNeighborsClassifier(), param_grid=params, n_jobs=-1, verbose=1)
gs_results = grid_search_cv.fit(x_tr, y_tr)

Fitting 5 folds for each of 80 candidates, totalling 400 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done 42 tasks      | elapsed:  1.3min
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  4.9min
[Parallel(n_jobs=-1)]: Done 400 out of 400 | elapsed:  8.6min finished

[76]: #Afficher les meilleurs paramètres
print("Les meilleurs hype-paramètres sont : \n", gs_results.best_params_)

Les meilleurs hype-paramètres sont :
{'algorithm': 'kd_tree', 'metric': 'manhattan', 'n_neighbors': 2, 'weights': 'distance'}
```

FIGURE 8.6 – Choix des hyper-paramètres avec GridSearch.

8.3.2.1 Résultats de Tests

Après avoir détecté les meilleurs paramètres nous les avons utilisé pour entraîner notre modèle KNN et nous avons tiré les performances finales. Nous avons listé le rapport de classification pour chacune des classes disposée dans notre ensemble d'apprentissage que nous résumons dans la [Figure 8.7].

Dans cette figure nous avons pour chaque classe contenant 1000 image chacune, la précision, le recall, et le f-score. Nous avons de plus le score ROC de l'aire sous la courbe de notre modèle en entier.

La matrice de confusion de notre modèle qui nous donne pour chaque classe les portions de prédictions par rapport aux autres classes possibles dans notre ensemble de donnée, est illustrée dans la [Figure 8.8]

La matrice de confusion dans sa diagonale nous montre combien de fois la classe i a été

Rapport de classification :				
	precision	recall	f1-score	support
Avion	0.64	0.65	0.65	1000
Voiture	0.61	0.67	0.64	1000
Oiseau	0.47	0.47	0.47	1000
Chat	0.43	0.34	0.38	1000
Cerf	0.46	0.48	0.47	1000
Chien	0.49	0.49	0.49	1000
Grenouille	0.55	0.62	0.58	1000
Cheval	0.57	0.57	0.57	1000
Bateau	0.66	0.57	0.61	1000
Camion	0.56	0.59	0.57	1000
micro avg	0.54	0.54	0.54	10000
macro avg	0.54	0.54	0.54	10000
weighted avg	0.54	0.54	0.54	10000
samples avg	0.54	0.54	0.54	10000

```
# Score Aire sous la courbe (ROC_AUC)
print('Score ROC AUC = ',roc_auc_score(y_test,knn_prediction))
```

Score ROC AUC = 0.7469444444444444

FIGURE 8.7 – Rapport de classification KNN.

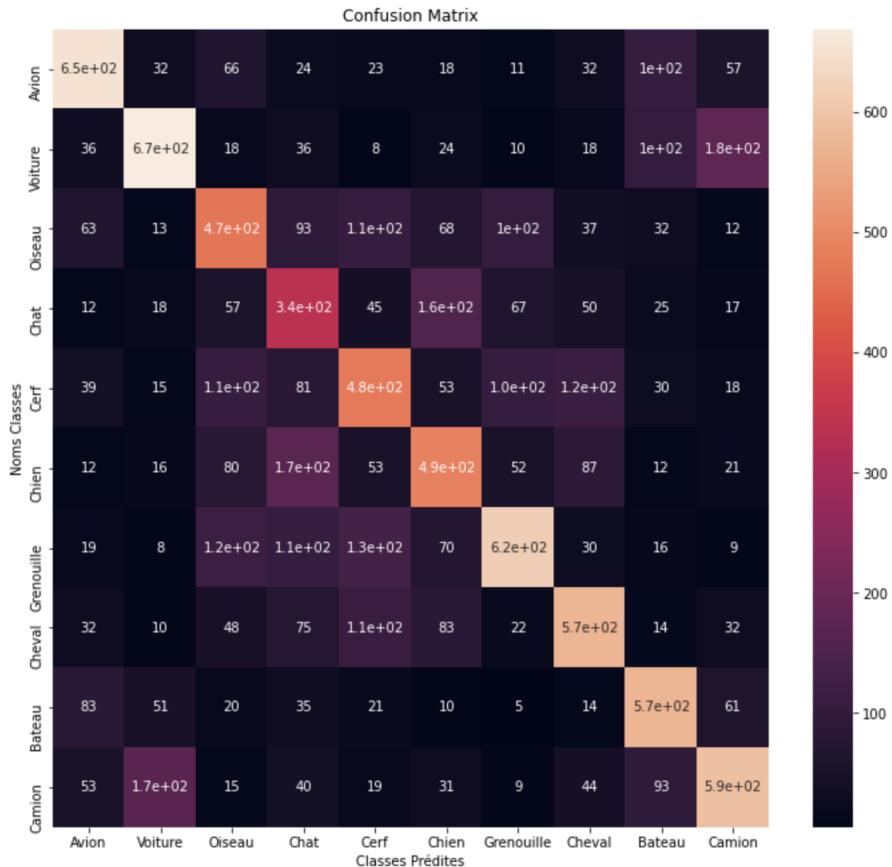


FIGURE 8.8 – Matrice de confusion du KNN.

prédite comme elle même, l'axe des abscisses contient les étiquettes des classes prédites, l'axe des ordonnées contient les étiquettes des classes réelles. Nous pouvons voir certaines confusion que notre modèle a pu commettre, si on prend l'exemple de l'ordonnée "camion" nous pouvons voir que 170 fois le modèle a pu trompé le camion avec "voiture", et ceci car

ces deux exemples contiennent des caractéristiques en commun, par contre il l'a trompé uniquement 9 fois avec une "grenouille".

A ce stade nous avons toutes les mesures qui nous montrent que nos deux modèles MLP et KNN sont assez précis, ce qui nous permettra d'avoir une certaine homogénéité entre ces deux modèles puisqu'ils utilisent les mêmes caractéristiques dans leurs phases d'apprentissage.

8.4 Résultat du couple CNN-KNN

Après l'implémentation de notre solution "Explication locale post-hoc spécifique au modèle" des résultats de réseaux de neurones convolutifs, nous avons tester notre solution sur des images téléchargées sur le net, qui ne figurent pas dans notre dataset Cifar-10. Nous avons testé notre solution sur deux cas de figures pour vérifier l'homogénéité entre nos deux modèles jumeaux :

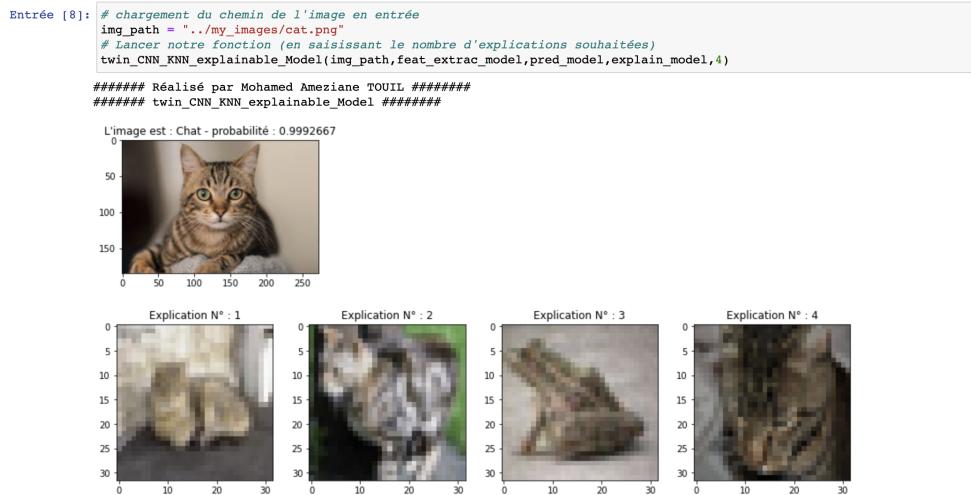
1. Cas d'une prédiction correcte : c'est le cas où notre modèle de prédiction ne s'est pas trompé dans sa prédiction, le modèle d'explication est censé nous donner une explication sur cette prédiction correcte.
2. Cas d'une prédiction fausse : c'est le cas où notre modèle de prédiction se trompe dans sa prédiction, le modèle d'explication est censé de même nous donner une explication sur cette fausse prédiction.

8.4.1 Cas de prédiction correcte

Les [Figures 8.9a & 8.9b] nous illustrent deux exemples de bonnes prédiction avec les explications visuelle qui lui sont associées.



(a) Bonne prédiction de voiture.



(b) Bonne prédiction de chat.

FIGURE 8.9 – Explication d'une prédiction correcte.

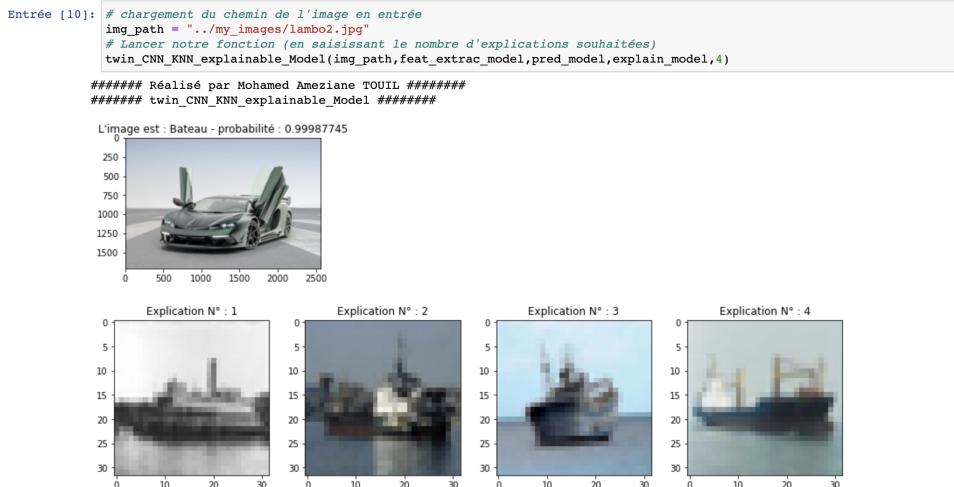
Nous pouvons voir dans ces deux exemples comment le modèle d'explication nous a illustré les explications de nos prédictions. Dans l'exemple de voiture nous pouvons conclure visuellement les ressemblances entre notre requête et nos explications : couleur de la Tôle est bleue, l'arrière plan contient des arbres, la route est grise, la taille du véhicule, ...etc.

8.4.2 Cas de prédiction fausse

Les [Figures 8.10a & 8.10b] nous illustrent deux exemples de mauvaise prédiction avec les explications visuelle qui lui sont associées.



(a) Fausse prédiction de voiture en camion.



(b) Fausse prédiction de voiture en bateau.

FIGURE 8.10 – Explication d'une prédiction correcte.

Nous pouvons voir dans ces deux exemples comment le modèle d'explication nous a illustré les explications de nos prédictions fausses. Dans l'exemple de voiture grise avec portes écartées prédite comme bateau nous pouvons conclure visuellement les ressemblances entre notre requête et nos explications : couleur grise comme celle d'un bateau, l'arrière plan du ciel bleu, la route est grise ressemble à un océan, la hauteur des portes est comme les axes verticaux d'un bateau, ...etc.

8.5 Conclusion

A travers ce chapitre, nous avons évalué nos modèles décrit dans la conception. Cette évaluation nous a permis de tester diverses métriques de performances (la précision, F-score, Rappel, AUC, ...), pour créer un modèle homogène et robuste. Par conséquent, Les résultats que nous avons obtenu confirment nos hypothèses concernant l'utilisation des caractéristiques extraites de nos images pour entraîner nos deux modèles, de prédiction et d'explication respectivement, pour avoir des explications sur les prédictions d'un réseau de neurones.

Conclusion et perspectives

Les méthodes d'apprentissage automatique se développent de plus en plus, parmi elles les réseaux de neurones artificiels, qui sont une technique très utilisée dans les problèmes de prédiction. Cette technique répond à la majorité de ces problèmes, mais sans nous permettre d'avoir une explication et interprétation des résultats. Par conséquent, le domaine de l'XAI (eXplainable Artificial Intelligence) vise à traiter cette contrainte.

Dans ce contexte, les travaux de recherche visent à trouver des solutions interprétables et compréhensibles par l'utilisateur. L'objectif en commun de ces solutions consiste à présenter un modèle d'apprentissage automatique accompagné d'une explication de sa structure et son fonctionnement, ou de ses prédictions.

Après une recherche bibliographique sur les travaux existants dans la thématique d'explication des résultats de RNA, nous avons établis une étude comparatives de ses méthodes, avant de proposer et concevoir notre solution. Ces techniques d'interprétation sont souvent divisées en deux grandes classes, et chacune de ses deux classes peut être sous deux formes différentes. La première classe s'agit d'une intégrabilité Intrinsèque, qui signifie qu'un seul modèle peut s'auto-expliquer, contrairement à l'intégrabilité post-hoc qui nécessite un deuxième modèle expliquant le modèle de prédiction.

En se basant sur les deux grandes classes d'interprétation, nous distinguons deux formes d'intégrabilité. L'intégrabilité globale, nous aide à comprendre le fonctionnement global du modèle en se basant sur une inspection de sa structure et ses paramètres, tandis que l'intégrabilité locale explique une prédiction individuelle du modèle, localement, dans le but de comprendre la prise de décision de ce modèle.

Dans ce projet de fin d'études, nous nous sommes intéressés aux réseaux de neurones convolutifs qui sont utilisés de plus en plus dans l'apprentissage automatique et la vision par ordinateur. Nous avons traité la contrainte d'intégrabilité des RNA, en proposant un système composé de deux modèles homogènes, un CNN qui extrait une carte de caractéristiques utilisée pour effectuer une prédiction, et qui sera aussi passée au deuxième modèle KNN qui, à la base de cette carte, il donne une explication visuelle de la prédiction sous forme du cas le plus similaire à la donnée passée en entrée.

Cette technique est simple à mettre en place à l'aide du transfert learning, qui consiste en l'utilisation d'un réseau de neurones déjà entraîné pour résoudre un problème dans le but d'en résoudre un autre similaire au premier. Nous avons utilisé le réseau convolutif VGG-16 pour la partie d'extraction de caractéristiques qui contribuent le plus à la prédiction, et seront utilisées pour l'interprétabilité.

Cette solution est post-hoc, nous avons utilisé un autre modèle (un KNN) pour expliquer notre modèle de prédiction (un CNN). Elle est interprétable locale, vise à expliquer une prédiction individuelle à base d'une carte de caractéristiques de la requête à prédire. Elle est spécifique au modèle, consiste à expliquer un type de RNA (CNN) dans la reconnaissance d'objets de ImageNet.

L'implémentation et le test de la solution nous montrent de bons résultats. Nous arrivons à comprendre visuellement (donc nécessite la présence d'un superviseur) la prédiction obtenue par le RNA, et ceci en voyant les ressemblances entre l'image prédite et ses k-plus proches voisins. Les tests effectués montrent l'homogénéité entre le modèle de prédiction et le modèle d'explication, c-a-d ce dernier explique les bonnes et les mauvaises prédictions du premier.

Pour renforcer nos modèles et préciser encore plus l'interprétabilité nous avons élaboré les perspectives suivantes :

- Utilisation des détecteurs de points d'intérêts et les descripteurs (tels que SIFT, SURF, FAST, BRIEF ... etc) , entre notre image en requête et les explications données par le KNN dans le but de voir les parties des images en communs. Cette idée nous servira à mieux interpréter nos résultats, car avec les correspondances entre les descripteurs des explications et celui de l'image de prédiction nous allons voir les parties en communs entre ces images, nous saurons alors quelles sont ces parties qui participent le plus à la prédiction.
- Une autre expérimentation que nous pouvons faire, c'est de trouver une relation mathématique (un ratio par exemple) entre notre image de prédiction et les explications renvoyées par le KNN pour mieux choisir ces dernières pour ne garder que les meilleures correspondances et éviter les mauvaises.
- Nous pouvons aussi tester d'autres modèles les explications (tels que K-Means, Random-Forests, SVM) , et tirer le meilleur modèle qui donne une meilleure précision et aussi qui s'approche de mieux avec celle du modèle de prédiction.

Bibliographie

- [1] Jean-Paul Haton, Marie-Christine Haton **L'Intelligence Artificielle**, Edition publiée par Presses Universitaires de France - PUF, 1993.
- [2] Minsky, M. , **The emotion machine : Commonsense thinking, artificial intelligence, and the future of the human mind.** , Edition publiée par SIMON & SCHUSTER, 2007.
- [3] A. Cornuéjols, L. Miclet, and Y.Kodratoff, **Apprentissage Artificiel, Concepts et algorithmes.** , Edition publiée par EYROLLES, 2002.
- [4] Tom M. Mitchell **Machine Learning**, Edition publiée par McGraw-Hill Science/Engineering/Math, 1997.
- [5] SUPINFO. **Machine Learning : Introduction à l'apprentissage automatique**, 2017. <https://www.supinfo.com/articles/single/6041-machine-learning-introduction>.
- [6] Kevin Gurney, University of McMaster **Neural Networks and Learning Machines**, Edition published by Pearson Education, Library of Congress Cataloging-in-Publication Data, 2009.
- [7] Simon Haykin , University of Sheffield **An introduction to neural networks**, Edition published in the Taylor Francis e-Library, 2004.
- [8] David Kriesel, University of Bonn **A Brief Introduction to Neural Networks**, Edition published online under www.dkriesel.com on 5/27/2005.
- [9] Claude Touzet, University of Sheffield **Les réseaux de neurones artificiels, Introduction au connexionisme**, 1992.
- [10] Ivan Nunes da Silva, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, Silas Franco dos Reis Alves , University of São Paulo **Artificial Neural Networks, Practical course**, 2017.
- [11] Jagannathan Sarangapani,University of Missouri **Neural Network Control of Nonlinear Discrete-Time Systems**Publiée par CRC Press (Taylor Francis Group), 2006.
- [12] G. Dreyfus, J.-M. Martinez, M. Samuelides, M. B. Gordon, F. Badran, S. Thiria, **Apprentissage statistique**Publiée par EYROLLES), 2002.
- [13] Mengnan Du, Ninghao Liu, Xia Hu. **Techniques for Interpretable Machine Learning**. arXiv 2018, arXiv :1808.00033.

-
- [14] Montavon, G. ; Lapuschkin, S. ; Binder, A. ; Samek, W. ; Müller, K.R. **Explaining nonlinear classification decisions with deep Taylor decomposition.** Pattern Recognit. 2017, 65, 211–222.
 - [15] Golovin, D. ; Solnik, B. ; Moitra, S. ; Kochanski, G. ; Karro, J. ; Sculley, D. Google vizier. **A service for black-box optimization.** In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, 13–17 August 2017; pp. 1487–1495.
 - [16] Rudin, C. **Please Stop Explaining Black Box Models for High Stakes Decisions.** arXiv 2018, arXiv :1811.10154.
 - [17] F. Doshi-Velez and B. Kim. **Towards a rigorous science of interpretable machine learning.** 2017.
 - [18] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, D. Pedreschi, and F. Giannotti. **A survey of methods for explaining black box models.** arXiv preprint arXiv :1802.01933, 2018.
 - [19] Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael Specter and Lalana Kagal. **Explaining Explanations : An Overview of Interpretability of Machine Learning.** arXiv preprint arXiv :1806.00069, 2019.
 - [20] STATISTICA. **Concepts Fondamentaux en Statistique et Data Mining**, 2016. <http://www.statsoft.fr/concepts-statistiques/index/concepts-index.html>.
 - [21] A. A. Freitas. **Comprehensible classification models : a position paper.** ACM SIGKDD explorations newsletter, 2014.
 - [22] K. Simonyan, A. Vedaldi, and A. Zisserman. **Deep inside convolutional networks : Visualising image classification models and saliency maps .** ICLR Workshop, 2014
 - [23] Riccardo Guidotti, Anna Monreale, Stan Matwin, and Dino Pedreschi. **Black Box Explanation by Learning Image Exemplars in the Latent Feature Space.** ECMLPKDD Accepted Papers :572, 2019.
 - [24] Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin. **“Why Should I Trust You ?” Explaining the Predictions of Any Classifier** arXiv preprint arXiv :1602.04938, 9 Aug 2016.
 - [25] Marco Tulio Ribeiro. **LIME - Local Intepretable Model-Agnostic Explanations.** Dans UW Wachington Edu, 2 April 2016. URL : <https://homes.cs.washington.edu/~marcotcr/blog/lime/>.
 - [26] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. **Local Interpretable Model-Agnostic Explanations (LIME) : An Introduction. A technique to explain the predictions of any machine learning classifier.** Dans Oreilly, oreilly/learning, 12 August 2016. URL :

<https://www.oreilly.com/learning/introduction-to-local-interpretable-model-agnostic-explanations-lime>.

- [27] Christoph Molnar. **Interpretable Machine Learning, A Guide for Making Black Box Models Explainable.** Dans le Repository Github "christophm", Dernière mise à jour le 17/12/2019. URL : <https://christophm.github.io/interpretable-ml-book/>.
- [28] Kary FRÄMLING. **Explaining Results of Neural Networks by Contextual Importance and Utility.** Researchgate, Janvier 1996.
- [29] Sule Anjomshoae,Kary Främling, et Amro Najjar. **Explanations of Black-Box Model Predictions by Contextual Importance and Utility.** Researchgate, Mai 2019.
- [30] Miller Tim. **Explanation in artificial intelligence : insights from the social sciences.** Researchgate, Juin 2017.
- [31] K. Simonyan, A. Vedaldi, et A. Zisserman. **Deep inside convolutional networks : Visualising image classification models and saliency maps.** Researchgate, décembre 2013.
- [32] M. D. Zeiler et R. Fergus. **Visualizing and understanding convolutional networks.** Springer, 2014, pages 818–833.
- [33] S. Bach, A. Binder, G. Montavon, F. Klauschen, K. R. Müller, et W. Samek. **On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation.** PLoS ONE 10.7, 2015.
- [34] A. Shrikumar, P. Greenside, et A. Kundaje. **Learning Important Features Through Propagating Activation Differences.** arXiv : 1704.02685, Juin 2017.
- [35] M. Sundararajan, A. Taly, et Q. Yan. **Axiomatic attribution for deep networks.** arXiv : 1703.01365, 2017.
- [36] G. Montavon, W. Samek, et K.-R. Müller. **Methods for interpreting and understanding deep neural networks.** Researchgate, Février 2018.
- [37] O. Li, H. Liu, C. Chen, and C. Rudin. **Deep Learning for Case-Based Reasoning through Prototypes : A Neural Network that Explains Its Predictions.** arXiv : 1710.04806, 2018.
- [38] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, et R. Sayres.. **Interpretability Beyond Feature Attribution : Quantitative Testing with Concept Activation Vectors (TCAV) .** Dans International Conference on Machine Learning (ICML). 2018.
- [39] T. Lei, R. Barzilay, et T. Jaakkola. **Rationalizing Neural Predictions.** arXiv : 1606.04155, 2017.

-
- [40] Osbert Bastani, Carolyn Kim, et Hamsa Bastani. **Interpreting Blackbox Models via Model Extraction.** arXiv : 1705.08504, 24 Janvier 2019.
- [41] Rajiv Khanna, Been Kim, Joydeep Ghosh, et Oluwasanmi Koyejo. **Interpreting Black Box Predictions using Fisher Kernels.** arXiv : 1810.10118, 23 Octobre 2018.
- [42] David Alvarez-Melis et Tommi S. Jaakkola. **Towards Robust Interpretability with Self-Explaining Neural Networks.** Researchgate, juin 2018.
- [43] Paolo Tamagnini, Josua Krause, Aritra Dasgupta, et Enrico Bertini. **Interpreting Black-Box Classifiers Using Instance-Level Visual Explanations.** Researchgate, Mai 2017.
- [44] Tom Ralf Herbrichl **Learning Kernel Classifiers Theory and Algorithms**, Edition publiée par Library of Congress Cataloging-in-Publication Data, 2001.
- [45] Zhang, Yu-Dong, et al. "Magnetic resonance brain image classification based on weighted-type fractional Fourier transform and nonparallel support vector machine." International Journal of Imaging Systems and Technology 25.4 (2015) : 317-327.
- [46] Srinivas, B., and G. Sasibhushana Rao. "A Hybrid CNN-KNN model for MRI brain tumor Classification." 2019.
- [47] Olden, Julian D., and Donald A. Jackson. "Illuminating the “black box” : a randomization approach for understanding variable contributions in artificial neural networks." Ecological modelling 154.1-2 (2002) : 135-150.
- [48] Selvaraju, Ramprasaath R., et al. "Grad-cam : Visual explanations from deep networks via gradient-based localization." Proceedings of the IEEE international conference on computer vision. 2017.
- [49] Kenny, Eoin M., and Mark T. Keane. "Twin-systems to explain artificial neural networks using case-based reasoning : comparative tests of feature-weighting methods in ANN-CBR twins for XAI." Twenty-Eighth International Joint Conferences on Artificial Intelligence (IJCAI), Macao, 10-16 August 2019. 2019.
- [50] Ramon Lopez De Mantaras, David McSherry, Derek Bridge, David Leake, Barry Smyth, Susan Craw, Boi Faltings, Mary Lou Maher, Michael T Cox, Kenneth Forbus, et al. "Retrieval, reuse, revision and retention in case-based reasoning." The Knowledge Engineering Review, 20(3) :215–240, 2005.
- [51] Bench-Capon, Trevor JM. "Hypo’s legacy : introduction to the virtual special issue." Artificial Intelligence and Law 25.2 (2017) : 205-250.
- [52] Cunningham, Pádraig, Dónal Doyle, and John Loughrey. "An evaluation of the usefulness of case-based explanation." International Conference on Case-Based Reasoning. Springer, Berlin, Heidelberg, 2003.

-
- [53] Suguna, N., and K. Thanushkodi.. "**An improved k-nearest neighbor classification using genetic algorithm.**" International Journal of Computer Science Issues 7.2 (2010) : 18-21.
- [54] Keane, Mark T., and Eoin M. Kenny. "**How case-based reasoning explains neural networks : A theoretical analysis of XAI using post-hoc explanation-by-example from a survey of ANN-CBR twin-systems.**" International Conference on Case-Based Reasoning. Springer, Cham, 2019.
- [55] Jason Brownlee. "**A Gentle Introduction to Transfer Learning for Deep Learning**", 2017. <https://machinelearningmastery.com/transfer-learning-for-deep-learning/>.
- [56] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "**Deep learning.**" MIT press, 2016.
- [57] Pascal Monasse et Kimia Nadjahi. "**Classez et segmentez des données visuelles**", 2020. <https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles>.
- [58] Dipanjan Sarkar. "**A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning**", 2018. Url : <https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a>.
- [59] Aditya Ananthram. "**Deep Learning For Beginners Using Transfer Learning In Keras**", 2018. <https://towardsdatascience.com/keras-transfer-learning-for-beginners-6c9b8b7143e>.
- [60] Simonyan, Karen, and Andrew Zisserman. "**Very deep convolutional networks for large-scale image recognition**". arXiv preprint arXiv:1409.1556 (2014).
- [61] Tavish Srivastava. "**Introduction to k nearest neighbors**". Url : <https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/>. (2018).
- [62] Perner, Petra, ed. "**Case-based reasoning on images and signals**". Vol. 73. Springer, 2008.
- [63] Smith, Linda B. "**From global similarities to kinds of similarities : The construction of dimensions in development**". (1989).
- [64] Srivastava, Nitish, et al. "**Dropout : a simple way to prevent neural networks from overfitting**". JMLR 2014
- [65] Sparsh Gupta. "**Most Common Loss Functions in Machine Learning**", 2020. Url : <https://towardsdatascience.com/most-common-loss-functions-in-machine-learning-c7212a99dae0>.

-
- [66] Somnath Paul. "**Different Optimization Algorithm for Deep Neural Networks : Complete Guide**", 2020. Url : <https://medium.com/Analyticsvidhya/different-optimization-algorithm-for-deep-neural-networks-complete-guide-7f3e49eb7d42>.
- [67] P.J. Werbos, "**Backpropagation through time : What it does and how to do it ?**". Proc. IEEE, vol. 78, pp. 1550-1560, 1990
- [68] M. Minoux, "**Mathematical Programming : Theory and Algorithms**". Wiley and Sons, 1984.
- [69] Nagesh Singh Chauhan. "**Optimization Algorithms in Neural Networks**", 2020. Url : <https://www.theaidream.com/post/optimization-algorithms-in-neural-networks>
- [70] Algorithmia. "**Introduction to Optimizers**", 2018. Url : <https://algorithmia.com/blog/introduction-to-optimizers>
- [71] Duchi, J., Hazan, E., Singer, Y. "**Adaptive Subgradient Methods for Online Learning and Stochastic Optimization**. *Journal of Machine Learning Research*", 2011. 12, 2121–2159.
- [72] Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., ... Ng, A. Y. . "**Large Scale Distributed Deep Networks**", 2012. NIPS 2012 : Neural Information Processing Systems, 1–11.
- [73] Pennington, J., Socher, R., Manning, C. D. "**Glove : Global Vectors for Word Representation**", 2014. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 1532–1543.
- [74] Zeiler, M. D. "**ADADELTA : An Adaptive Learning Rate Method**", 2012.
- [75] Geoff Hinton. "**Introduction to Optimizers**", 2018. Url : <https://algorithmia.com/blog/introduction-to-optimizers>
- [76] Geoff Hinton. "**Lecture 6a : Overview of mini-batch gradient descent**", x . 6e conférence de sa classe de Coursera : Neural Networks for Machine Learning.
- [77] Kingma, D. P., Ba, J. L. "**Adam : a Method for Stochastic Optimization**", 2015. International Conference on Learning Representations, 1–13.
- [78] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S. "**GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium**", 2017. In Advances in Neural Information Processing Systems 30 (NIPS 2017).
- [79] Zhang, A., Lipton, Z. C., Li, M., Smola, A. J . "**Dive into deep learning**", 2019. Unpublished Draft. Retrieved, 2019, vol. 19, p. 2019.

Liste des tableaux

5.1 Résumé des méthodes d'explication des boîtes noires.	71
6.1 Structure de la base de Cas.	92

Table des figures

2.1	Types d'apprentissage automatique	14
2.2	Apprentissage supervisé.	14
2.3	Apprentissage non supervisé.	15
2.4	Apprentissage par renforcement.	16
3.1	Structure d'un neurone artificiel.	19
3.2	Représentation d'un neurone artificiel.	19
3.3	Différentes formes de la fonction d'activation pour le neurone artificiel. . .	21
3.4	Couches d'un réseau de neurones artificiels.	22
3.5	Architecture d'un réseau monocouche.	23
3.6	Architecture d'un réseau multicouche.	24
3.7	Architecture d'un réseau bouclé ou récurrent.	25
3.8	Architecture d'un réseau maillé.	26
3.9	Architecture d'un réseau à connexion complète.	26
3.10	Architecture d'un réseau à connexion locale.	27
3.11	Tableau d'une matrice RVB.	28
3.12	Réseau de neurones avec de nombreuses couches convolutives.	28
3.13	Processus des couches de convolution.	29
3.14	Réseau convolutionnel avec filtre 3x3, 1 pas et entrée de padding de zéro. .	30
3.15	Max Pooling.	31
3.16	La couche de pooling est aplatie en une couche FC.	31
3.17	Exemple 1 de classification par un KNN.	32
3.18	Exemple 1 de classification par un KNN.	33
3.19	Contribution de la valeur de "k" dans la classification dans un KNN. . . .	33
3.20	Courbe du taux d'erreur d'entraînement avec une valeur variable de "K". .	34
3.21	Courbe d'erreur de validation avec une valeur variable de "K".	34
4.1	Différentes formes de la fonction d'erreur pour la classification et la régression.	40
4.2	Algorithme de la descente du gradient.	42
4.3	Influence du taux d'apprentissage dans la descente du gradient.	43
4.4	Graphique de calcul de la propagation avant.	46
5.1	Modèle paramétrique et modèle non-paramétrique.	50

5.2	Illustration de trois lignes de techniques d'apprentissage automatique interprétables : Explication intrinsèque, explication globale post-hoc d'un modèle et explication locale post-hoc d'une prédiction.	54
5.3	A gauche : Architecture d'auto-encodeur contradictoire : l'encodeur (encoder) transforme l'image x en sa représentation latente z , le décodeur (decoder) reconstruit une approximation \tilde{x} de x à partir de z , et le discriminateur identifie si une instance latente h générée aléatoirement peut être considérée comme valide ou non. A droite : Module discriminateur et décodeur (discr.) : l'entrée est une instance latente h générée aléatoirement et, si elle est considérée comme valide par le discriminateur, celui-ci la renvoie avec sa version décompressée \tilde{h}	57
5.4	Extracteur de règles locales latentes. Il prend en entrée l'image x à expliquer et la boîte noire b (black box). Avec l'encodeur (encoder) formé par l'AAE, il transforme x en sa représentation latente z . Ensuite, le module (neighgen) utilise z et b pour générer le voisinage local latent H . Les instances valides sont décodées dans \tilde{H} par le module (disde). Les images dans \tilde{H} sont étiquetées avec la boîte noire $Y = b(\tilde{H})$. H et Y sont utilisés pour apprendre un classificateur d'arbre de décision. Enfin, une règle de décision r et les règles contre-factuelles Θ pour z sont retournées.	58
5.5	Processus de LIME.	58
5.6	Transformation d'une image en composantes interprétables.	60
5.7	Explication d'une prédiction avec LIME.	61
5.8	Illustration des notions d'importance contextuelle et d'utilité contextuelle à partir de la problématique du choix d'une voiture	63
5.9	Fournir des explications pour les cas individuels utilisant IC et UC.	64
6.1	Architecture du système jumelé RNA-RàPC : Une requête à un RNA produit une prédiction, mais inexpliquée, de la classe d'une image. Le RNA est jumelé avec un système RàPC, ce qui permet à ce dernier de retrouver le cas du voisin le plus proche, en utilisant les poids des caractéristiques du RNA, pour expliquer la prédiction.	74
6.2	Module d'acquisition des données en entrée.	75
6.3	Format de la requête.	75
6.4	CIFAR-10 dataset.	76
6.5	Module d'extraction de caractéristiques.	76
6.6	Visualisation des couches 1 et 2.	77
6.7	Visualisation des couches 3,4 et 5.	78
6.8	Technique d'extraction de caractéristiques.	79
6.9	Architecture du VGG16.	80

6.10	Architecture du VGG16, sur Keras.	81
6.11	Le graphe représentant les couches de convolution du VGG16.	82
6.12	La forme des filtres de chaque bloc de couches de convolutions.	82
6.13	Les six premiers filtres sur les 64 filtres de la première et dernière couches du modèle VGG16.	83
6.14	L'image en entrée du VGG-16.	84
6.15	Forme des cartes de caractéristiques du VGG16.	84
6.16	Cartes de caractéristiques de la voiture dans la première couche du VGG-16.	85
6.17	Visualisation des 64 premières cartes des caractéristiques extraites dans les cinq blocs de convolutions du VGG-16.	86
6.18	Les caractéristique des images de cifar10 apprises dans les blocs de convolution du VGG-16.	87
6.19	Module de classification.	89
6.20	Architecture du MLP.	89
6.21	Le graphe représentant les couche du MLP.	89
6.22	Module RàPC.	91
6.23	Forme de la base de cas.	91
6.24	Représentation d'un cas basée sur les caractéristiques.	93
7.1	Python.	95
7.2	TensorFlow.	96
7.3	Keras.	96
7.4	Scikit-learn.	97
7.5	Joblib.	97
7.6	Matplotlib.	97
7.7	NumPy.	98
7.8	Pandas.	98
7.9	Google Cloud Plateform.	98
7.10	AI Platform Notebooks	99
7.11	Architecture technique.	100
8.1	Matrice de confusion.	104
8.2	Aire sous la courbe AUC.	105
8.3	Paramètres d'évaluation.	106
8.4	Résultats d'évaluation.	108
8.5	Courbes d'évaluations du MLP.	108
8.6	Choix des hyper-paramètres avec GridSearch.	110
8.7	Rapport de classification KNN.	111
8.8	Matrice de confusion du KNN.	111
8.9	Explication d'une prédiction correcte.	113

