République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي

ECOLE NATIONALE SUPÉRIEURE D'INFORMATIQUE

المدرسة الوطنية العليا للإعلام الآلي
(المعهد الوطني للتكوين في الإعلام الآلي سابقا)
École nationale Supérieure d'Informatique
ex. INI (Institut National de formation en Informatique)

Mémoire de fin d'études

pour l'obtention du diplôme d'Ingénieur d'État en Informatique

Option : Systèmes Informatiques (SQ)

Optimisation des architectures en deep learning en utilisant les techniques de plongement de graphes

Réalisé par :
M. Kacemi Souhib
M. Taibi Mohamed Kamel Eddine

Encadré par :
M. Ait Ali Yahia Yacine (ESI)
Mme. Amrouche Karima (ESI)

Soutenu le 20 Septembre 2023, Devant le jury composé de :

Mme. Fatima Benbouzid-Si Tayeb :	ESI	- Président
M. Hachemi Nabil Dellys :	ESI	- Rapporteur
Mme. Sara Ghorab :	ESI	- Examinateur

Promotion : 2022/2023

Dédicace

“

À mes chers parents, 2035

À mes chers frères et soeurs et leurs enfants,

À tous mes amis,

À mes professeurs,

Merci.

”

- Souhib

Remerciements

Nous tenons à exprimer nos profondes gratitudes pour tous ceux qui ont contribué à la réalisation de ce mémoire de master.

Nos sincères remerciements vont à nos directeurs de mémoire qui nous ont offert un encadrement de qualité, un soutien inestimable et des conseils judicieux tout au long de ce parcours académique.

Nous tenons également à remercier tous les enseignants qui nous ont prodigué leur savoir et leurs compétences pour que nous puissions réussir dans nos études.

Nous exprimons également nos profonds remerciements à **M. Ali TFAILY** et **M.Souhib KACEMI** pour leur support et leurs orientations.

Nos remerciements s'adressent également à nos familles et nos amis pour leur soutien, leur encouragement et leur amour indéfectibles.

Enfin, nous sommes reconnaissants envers toutes les personnes qui ont contribué, de près ou de loin, à l'aboutissement de ce travail de recherche. Merci infiniment pour votre précieuse aide et vos précieux conseils tout au long de cette aventure.

Résumé

La maintenance prédictive est cruciale aujourd'hui pour anticiper les pannes avant qu'elles ne surviennent, permettant ainsi de réduire les coûts et le temps de maintenance. Les moteurs électriques, largement utilisés dans l'industrie, les transports et les usines, sont sujets à de nombreux défauts électriques et mécaniques, rendant leur maintenance coûteuse mais indispensable.

Ce mémoire explore l'application du machine learning pour la maintenance prédictive des moteurs électriques. La qualité des données est un élément clé du machine learning. Pour enrichir notre jeu de données sur différents moteurs présents sur le marché, nous utilisons l'intelligence artificielle générative pour créer des données similaires aux données réelles, sous forme de séries temporelles. Des techniques avancées telles que les auto-encodeurs variationnels (VAE), les réseaux adverses génératifs (GAN) et les grands modèles de langage (LLM) sont employées pour générer ces données.

Le travail réalisé a permis de produire des séries temporelles très proches des données réelles, lesquelles sont utilisées par un modèle de classification pour prédire de manière fiable si un moteur va tomber en panne ou non. Les résultats obtenus démontrent que notre solution est capable de générer des données de haute qualité et de prédire efficacement les pannes des moteurs électriques, offrant ainsi une approche prometteuse pour la maintenance prédictive dans divers secteurs industriels.

Mots clés : Apprentissage profond, Apprentissage automatique, Réseaux neuronaux profonds, Maintenance prédictive, Séries temporelles, Réseaux adverses génératifs (GAN), Auto-encodeurs variationnels (VAE), Grands modèles de langage (LLM)

Abstract

Predictive maintenance is crucial today for anticipating failures before they occur, thereby reducing costs and maintenance time. Electric motors, widely used in industry, transportation, and factories, are subject to numerous electrical and mechanical faults, making their maintenance costly but indispensable.

This thesis explores the application of machine learning for the predictive maintenance of electric motors. Data quality is a key element of machine learning. To enrich our dataset on various motors available on the market, we use generative artificial intelligence to create data similar to real data, in the form of time series. Advanced techniques such as variational autoencoders (VAE), generative adversarial networks (GAN), and large language models (LLM) are employed to generate these data.

The work carried out has produced time series very close to real data, which are used by a classification model to reliably predict whether a motor will fail or not. The results obtained demonstrate that our solution is capable of generating high-quality data and effectively predicting electric motor failures, thus offering a promising approach for predictive maintenance in various industrial sectors.

Keywords : Deep learning, Machine learning, Deep neural networks, Predictive maintenance, Time series, Generative adversarial networks (GAN), Variational autoencoders (VAE), Large language models (LLM)

Table des matières

Table des figures

Liste des tableaux

Liste des sigles et acronymes

AI	*Artificial Intelligence*
GenAI	*Generative Artificial Intelligence*
ReLU	*Rectified Linear Unit*
ANN	Artificial Neural Network
DL	*Deep Learning*
ML	*Machine Learning*
GAN	*Generative Adversarial Networks*
LLM	*Large language models*
VAE	*Variational Autoencoders*
CNN	*Convolutional Neural Networks*
RNN	*Recurrent Neural Network*
LSTM	*Long Short-Term emory*
MDP	*Markov Decision process*
NLP	*Natural language Processing*
GD	*Gradient Descent*
MLP	*MultiLayer Perceptron*

Introduction générale

L'apprentissage profond, ou deep learning, est une branche de l'intelligence artificielle (IA) qui a transformé de nombreux secteurs et industries, en particulier ces dernières années. Grâce à ses capacités avancées, l'apprentissage profond a permis des avancées significatives dans des domaines tels que la reconnaissance d'image, la compréhension du langage naturel et la génération de données. En tant que composant essentiel de l'intelligence artificielle générative, l'apprentissage profond est utilisé pour créer divers types de données, y compris des textes, des images, des données tabulaires et des données séquentielles.

Les architectures d'intelligence artificielle générative, telles que les GANs (Generative Adversarial Networks), les LLMs (Large Language Models) et les autoencodeurs variationnels (VAE), jouent un rôle crucial dans la génération de données synthétiques. Ces modèles sont particulièrement efficaces pour augmenter les ensembles de données existants, ce qui est essentiel pour entraîner d'autres modèles de machine learning avec des ensembles de données plus diversifiés et représentatifs.

Un domaine d'application particulièrement intéressant est celui des moteurs électriques, qui sont largement utilisés aujourd'hui et jouent un rôle crucial dans divers secteurs de l'industrie, notamment le transport. Ces moteurs, provenant de multiples fabricants et marques, nécessitent une maintenance prédictive pour garantir leur bon fonctionnement et prolonger leur durée de vie. Cependant, la diversité des marques et des modèles de moteurs pose un défi en termes de collecte de données suffisantes et variées pour chaque type de moteur.

Dans ce contexte, l'IA générative peut être utilisée pour générer des données synthétiques qui couvrent un large éventail de moteurs électriques. En généralisant sur toutes les variétés de moteurs existants, l'IA générative permet d'augmenter les ensembles de données, ce qui est crucial pour entraîner des modèles de classification et de prédiction plus précis et robustes. Ces modèles peuvent ensuite être utilisés pour effectuer une maintenance prédictive efficace, réduisant ainsi les temps d'arrêt et les coûts de maintenance.

Ce travail se concentrera sur l'application de l'IA générative pour la génération de données de type séries temporelles. Nous viserons à généraliser ces données pour qu'elles représentent une large gamme de moteurs électriques. L'objectif final est de faciliter la maintenance prédictive de ces moteurs en utilisant des ensembles de données augmentés et diversifiés, permettant ainsi d'améliorer la fiabilité et l'efficacité des systèmes de maintenance.

Dans cet article, nous présentons les méthodes d'intelligence artificielle générative dans le contexte de l'augmentation de dataset pour la maintenance prédictive. Nous utilisons notamment les réseaux adversatifs génératifs (GAN) et les modèles de diffusion. Les GAN, qui sont généralement composés d'un générateur et d'un discriminateur, permettent de générer des données synthétiques où le générateur crée des données et le discriminateur évalue la qualité de ces données. Par ailleurs, nous abordons les modèles de diffusion qui génèrent des données à partir d'un bruit gaussien. Nous détaillons les différentes étapes nécessaires pour générer des séries temporelles et discutons des méthodes d'évaluation pour apprécier la qualité des données générées par ces modèles. En outre, nous appliquons des techniques de traitement du signal pour visualiser les données dans le domaine fréquentiel.

Nous commençons par une revue de la littérature sur l'apprentissage profond et les différentes architectures existantes. Nous y présentons également des modèles génératifs tels que les autoencodeurs variationnels (VAE), les GAN et les modèles de langage de grande taille (LLM). Le dernier chapitre de cette revue bibliographique est consacré aux bases de la maintenance prédictive, aux composants des moteurs électriques, ainsi qu'aux techniques de traitement du signal comme la transformation de Fourier rapide (FFT).

partie I

Etude bibliographique

Chapitre 1

Apprentissage profond

1.1 Introduction

L'apprentissage profond (*deep learning* en anglais) est une branche de l'intelligence artificielle (IA) qui s'intéresse à la résolution des problèmes intuitifs, c'est-a-dire des tâches qui sont faciles à réaliser par les humains mais difficiles à décrire formellement. Ce sont des problèmes qui semblent automatiques, comme la reconnaissance des mots parlés ou des visages dans les images. L'apprentissage profond permet aux ordinateurs d'apprendre des concepts complexes en rassemblant de l'expérience. Cela permet d'éviter la spécification formelle des connaissances dont l'ordinateur a besoin [Goodfellow et al. 2016].

L'apprentissage profond utilise des réseaux de neurones profonds pour résoudre ces problèmes. Ces réseaux sont des modèles computationnels qui imitent le fonctionnement du cerveau humain [McCulloch et al. 1943, Rosenblatt 1958]. Ils sont constitués de plusieurs couches de neurones artificiels cachées qui traitent les données d'entrée.

Il existe trois grandes catégories d'apprentissage automatique : *supervisé*, *non-supervisé* et *semi-supervisé*. Dans l'apprentissage supervisé, on utilise un ensemble de données étiquetées, tandis que dans l'apprentissage non-supervisé, on ne dispose pas d'un ensemble de données étiquetées. L'apprentissage semi-supervisé est une combinaison d'apprentissage supervisé et non-supervisé. Dans l'apprentissage semi-supervisé, un ensemble de données est étiqueté, mais la majorité des données sont non étiquetées [Goodfellow et al. 2016, Bishop 2016].

Dans l'apprentissage profond, plusieurs types d'architecture existent, chacune adaptée à des tâches spécifiques. Parmi les plus courants, on trouve : les *réseaux de neurones convolutionnels* (CNN), les *réseaux de neurones récurrents* (RNN), les *réseaux de neurones générateurs adversaires* (GAN) et les *réseaux de neurones de transformation* (Transformer) [Goodfellow et al. 2016].

Dans ce chapitre, nous allons expliquer brièvement les différentes notions en relation avec l'apprentissage profond, telles que les couches du réseau, les fonctions d'activation, les types de réseaux, les connexions et les poids, le processus d'apprentissage et les types d'apprentissage.

1.2 Réseau de neurones artificiels

Un réseau de neurones artificiels (*Artificial Neural Network* en anglais) est un modèle de traitement de l'information construit de couches de neurones interconnectées qui traitent les données d'entrée en les transmettant à travers des poids de connexion qui peuvent être ajustés par un processus d'apprentissage [Aggarwal 2018]. Ce réseau s'inspire du fonctionnement des neurones biologiques du cerveau.

Chaque neurone dans les couches cachées du réseau reçoit des signaux d'entrée à partir des neurones précédents, les somme, et les transmet aux neurones de la couche suivante à travers une fonction d'activation. Les réseaux de neurones peuvent avoir plusieurs couches cachées, qui permettent de modéliser des relations non linéaires complexes entre les données d'entrée et de sortie. Ces réseaux neuronaux peuvent compter jusqu'à 150 couches, d'où le nom "profond". [Goodfellow et al. 2016].

Les réseaux de neurones peuvent faire des prédictions précises sur des données nouvelles qui ne sont pas vues pendant l'entraînement. Ils peuvent donc apprendre des relations complexes entre les données d'entrée et de sortie, ce qui leur permet de généraliser et de prédire les sorties pour de nouvelles données. Cependant, la qualité des prédictions dépend fortement de la qualité et de la quantité des données d'entraînement. Si les données d'entraînement sont mauvaises ou insuffisantes, les prédictions pour de nouvelles données peuvent être inexactes [Goodfellow et al. 2016].

Les réseaux de neurones artificiels sont généralement caractérisés par :

- **Traitement parallèle** : les réseaux de neurones sont capables d'effectuer plusieurs calculs simultanément. Cela les rend bien adaptés aux tâches nécessitant des calculs à grande échelle, telles que la reconnaissance d'images, la reconnaissance de la parole, et la traduction automatique [Goodfellow et al. 2016].
- **Apprentissage hiérarchique** : les modèles d'apprentissage profond sont généralement structurés en plusieurs couches . Chaque couche possède un niveau d'abstraction différent. Cela permet au modèle d'apprendre des motifs et des relations complexes dans les données, et plus le réseaux est profond, plus la capacité du modèle à découvrir ces relations est grande [Goodfellow et al. 2016].
- **Grandes quantités de données** : les modèles d'apprentissage profond nécessitent de grandes quantités de données pour s'entraîner efficacement. En effet, les modèles comportent un grand nombre de paramètres qui ne peuvent être réglés qu'à partir d'une grande quantité de données [Goodfellow et al. 2016].

1.3 Connexions et poids

Un réseau de neurones est constitué de nœuds et des connexions entre eux [Aggarwal 2018]. Chaque nœud possède un **ensemble d'entrées** (qui sont souvent les sorties des nœuds de la couche précédente), un **poids** et une valeur ajoutée appelée le **biais**. Dans

les réseaux neuronaux, le biais est un paramètre supplémentaire qui est ajouté à chaque neurone pour ajuster sa sortie. Il permet au réseau de déplacer la fonction d'activation horizontalement[Goodfellow et al. 2016].

Lorsque des signaux entrent dans un neurones, chaque signal est multiplié par le poids associé à son entrée, puis additionné avec les autres résultats. Le biais est ensuite ajoute au résultat final et ce dernier est transmet vers les entrées des neurones de la couche suivante en passant par une fonction d'activation (voir la figure **??**) [McCulloch et al. 1943].

On peut dire que la taille du réseau de neurones est définie par le nombre de ses paramètres et le nombre de ses couches, qui sont des variables appelées **hyperparamètres**. Par contre, les poids et le bais sont des paramètres entraînables. Au début de l'entraînement, on affecte à ces deux paramètres des valeurs aléatoires, et au fur et à mesure, les valeurs de ces deux paramètres sont ajustées et modifiées afin d'obtenir les bonnes valeurs [Aggarwal 2018, Goodfellow et al. 2016].

1.4 Fonction d'activation

Un neurone dans le réseau artificiel calcule la somme pondérée de ses entrées et la valeur résultante de cette opération passe par une fonction appelée **fonction d'activation** (ou **fonction de transfert**) avant d'être transférée vers les neurones de la couche suivante. La sortie de neurone est donc calculée selon la formule **??** [McCulloch et al. 1943].

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{1.1}$$

Où :

w_i : le poids associé à l'entrée i
x_i : la valeur associée à l'entrée i
n : le nombre total d'entrées
b : le biais (constante entraînable ajoutée)
f : la fonction d'activation
y : la sortie du neurone

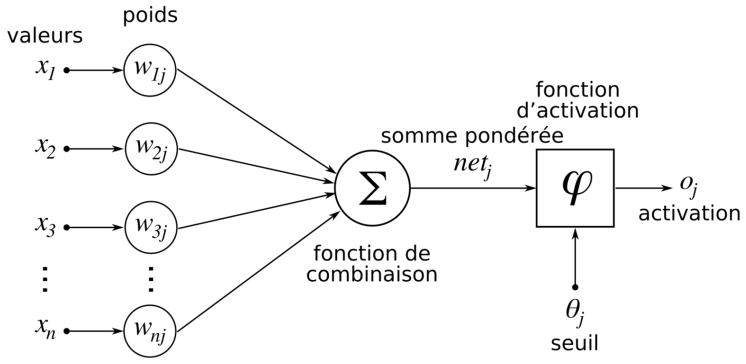

FIG. 1.1 : Le fonctionnement d'un neurone artificiel [McCULLOCH et al. 1943].

La fonction d'activation est utilisée pour introduire de la non-linéarité dans le modèle, permettant ainsi de modéliser des relations complexes entre les données d'entrée et de sortie [GOODFELLOW et al. 2016]. Les propriétés d'une fonction d'activation doivent être vérifiées dans un problème d'apprentissage profond. Ces propriétés sont :

- **Non-linéarité** : lorsque la fonction d'activation est non linéaire, il est possible de prouver qu'un réseau neuronal à deux couches peut approximer n'importe quelle fonction continue sur un domaine compact à une précision arbitraire, ce que l'on appelle le **théorème d'approximation universelle** [GOODFELLOW et al. 2016].
- **L'intervalle** : lorsque l'intervalle des valeurs est fini, l'apprentissage de manière générale est plus efficace.
- **Différentiabilité** : cette propriété est importante quand les méthodes d'optimisation sont basées sur le gradient, car elles cherchent à optimiser l'apprentissage en se basant sur la différentiabilité de la fonction.
- **Monotonie** : Une fonction d'activation est monotone si sa sortie augmente (ou diminue) à mesure que son entrée augmente. Cela garantit que le gradient de la fonction est toujours positif ou négatif, simplifiant ainsi l'apprentissage.
- **Efficacité en termes de calcul** : les fonctions d'activation doivent être efficaces en termes de calcul, afin que le réseau puisse être utilisé dans des applications en temps réel, sans ralentir le processus de l'apprentissage.

Parmi les fonctions d'activation les plus couramment utilisées dans les réseaux de neurones, on peut citer :

- **La fonction Sigmoïde** : si la probabilité d'un résultat est comprise entre 0 et 1, la fonction sigmoïde est le meilleur choix. Cette fonction est largement utilisée grâce à son intervalle et sa différentiabilité.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.2}$$

- **La fonction Unité linéaire rectifiée (ReLU)** : c'est une fonction qui possède une dérivée et permet la rétropropagation (backpropagation) tout en étant efficace

sur le plan informatique. Cependant, elle n'active pas les neurones en même temps, et c'est considéré comme désavantage pour cette fonction.

$$ReLU(x) = \max(0, x) \tag{1.3}$$

- **La fonction Tangente hyperbolique (Tanh)** : cette fonction est très identique à la fonction d'activation sigmoïde. Sa plage de sortie est comprise entre -1 et 1. Avec cette fonction, plus l'entrée est grande, plus la valeur de sortie sera proche de 1, et plus l'entrée est petite, plus la sortie sera proche de -1.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.4}$$

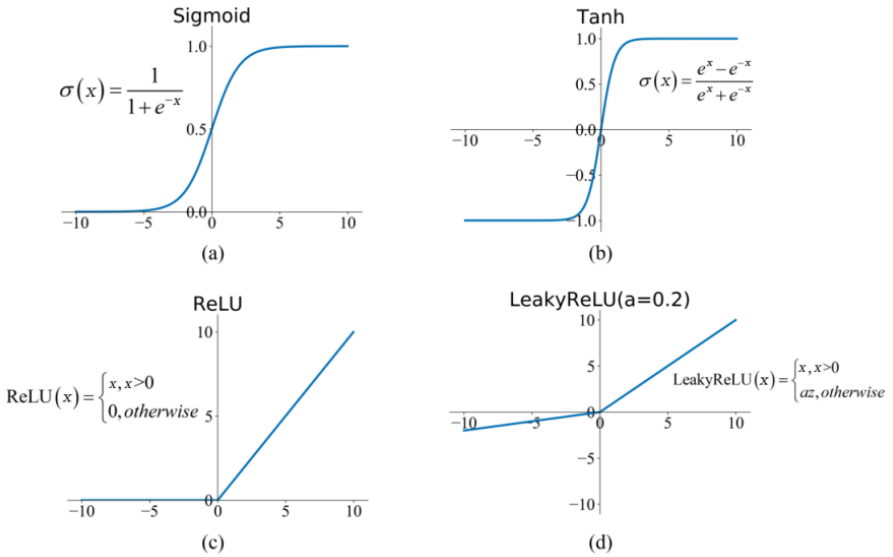

FIG. 1.2 : Les fonctions d'activations couramment utilisées [Junxi FENG et al. 2019].

1.5 Couches dans un réseau de neurones

Une couche (*layer* en anglais) est une succession verticale des neurones. Mathématiquement, elle est vue comme une composition de deux fonctions h et g où g est une fonction linéaire et h une fonction d'activation non linéaire. Cette composition de fonction est définie par l'équation **??** [GOODFELLOW et al. 2016].

$$y = h(g(x) + b) \tag{1.5}$$

Une couche intermédiaire est donc l'ensemble des nœuds verticaux qui sont connectés à la couche précédente et à la couche suivante. La connectivité entre les couches détermine la manière dont les informations circulent sur le réseau. La façon de connexions des nœuds entre eux est différente d'une architecture à une autre [GOODFELLOW et al. 2016], et c'est ce qui détermine le type d'une couche :

- **Couche entièrement connectée** : tous les neurones d'une couche sont connectés à tous les neurones de la couche suivante.
- **Couche partiellement connectée** : certains neurones ne sont pas connectés aux neurones de la couche suivante.

Les couches sont le composant principal des réseaux de neurones. Elles ont plusieurs caractéristiques qui définissent leur comportement et influencent les performances globales du réseau. Ces caractéristique sont les suivants :

- **La matrice de poids** : Dans une couche d'un réseau de neurones, la matrice de poids est une matrice de paramètres qui représente les connexions entre les neurones d'entrée et les neurones de sortie de cette couche [Aggarwal 2018]. Elle définie la puissance des connexions entre les neurones des différentes couches. Chaque ligne de la matrice correspond aux poids associés à un neurone d'entrée particulier, et chaque colonne correspond aux poids associés à un neurone de sortie particulier,La taille de la matrice de poids dépend du nombre de neurones d'entrée et du nombre de neurones de sortie dans la couche.

  La forme générale de la matrice de poids dans un réseau de neurones peut être exprimée comme suit :

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,m} \\ w_{2,1} & w_{2,2} & \dots & w_{2,m} \\ \dots & \dots & \dots & \dots \\ w_{n,1} & w_{n,2} & \dots & w_{n,m} \end{bmatrix} \tag{1.6}$$

  où $w_{i,j}$ représente le poids de la connexion entre le neurone i de la couche actuelle et le neurone j de la couche suivante et *(n, m)* représente la dimension de la matrice.

  La matrice de poids est crucial pour la performance du réseau neuronal, puisqu'elle détermine la capacité du réseau d'apprendre et généraliser les motifs a partir des données en entrées.

- **Type de couche** : Les couches forment les blocs de construction de base des réseaux de neurones. Elle permettent d'effectuer des calculs complexes et d'apprendre des relations qui existent entre données d'entrée et de sortie. Dans le réseau neuronal, il existe trois types de couches différents :
  - **Couche d'entrée** : Cette couche est responsable de la réception des données d'entrée et de leur transmission à la couche suivante (la première couche parmi les couches cachées).
  - **Couche cachée** : Cette couche traite les entrées de la couche précédente et génère des valeurs de sortie qui sont transmises à la couche suivante. Les réseaux de neurones peuvent avoir plusieurs couches cachées, chacune effectuant différentes opérations sur les entrées.
  - **Couche de sortie** : Cette couche produit la sortie finale du réseau de neurones, qui peut être une classification, une régression ou un autre type de prédiction.

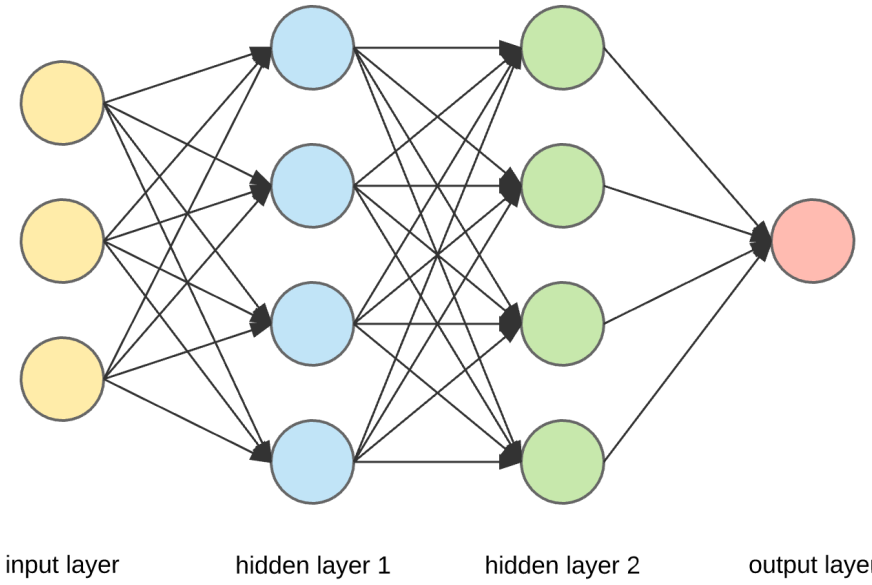

FIG. 1.3 : Schéma simple d'un réseau de neurones feedforward [MUNIASAMY et al. 2020].

1.6 Types de réseaux de neurones

Dans l'apprentissage profond, il existe plusieurs classes de réseaux de neurones, chacune avec sa propre architecture, caractéristiques, algorithme d'apprentissage et application. Dans cette section, nous allons présenter les différentes architecture de réseaux de neurones.

1.6.1 Réseaux feedforward

Les réseaux feedforward (ou réseaux entièrement connectés) sont un des types de réseau de neurones artificiels où les informations circulent dans une seule direction, de l'entrée vers la la sortie [GOODFELLOW et al. 2016]. Ils sont composés d'une succession de couches interconnectées, où chaque neurone d'une couche est connecté aux neurones de la couche suivante. Dans un Réseau de ce type, les données sont introduites dans la première couche du réseau (couche d'entrée), puis elles traversent plusieurs couches cachées avant d'atteindre la couche de sortie (*la figure* **??** *est un schéma simple d'un réseau feedforward*).

Les réseaux feedforward sont indépendants de la structure, c'est-à-dire il n'existe pas d'hypothèses particulières à faire sur l'entrée, ce qui les rend largement applicables. Cependant, ils ont tendance à être moins performants que les réseaux à usage spécial. Les réseaux feedforward sont couramment utilisés dans les applications d'apprentissage supervisé. Ils peuvent également être utilisés dans des applications d'apprentissage non supervisé [AGGARWAL 2018].

1.6.2 Réseaux de neurones récurrents (RNN)

Les réseaux de neurones récurrents (RNN) sont des architectures conçus pour fonctionner avec des données séquentielles, telles que : la reconnaissance de la parole, la recon-

naissance de la voie, l'analyse de séries chronologiques et le traitement du langage naturel [Goodfellow et al. 2016]. Les principales caractéristiques des RNN sont :

- **Connexions récurrentes** : Les connexions dans les réseaux récurrents sont des connexions récurrentes qui permettent à l'information de persister au fil du temps. Cela signifie que la sortie du réseau à un pas de temps est réinjectée en entrée du réseau au pas de temps suivant.
- **État caché** : Les réseaux RNN maintiennent un état caché qui représente la mémoire du réseau. Cet état est mis à jour à chaque pas de temps en fonction de l'entrée courante et de l'état caché précédent.
- **RNN bidirectionnels** : Les réseaux RNN bidirectionnels traitent la séquence d'entrée dans les sens avant et arrière, ce qui permet de capturer le contexte des pas de temps passés et futurs.

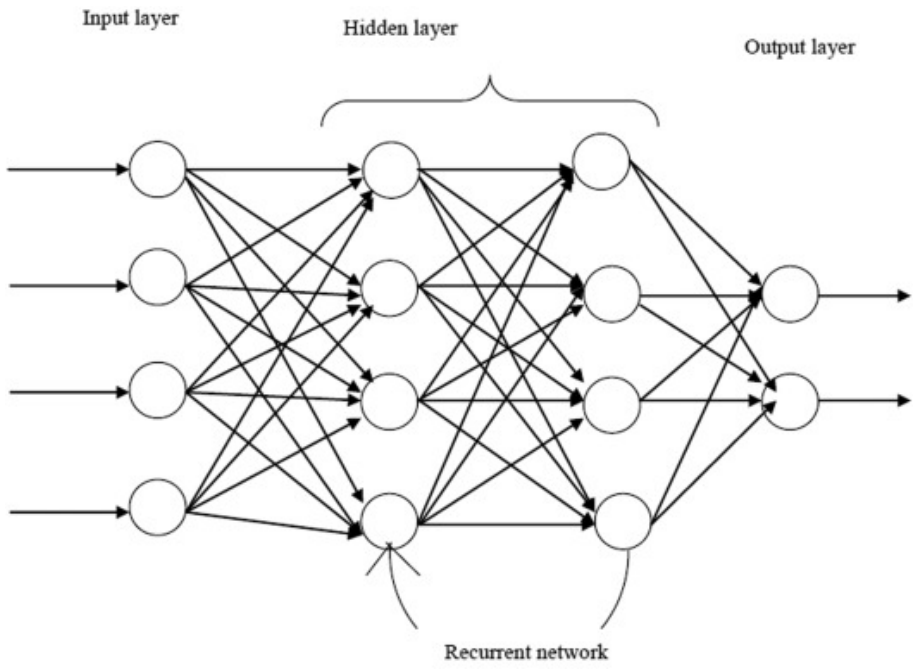

Fig. 1.4 : Exemple d'un réseau de neurones récurrent [Kumaraswamy 2021].

La capacité à conserver une mémoire des pas de temps précédents et à gérer les dépendances à long terme rend les réseaux récurrents utiles pour les tâches qui nécessitent de comprendre le contexte de la séquence d'entrée.

1.6.3 Réseaux de neurones convolutifs (CNN)

L'architecture des réseaux de neurones convolutifs (CNN) est une architecture spéciale qui est bien adaptée à la tâche de classification d'images. Ces réseaux comportent trois types de couches : **convolutives**, **pooling** et **d'activation** [Goodfellow et al. 2016].

Les couches convolutives sont appliquées à l'image en entrée pour l'extraction des caractéristiques importantes de l'image. Ensuite, ces dernières traversent des couches d'activation, qui sont responsables de l'application d'une fonction d'activation non-linéaire à

ces caractéristiques. Les couches d'activation sont suivies de couches de pooling qui réduisent la taille de l'image. Les couches de pooling sont elles même suivies par une couche entièrement connectée qui donne la classification de l'image (la sortie finale du modèle) [Goodfellow et al. 2016].

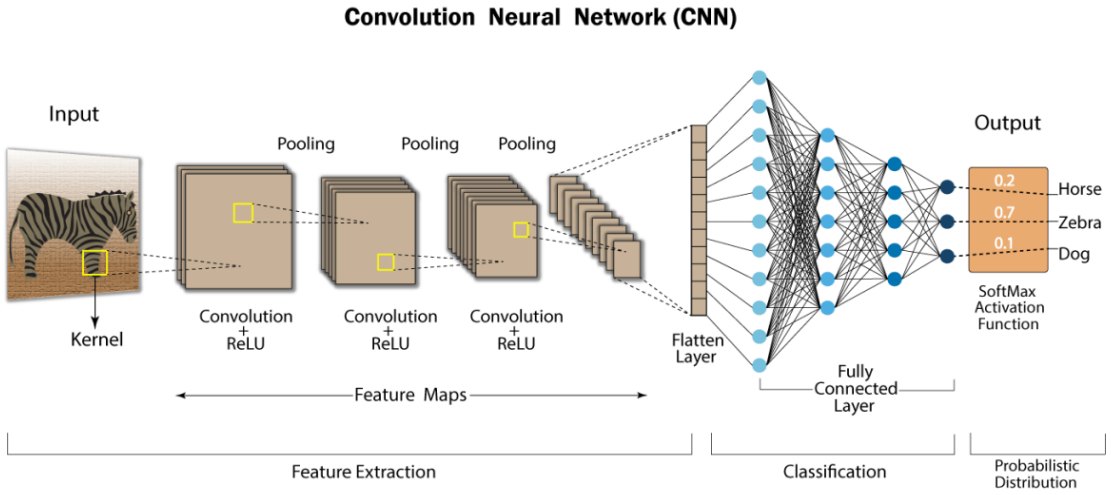

Fig. 1.5 : Le fonctionnement d'un réseau neuronal convolutif [Alharbi et al. 2021].

Couche convolutive

Une couche convolutive est un élément constitutif des réseaux de neurones convolutifs (CNN). Elle est utilisé pour extraire des caractéristiques à partir de données d'entrée, souvent des images, en appliquant un ensemble de filtres convolutifs appris à l'entrée.

Dans une couche convolutive, chaque filtre est convolué avec l'entrée pour produire une carte de caractéristiques. Cette opération est faite en glissant le filtre sur l'entrée et en calculant le produit scalaire à chaque position. Généralement, une couche convolutive possède trois hyperparamètres qui doivent être définis : **le nombre de filtres**, **la taille des filtres** et **la Stride**. Le nombre de filtres détermine le nombre de cartes d'entités produites, tandis que la taille des filtres détermine la taille du champ récepteur de chaque carte d'entités. La stride détermine la quantité de décalage du filtre à chaque étape.

Les couches convolutives sont suivies de fonctions d'activation,et de couches de pooling. Ces dernières permettent de réduire les dimensions spatiales des cartes d'entités. Plusieurs couches convolutives peuvent être empilées pour créer un réseau neuronal convolutif profond.Les couches convolutives sont particulièrement efficaces pour traiter des images et d'autres données de grande dimension avec une structure spatiale, car elles peuvent apprendre automatiquement à détecter les caractéristiques importantes, telles que les bords, les coins et les textures [Kimura et al. 2019, Goodfellow et al. 2016].

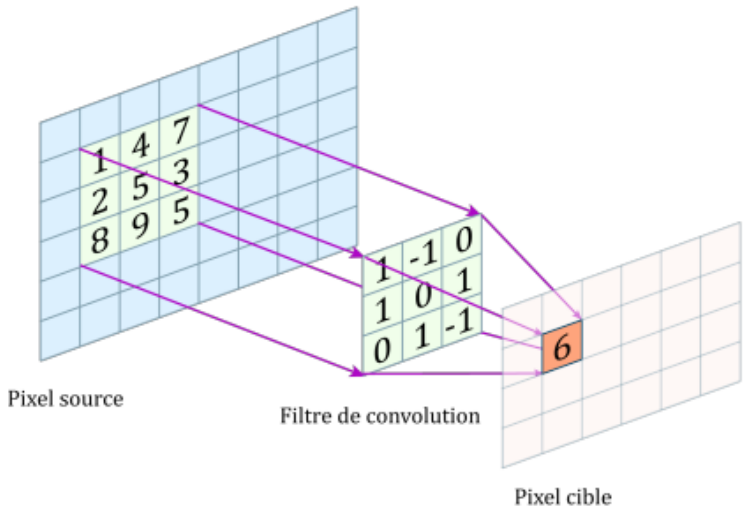

FIG. 1.6 : Exemple du fonctionnement d'une couche convolutive [KIMURA et al. 2019].

Couche pooling

Le pooling est une opération quie est utilisée pour sous-échantillonner les cartes de caractéristiques résultantes des couches convolutives en réduisant leurs dimensions spatiales tout en conservant les informations importantes. Parmi les types de pooling, on trouve le pooling maximal et le pooling moyen [GOODFELLOW et al. 2016].

Dans le pooling maximal, la valeur maximale de chaque région est sélectionnée comme valeur représentative, tandis que dans le pooling moyen, la valeur moyenne est calculée à la place. Il en résulte une carte d'entités plus petite avec une résolution spatiale réduite, qui peut être traitée plus efficacement par les couches suivantes du réseau.

Cependant, le pooling excessive peut entraîner une perte d'informations spatiales importantes. Il est donc important d'équilibrer la quantité de pooling avec les besoins du réseau et la nature de la tâche à accomplir.

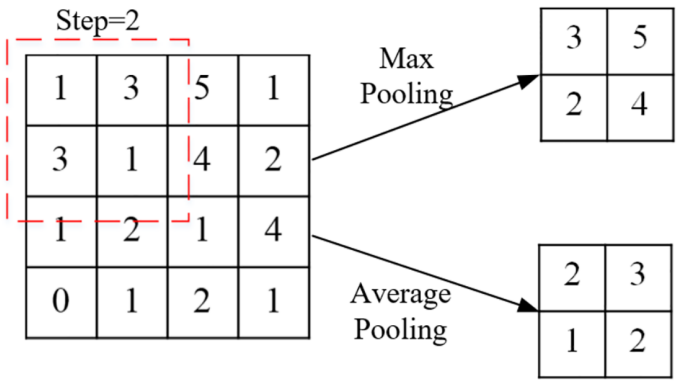

FIG. 1.7 : Exemples de pooling maximal et pooling moyen [HU et al. 2022].

1.6.4 Réseaux générateurs adversaires (GAN)

Les GAN (Generative Adversarial Networks) sont utilisés afin de générer des données synthétiques, telles que des images, des vidéos ou des sons, qui ressemblent à des données réelles. Les GAN sont composés de deux parties : le générateur et le discriminant [Goodfellow et al. 2020].

Le générateur prend comme entrée un vecteur de bruit aléatoire et produit en sortie une image synthétique, tandis que le discriminant en entrée prend une image et produit en sortie une probabilité qui indique si l'image est réelle ou synthétique. Les deux réseaux sont formés en même temps, avec le générateur essayant de tromper le discriminant en produisant des images synthétiques qui sont indiscernables des images réelles [Goodfellow et al. 2020, Jie Feng et al. 2020].

Les GAN sont capables de produire des résultats très proches des données réelles, allant de portraits d'humains à des paysages naturels, en passant par des objets et des bâtiments. Cependant, ils sont souvent difficiles à former et peuvent être instables, en particulier lorsque les données d'entrée sont complexes.

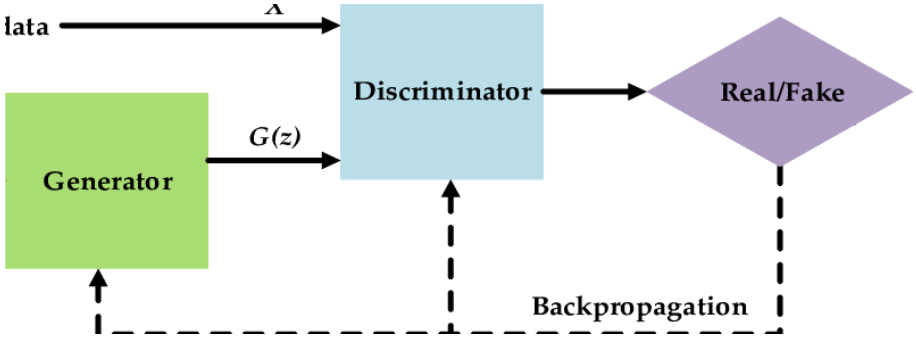

Fig. 1.8 : Un modèle de réseau générateur adversaire (GAN) [Jie Feng et al. 2020].

1.6.5 Réseaux de neurones de transformation (Transformer)

Les réseaux de neurones de transformation (Transformers) sont une architecture de réseau de neurones conçue par Google. Il sont utilises principalement pour le traitement du langage naturel, la traduction du texte et la génération de texte.

De ce fait, les Transformers ressemblent aux réseaux récurrents (RNN) mais la principale différence entre eux est que les Transformers n'utilisent pas de couches récurrentes pour modéliser les dépendances séquentielles. Au lieu de cela, ils utilisent une technique appelée "self-attention", qui permet au modèle de s'auto-atténuer sur les parties importantes de l'entrée [Vaswani et al. 2017].

Dans un réseau de neurones à auto-attention, chaque mot ou token en entrée est représenté par un vecteur, et ces vecteurs sont utilisés pour calculer les scores d'attention entre les différentes parties de l'entrée. Ces scores sont ensuite utilisés pour pondérer les vecteurs en entrée, en mettant plus d'importance sur les parties les plus pertinentes [Vaswani et al. 2017].

Les Transformers ont contribué significativement aux avancées récentes dans le domaine du traitement du langage naturel, en particulier dans la traduction automatique et de la génération de texte. Les modèles les plus avancés, telles que GPT-3 de OpenAI ou Bard de Google, utilisent des architectures de Transformers massives avec des milliards de paramètres.

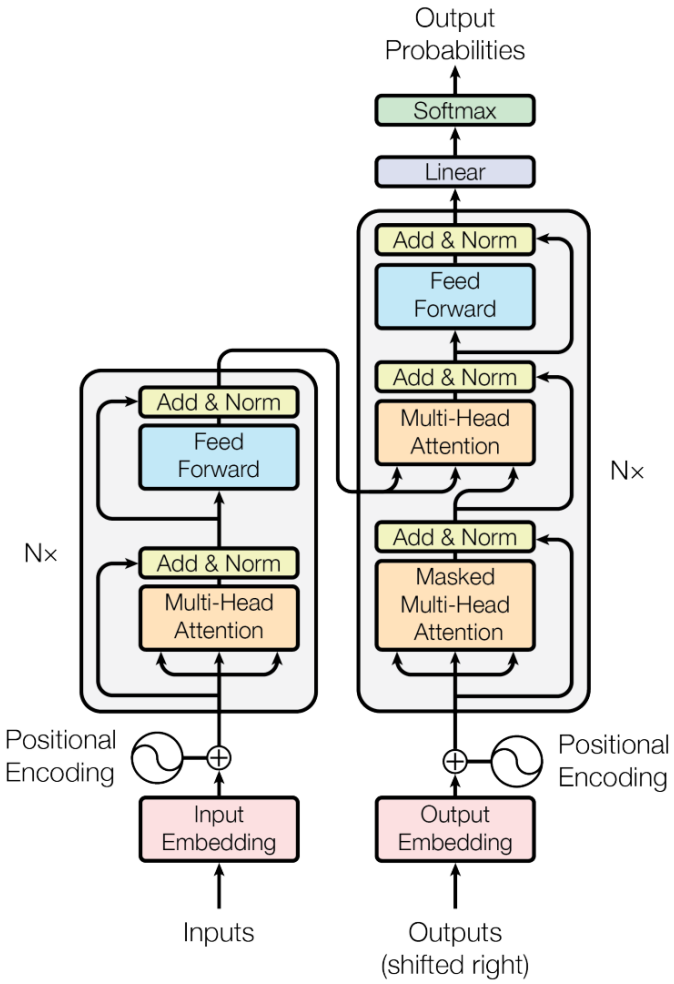

Fig. 1.9 : Architecture d'un reseau de neurones de transformation (Transformer) [Vaswani et al. 2017].

1.6.6 Réseaux résiduels (ResNet)

Les réseaux résiduels ont été conçus par Microsoft afin de résoudre le problème des gradients qui disparaissent dans les réseaux de neurones très profonds, ce qui peut rendre l'entraînement difficile et diminuer significativement les performances du réseau.

Un ResNet est composé d'une ensemble de blocs résiduels, qui sont constitués de plusieurs couches avec des connexions de raccourci qui contournent une ou plusieurs couches. Ces raccourcis permettent aux gradients de circuler plus facilement à travers le réseau et évitent qu'ils ne disparaissent à mesure que le réseau devient plus profond [He et al. 2016].

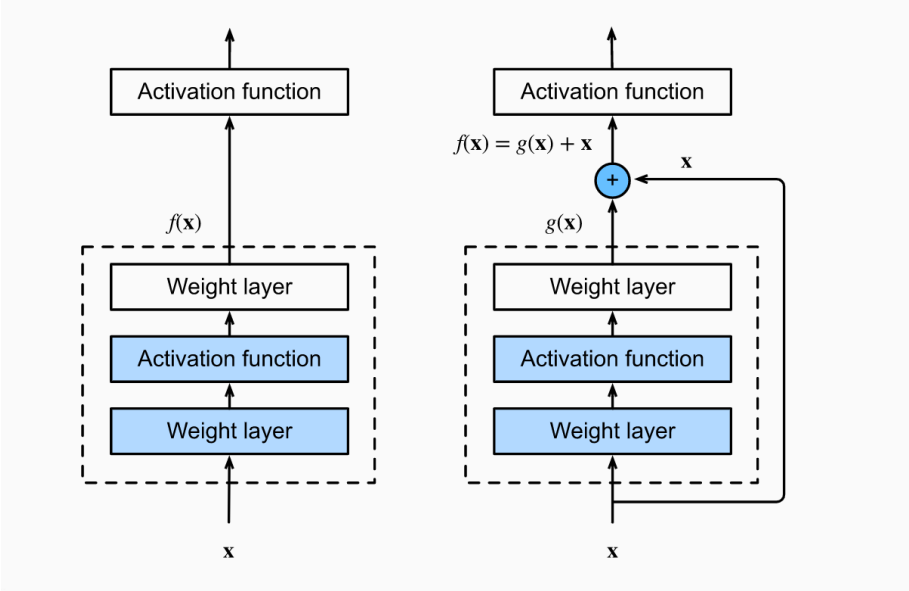

FIG. 1.10 : Un bloc régulier (gauche) et un bloc résiduel (droite) [DONG et al. 2022].

ResNet est très efficace dans les tâches de vision par ordinateur, telles que la classification d'images, la détection d'objets et la segmentation. Il a joué un rôle important dans l'avancement de l'état de l'art en apprentissage profond et en vision par ordinateur, et continue d'être un domaine de recherche actif.

1.6.7 Réseaux de neurones en graphe (GNN)

Les réseaux de neurones en graphe (GNN) sont conçus pour traiter des données structurées sous forme de graphes. Les GNN sont capables d'apprendre et de faire des prédictions sur les données du graphe en propageant les informations à travers les bords et les nœuds du graphe [ZHOU et al. 2020].

L'idée de base derrière les GNN est d'utiliser un ensemble de paramètres entraînables pour transformer les caractéristiques de chaque nœud dans le graphe en fonction des caractéristiques de ses nœuds voisins (plongement de graphe). Ce processus est répété de manière itérative sur plusieurs couches du réseau, permettant au GNN d'apprendre des motifs et des relations complexes à partir des données [ZHOU et al. 2020].

Parmi les tâches réalisées par ce type de réseaux :

- **Classification des graphes** : Les GNN peuvent être utilisées pour classer des graphes en différentes catégories, tels que dans le cas de l'analyse des réseaux sociaux, la classification de textes et classification des molécules.
- **Prédiction de lien** : Les GNN peuvent prédire le lien manquant entre une paire de nœuds dans un graphe avec une matrice d'adjacence incomplète. Ils sont souvent utilisés pour les réseaux sociaux (tel que la suggestion des amis)
- **Plongement de graphe** : Les GNN permettent de faire la transformation de

graphe en vecteurs, en préservant les informations pertinentes sur les nœuds, les arêtes et la structure générale du graphe.

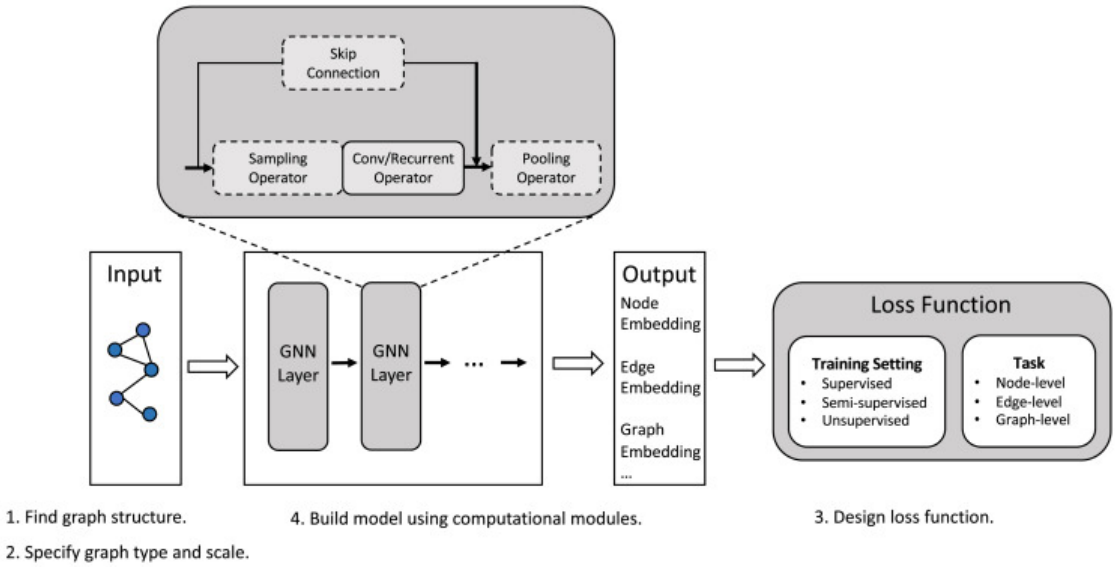

Fig. 1.11 : Le pipeline de conception générale pour un modèle GNN [Zhou et al. 2020].

1.7 Le processus d'apprentissage

L'apprentissage est le processus itératif et continu d'ajustement des paramètres du réseau neuronal afin d'obtenir une meilleure précision du modèle [Goodfellow et al. 2016]. Il peut être complexe car il nécessite souvent une combinaison de techniques telles que le **prétraitement** des données, le choix de l'architecture du réseau de neurones et des hyperparamètres, ainsi que l'algorithme d'optimisation.

Ce processus d'apprentissage commence d'abord par l'initialisation aléatoire des paramètres (poids) du modèle. Ensuite, le modèle est entraîné sur un ensemble de données (**dataset**) en utilisant un algorithme d'optimisation pour ajuster les poids du modèle afin de minimiser la perte.

Lors de l'entraînement, le modèle est alimenté en entrée avec des exemples à partir du dataset d'entrainement et compare sa sortie à la sortie attendue. Ensuite, il calcul la perte en faisant la différence entre la sortie prédite et la sortie attendue. L'algorithme de **rétropropagation** de gradient ajuste les poids du modèle en claculant les gradient de la fonction de perte afin de la minimiser.

Le processus d'apprentissage continue jusqu'à ce que la performance du modèle sur le dataset de validation arrête de s'améliorer ou jusqu'à ce qu'un nombre prédéfini d'itérations d'apprentissage soit atteint. Le modèle final est ensuite utilisé pour effectuer des prédictions sur de nouvelles données.

Ce processus peut être résumé dans les points suivants :

- L'apprentissage consiste à modifier progressivement les paramètres (poids) en passant un lot de données en entrée et en évaluant le taux de perte.
- La définition de la fonction de perte est utilisée pour les modifications de réseaux dans le processus de l'entraînement.
- La modification des poids du réseau est effectuée grâce a l'algorithme de rétropropagation (backpropagation).

1.7.1 Descente de gradient

La descente de gradient est une méthode utilisée dans le processus d'optimisation. Elle est basée sur la différentiabilité d'une fonction et elle est appliquée pour calculer la fonction dérivée du premier ordre pour trouver le minimum de la fonction de perte [Goodfellow et al. 2016]. Sa simplicité d'application est l'un des avantages de cette méthode.

L'algorithme commence par l'initialisation aléatoire des paramètres (poids) du modèle. Ensuite, il calcule le gradient de la fonction de perte par rapport à chaque paramètre. Le gradient indique la direction dans laquelle la fonction de perte augmente le plus, donc l'algorithme met à jour les paramètres dans la direction opposée au gradient pour réduire la valeur de perte. Ce processus est répété itérativement jusqu'à ce que la valeur de perte ne puisse plus être réduite ou jusqu'à ce qu'un critère d'arrêt soit atteint. Le taux d'apprentissage est un hyperparamètre qui contrôle la taille des mises à jour de paramètres et la vitesse de convergence de l'algorithme [Goodfellow et al. 2016].

Des variantes de la descente de gradient existent, telles que la descente de gradient stochastique, Adam et la descente de gradient avec moment.

1.7.2 Propagation de l'erreur

La propagation d'erreur est utilisée dans le processus d'apprentissage pour calculer le gradient de la fonction de perte par rapport aux paramètres du modèle [Goodfellow et al. 2016].

Dans la propagation d'erreur, l'erreur est propagée en arrière à travers les couches du modèle, en commençant par la couche de sortie et en allant vers l'entrée. Chaque couche calcule la dérivée de sa sortie par rapport à ses entrées, qui est ensuite multipliée par l'erreur propagée de la couche suivante. Ce processus est répété jusqu'à ce que l'erreur soit propagée jusqu'à la couche d'entrée, où le gradient de la fonction de perte par rapport aux paramètres du modèle est obtenu [Goodfellow et al. 2016].

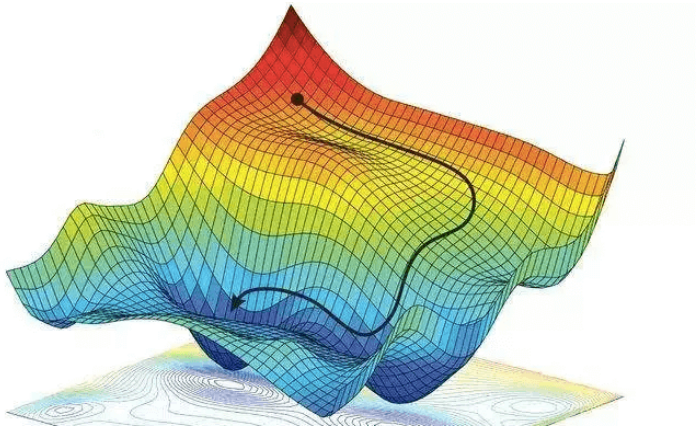

FIG. 1.12 : Fonction de taux d'erreur [AMINI et al. 2018].

1.7.3 Hyperparameters

Les hyperparamètres sont les paramètres qui sont définis avant de lancer le processus d'apprentissage et ils contrôlent l'entraînement. Les valeurs des hyperparamètres, contrairement aux paramètres du modèle, ne sont pas apprises lors de l'apprentissage [GOODFELLOW et al. 2016]. Parmi les hyperparamètres, on peut définir :

- **Taux d'apprentissage** : Il contrôle la vitesse d'apprentissage du modèle à partir des données.
- **Nombre d'époques** : Le nombre d'époques correspond au nombre d'itération sur le dataset d'entraînement.
- **Taille du batch** : Elle représente le nombre d'échantillons des données d'entraînement qui sont utilisés dans un passage avant/arrière (feedforward et backpropagation).
- **Nombre de couches** : C'est un hyperparamètre qui caractérise la profondeur du réseau.
- **Fonction d'activation** : La fonction d'activation est utilisée pour introduire la non-linéarité dans le modèle. C'est un hyperparamètre qui contrôle la sortie du neurone.
- **Initialisation des poids** : Les valeurs initiales des poids peuvent affecter de manière significative les performances du modèle. Elle affecte le processus de trouver le minimum local ou le minimum global.
- **Paramètre de régularisation** : Il est utilisé pour éviter le **surapprentissage**, c'est-à-dire éviter de construire un modèle qui est trop complexe par rapport à la quantité de données d'entraînement. Cela peut entraîner une adaptation excessive du modèle aux données d'entraînement et une mauvaise généralisation aux données inconnues.
- **Optimiseur** : L'optimiseur est l'algorithme utilisé pour mettre à jour les poids pendant l'entraînement.

Cependant, le réglage de ces hyperparamètres peut être une tâche complexe nécessitant souvent des essais et des erreurs.

1.8 Types d'apprentissage

Il existe plusieurs types d'apprentissage, et chacun de ces types a des applications spécifiques en deep learning, et peut être utilisé pour résoudre différents types de problèmes. Dans ce qui suit, nous parlons brièvement sur les quatre types d'apprentissage en deep learning les plus courants : l'apprentissage supervisé, l'apprentissage non supervisé, l'apprentissage semi-supervisé et l'apprentissage par renforcement.

1.8.1 Apprentissage supervisé

Dans l'apprentissage supervisé, l'apprentissage s'effectue sur un ensemble de données étiqueté, où les étiquettes sont connues à l'avance. En d'autres termes, les données d'entrée sont accompagnées d'étiquettes de sortie ou de valeurs cibles correspondantes. Le but de l'apprentissage supervisé est d'apprendre à prédire les étiquettes à partir des données d'entrée, de sorte que lorsqu'on donne au modèle de nouvelles données en entrée, l'algorithme puisse prédire la sortie correspondante. Pendant le processus d'apprentissage, l'algorithme ajuste itérativement ses paramètres pour minimiser la différence entre la sortie prédite et la sortie réelle [Aggarwal 2018, Goodfellow et al. 2016].

Ce type d'apprentissage est applicable dans plusieurs domaines, tels que la reconnaissance d'images et de la parole, le traitement du langage naturel et la modélisation prédictive dans la finance, la santé, le marketing, etc. Parmi les algorithmes d'apprentissage supervisés, nous pouvons citer la régression linéaire et les arbres de décision.

1.8.2 Apprentissage non-supervisé

L'apprentissage non supervisé se fait sur un dataset non étiqueté, sans aucune information sur les étiquettes [Goodfellow et al. 2016]. En d'autres termes, il n'y a pas de valeurs cibles ou d'étiquettes de sortie fournies à l'algorithme. L'algorithme est laissé à lui-même pour trouver les motifs et relations dans les données, sans recevoir d'instructions explicites sur ce qu'il faut rechercher.

L'utilisation de cette approche d'apprentissage peut impliquer le regroupement de points de données similaires, la découverte de structures ou de caractéristiques cachées dans les données (réduction de la dimensionnalité) ou l'identification de valeurs aberrantes ou d'anomalies dans les données.

Nous pouvons appliquer l'apprentissage non supervisé dans divers domaines, tels que la segmentation des marchés, la détection d'anomalies et l'extraction de caractéristiques. Parmi les algorithmes d'apprentissage non supervisés courants, on trouve le clustering k-means, l'analyse en composantes principales (PCA) et les auto-encodeurs.

L'un des principaux défis dans l'apprentissage non supervisé est d'évaluer la qualité des résultats, car il n'y a pas d'objectifs ou d'étiquettes explicites à comparer. Au lieu de cela, les résultats sont souvent évalués en fonction de leur utilité ou de leur interprétabilité pour une tâche ou un domaine particulier.

1.8.3 Apprentissage semi-supervisé

L'apprentissage semi-supervisé est un type d'apprentissage qui combine des éléments d'apprentissage supervisé et non supervisé. Dans l'apprentissage semi-supervisé, un petit ensemble de données étiquetées est fourni, tandis que la grande partie de données est non étiquetées [Goodfellow et al. 2016].

Le but dans l'apprentissage semi-supervisé est d'utiliser les données étiquetées pour guider le processus d'apprentissage sur les données non étiquetées, afin d'améliorer la précision du modèle. Cela peut être particulièrement utile dans les situations où il est difficile ou coûteux d'obtenir des données étiquetées, mais il existe une abondance de données non étiquetées disponibles.

Il existe plusieurs approches, mais une méthode courante consiste à utiliser les données étiquetées pour créer un modèle, puis à utiliser ce modèle pour faire des prédictions sur les données non étiquetées. Les étiquettes prédites sont ensuite utilisées pour améliorer le modèle, et le processus est répété de manière itérative.

L'un des défis de l'apprentissage semi-supervisé est que la qualité des résultats peut dépendre fortement de la distribution des données non étiquetées. Si les données non étiquetées ne sont pas représentatives du domaine cible, le modèle peut mal fonctionner même avec une grande quantité de données non étiquetées.

1.8.4 Apprentissage par renforcement

L'apprentissage par renforcement se concentre sur la prise de décision. Il s'agit d'une méthode d'apprentissage dans laquelle un agent apprend à prendre des décisions en interagissant avec un environnement. L'agent doit choisir une action à partir d'un état donné, et l'environnement renvoie un signal de récompense ou de pénalité en fonction de l'action choisie. L'objectif de l'agent est de maximiser la récompense totale sur une période donnée [Wiering et al. 2012]. Dans le chapitre suivant, nous présenterons l'apprentissage par renforcement et nous en parlerons avec plus de détails.

1.9 Catégories de données

La division du jeu de données est une étape cruciale avant de commencer l'entraînement du modèle de manière. Elle permet d'éviter les problèmes de surapprentissage. Il est courant de diviser un jeu de données en trois parties distinctes : l'ensemble d'entraînement, l'ensemble de validation et l'ensemble de test [Goodfellow et al. 2016].

- **Ensemble d'entraînement** : Il s'agit de la partie de données utilisée pour entraîner le modèle. Il est important que ce dataset soit représentatif de l'ensemble des données et qu'il contienne une variété de cas d'utilisation différents.
- **Ensemble de validation** : Il s'agit d'un sous-ensemble de l'ensemble de données utilisé pour évaluer les performances du modèle pendant l'entraînement. L'ensemble de validation est utilisé pour régler les hyperparamètres du modèle et pour éviter le surapprentissage. Il est important qu'il soit représentatif et distinct du dataset d'entraînement.
- **Ensemble de test** : Il s'agit d'un ensemble de données utilisé pour évaluer les performances du modèle après son entraînement. L'ensemble de test est utilisé pour obtenir une estimation impartiale de la performance du modèle sur de nouvelles données inédites, donc il est nécessaire que ce dataset soit représentatif de l'ensemble des données et distinct des deux autres datasets cités précédemment.

La distinctivité des datasets assure que le modèle se généralise bien aux nouvelles données invisibles. En règle générale, l'ensemble de données est divisé en ces trois ensembles dans un rapport de 60-20-20 ou 70-15-15 respectivement.

1.10 Défis de l'apprentissage profond

L'apprentissage profond a fait des progrès remarquables ces dernières années et a obtenu des résultats excellents dans divers domaines tels que la vision par ordinateur, le traitement du langage naturel et la reconnaissance de la parole. Cependant, il reste encore plusieurs défis à relever afin d'améliorer encore l'efficacité et l'efficience des modèles d'apprentissage profond. Certains défis majeurs incluent :

- **Rareté des données** : les modèles d'apprentissage profond nécessitent une grande quantité de données pour être entraînés efficacement. Cependant, dans de nombreux domaines, tels que l'imagerie médicale et la conduite autonome, les données sont rares et coûteuses à collecter.
- **Surapprentissage (overfitting)** : les modèles d'apprentissage profond peuvent facilement sur-adapter les données d'apprentissage, en particulier lorsque le modèle comporte un grand nombre de paramètres. Le surapprentissage peut entraîner de mauvaises performances de généralisation sur de nouvelles données.
- **Interprétabilité** : les modèles d'apprentissage profond sont souvent appelés "boîtes noires" car il peut être difficile de comprendre comment ils arrivent à leurs prédictions. Ce manque d'interprétabilité peut compliquer le débogage et l'amélioration des modèles de l'apprentissage profond
- **Limitations matérielles** : les modèles d'apprentissage profond sont coûteux en termes de calcul et nécessitent un matériel spécialisé tel que des unités de traitement graphique (GPU) ou des unités de traitement de tenseur (TPU). Le coût de ce matériel peut constituer un obstacle pour les petits groupes de chercheurs ou les entreprises.

- **Attaques contradictoires** : les modèles d'apprentissage profond peuvent être vulnérables aux attaques contradictoires, où un attaquant manipule délibérément les données d'entrée pour amener le modèle à faire des prédictions incorrectes.

1.11 Conclusion

En conclusion, l'apprentissage profond est devenu un domaine de recherche actif de l'apprentissage automatique qui a révolutionné la façon dont nous abordons de nombreux problèmes difficiles, tels que la vision par ordinateur, le traitement du langage naturel et la robotique. Avec l'avènement d'un matériel puissant, d'ensembles de données à grande échelle et d'algorithmes sophistiqués, les modèles d'apprentissage profond ont connu un avancement remarquable dans plusieurs domaines tels que la reconnaissance d'images, la reconnaissance vocale, la traduction linguistique et la conduite autonome.

Malgré son énorme succès, l'apprentissage en profondeur fait encore face à plusieurs défis, tels que le besoin d'algorithmes d'entraînement plus efficaces et fiables, une meilleure interprétabilité, etc. L'utilisation et l'entraînement de modèles d'apprentissage profond exigent des ordinateurs assez puissants et ne peuvent pas être utilisés sur les appareils moins puissants comme les ordinateur embarqués ou les smartphones, ce qui signifie qu'on a besoin de trouver des moyens pour optimisation ces modèles afin de les utiliser ultérieurement par les machines moins puissantes. Ces méthodes d'optimisation font l'objet du dernier chapitre.

Dans ce qui suit, nous présenterons l'apprentissage profond et ses algorithmes. La raison pour laquelle on a réservé un chapitre complet pour l'apprentissage profond est que ce type d'apprentissage peut aider beaucoup dans l'optimisation des réseaux de neurones profonds, comme nous le verrons plus tard.

Chapitre 2

Intelligence Artificielle Générative(GenAI)

2.1 Introduction

Avec les avancées fulgurantes du deep learning, de nouvelles méthodes d'intelligence artificielle ont émergé, souvent désignées sous le terme de modèles génératifs. Ces technologies de pointe, telles que les auto-encodeurs variationnels, les réseaux adversariaux génératifs (GAN), les modèles de diffusion, les transformers et les modèles de langage de grande taille (LLM) sont capables de produire des données d'une qualité impressionnante, ressemblant de manière frappante aux données réelles. Les données générées par ces modèles sont souvent indiscernables des données authentiques, ce qui pose de nouveaux défis en matière d'identification et d'authenticité.Les modèles génératifs permettent la génération de textes, d'images, de séries temporelles et de données tabulaires avec une grande précision.

La puissance de ces modèles repose sur des ressources de calcul considérables, notamment l'utilisation de GPU, et sur l'accès à des ensembles de données massifs. Par exemple, des modèles tels que ChatGPT-4 ont été entraînés sur l'intégralité du contenu disponible sur Internet.

Dans ce chapitre, nous allons examiner en détail les différentes architectures de ces modèles génératifs, ainsi que les techniques et pratiques associées à leur fonctionnement et à leur mise en œuvre. Nous aborderons les principes de base, les innovations récentes, et les applications potentielles de ces technologies révolutionnaires.

2.2 Les Auto-Encodeurs Variationnels (VAE)

2.2.1 Introduction

Les auto-encodeurs variationnels (VAE) sont des modèles génératifs puissants, reconnus pour leur capacité à apprendre une représentation compacte et structurée des données.

Cette approche a révolutionné le domaine de l'apprentissage non supervisé et des modèles génératifs. Les VAE appartiennent à la classe des modèles génératifs probabilistes, combinant les principes des autoencodeurs et des modèles de variational Bayes pour générer des données nouvelles et similaires à celles d'un jeu de données d'entraînement. Depuis leur introduction par Kingma et Welling en 2013 [Kingma et al. 2012] , les VAE ont connu un grand succès dans diverses applications, allant de la génération d'images à la synthèse de texte.

2.2.2 Fonctionnement

L'Encodeur :L'encodeur est responsable de la transformation des données d'entrée $\mathbf{x}$ en une distribution dans l'espace latent. Plus précisément, il mappe les données d'entrée à une distribution gaussienne paramétrée par une moyenne μ et une variance σ^2. Cette distribution est souvent représentée comme :

$$q(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\mathbf{z} \mid \mu(\mathbf{x}), \sigma^2(\mathbf{x})) \tag{2.1}$$

où $\mathbf{z}$ est la variable latente que l'encodeur cherche à estimer.

Le Décodeur : Le décodeur prend un échantillon $\mathbf{z}$ de la distribution gaussienne dans l'espace latent et génère une reconstruction $\hat{\mathbf{x}}$ des données d'entrée. Il modélise la distribution des données d'entrée conditionnellement à $\mathbf{z}$ comme suit :

$$p(\mathbf{x} \mid \mathbf{z}) \tag{2.2}$$

Typiquement, le décodeur est un réseau neuronal qui produit les paramètres de cette distribution, souvent supposée gaussienne ou Bernoulli selon la nature des données.

L'Espace latent :est la représentation comprimée et structurée des données d'entrée. Il est généralement conçu comme un espace continu, où chaque point de l'espace latent correspond à une instance possible de données générées. L'idée est que cet espace latent capture les caractéristiques essentielles des données d'entrée de manière à permettre la génération de nouvelles instances en échantillonnant de cet espace.

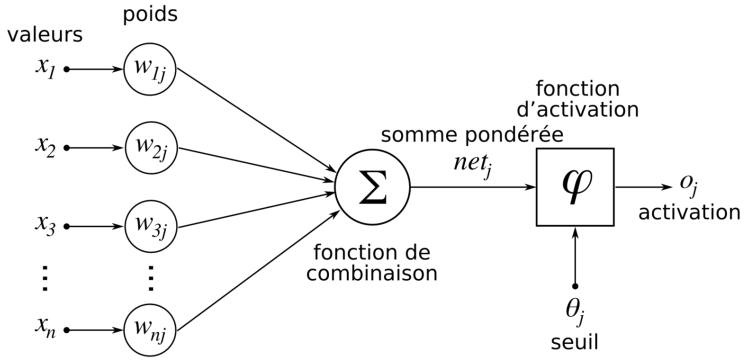

Fig. 2.1 : Le fonctionnement d'un neurone artificiel [McCulloch et al. 1943].

Fonction de Coût : l'objectif d'un VAE est d'optimiser la fonction de coût qui combine deux termes principaux : la divergence de Kullback-Leibler (KL) entre la distribution

latente approximée $q(\mathbf{z} \mid \mathbf{x})$ et la distribution prior $p(\mathbf{z})$, et la vraisemblance de reconstruction $p(\mathbf{x} \mid \mathbf{z})$. La fonction de coût totale, appelée la fonction de perte VAE, est donnée par :

$$\mathcal{L}(\mathbf{x}; \theta, \phi) = \mathrm{KL}\left[q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})\right] - \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x} \mid \mathbf{z})]$$

où :

$$\mathrm{KL}\left[q(\mathbf{z} \mid \mathbf{x}) \parallel p(\mathbf{z})\right]$$

mesure la divergence entre la distribution latente approximée et la distribution prior.

$$\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x} \mid \mathbf{z})]$$

est l'espérance de la log-vraisemblance de la reconstruction des données données la variable latente.

L'optimisation de cette fonction de coût permet au VAE d'apprendre à générer des données qui ressemblent aux données d'entrée tout en apprenant une représentation efficace dans l'espace latent. “‘

partie II

Contribution

Chapitre 3

Conception

3.1 Introduction

Le domaine de l'apprentissage profond a connu une grande croissance de la profondeur des architectures de réseaux de neurones. Cependant, cette croissante pose des défis importants en termes d'exigences de calcul et mémoire et de consommation d'énergie. Ainsi, la nécessité de concevoir des techniques efficaces de compression et d'optimisation des modèles est devenue primordiale. Dans ce chapitre, nous présenterons une méthode automatique pour l'elagage des réseaux neuronaux très profonds qui exploite l'apprentissage par renforcement et le plongement des couches du réseau.

3.2 Vue globale de la solution

L'élagage des réseaux de neurones profonds est devenu une technique essentielle afin de déployer ces réseaux sur des appareils mobiles disposant de ressources de calcul et de stockage limitées. Plusieurs techniques existent pour élaguer un réseau. Cependant, ces techniques nécessitent des experts du domaine afin d'explorer le vaste espace de conception et obtenir un bon compromis entre la taille, la vitesse et la précision du modèle, ce qui peut prendre beaucoup de temps et ne produit pas toujours des résultats optimals.

Dans cette partie, nous proposons une nouvelle méthode d'élagage automatique des réseaux de neurones profonds qui utilise l'apprentissage par renforcement et les technique de plongement pour fournir les pourcentages d'élagage optimals. Notre méthode, contrairement aux méthodes conventionnelles, ne nécessite pas d'experts humains dans le domaine et prend moins de temps.

Dans les réseaux de neurones profonds, les couches ne sont pas indépendantes les unes des autres en raison de la manière dont l'information est propagée et traitée à travers le réseau. Chaque couche prend les sorties de la couche précédente en entrée et génère des sorties qui sont ensuite utilisées comme entrées pour la couche suivante. Cette interconnexion crée des relations et des dépendances entre les couches.

Lors de l'apprentissage, les poids et les biais de chaque couche sont ajustés et chaque couche contribue de manière spécifique à la transformation des données en fonction des

poids et des biais. Donc, les ajustements dans une couche peuvent avoir un impact sur les performances et les caractéristiques apprises par les autres couches, et cela affecte directement la précision de notre modèle. Pour cela, nous avons décider d'utiliser une stratégie d'élagage continue. Nous commençons par l'initialisation des paramètres de notre modèle avec des valeurs aléatoires et puis nous entraînons le modèle jusqu'à l'obtention d'une bonne précision. Ensuite, nous commençons le processus d'élagage où nous utilisons un agent DDPG pour élaguer le réseau couche par couche (comme illustrée dans la figure **??**). Pour chaque couche L_n , l'agent DDPG reçoit comme entrée le plongement de la couche qui permet de préserver les caractéristiques utiles de cette couche, puis il génère en sortie un taux d'élagage a_n. Après l'élagage de la couche L_n avec le taux a_n, l'agent DDPG passe à la couche suivante L_{n+1}.

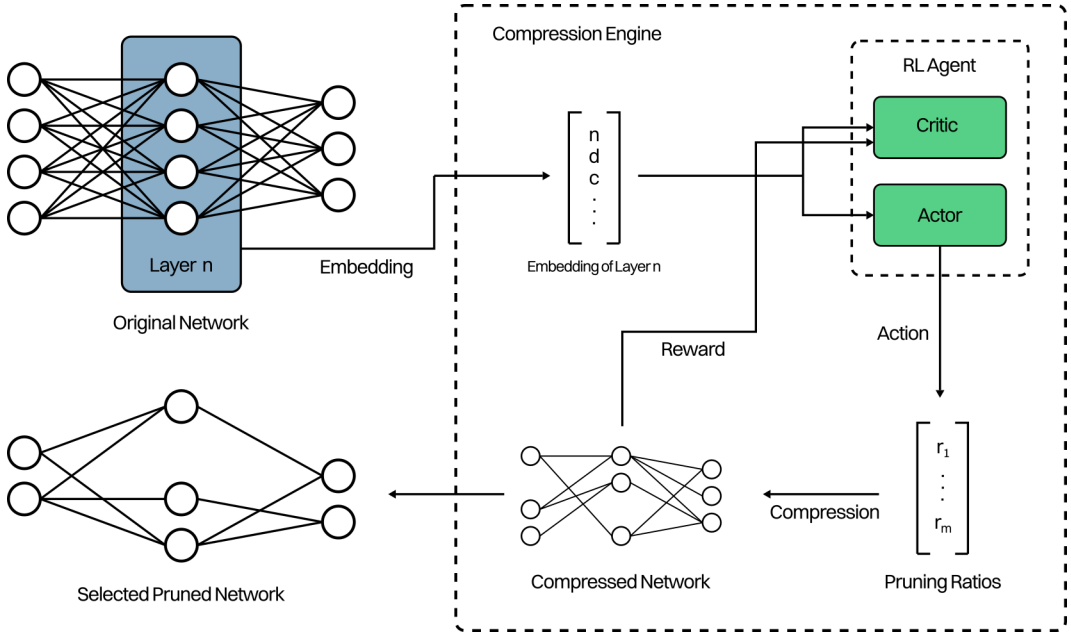

Fig. 3.1 : Présentation du processus d'élagage.

Une fois toutes les couches élaguées, nous évaluons la précision du modèle sans réglage fin afin d'estimer la précision du modèle final (avec le réglage fin). Cette valeur approchée permet d'améliorer le temps de recherche sans avoir à ré-entraîner le modèle à chaque fois. Une fois la recherche terminée, nous effectuons un réglage fin au modèle avec la meilleure précision pour améliorer encore sa précision.

Généralement, nous pouvons décomposer le processus d'élagage en 4 étapes principales (comme illustré dans la figure **??**) :

1. **Initialisation** : Nous commençons par le définition du modèle et l'initialisation aléatoire de ses paramètres.
2. **Entraînement** : Nous entraînons le modèle à l'aide du jeu de données CIFAR-10 jusqu'à ce que nous obtenions la meilleure précision.
3. **Élagage** : Nous utilisons l'agent DDPG pour fournir les pourcentages d'élagage de chaque couche, et nous sélectionnons le meilleure réseau parmi les réseaux élagués pour l'améliorer dans l'étape suivante.

4. **Réglage fin** : Nous ré-entraînons le réseau obtenu dans l'étape précédente pour améliorer encore sa précision.

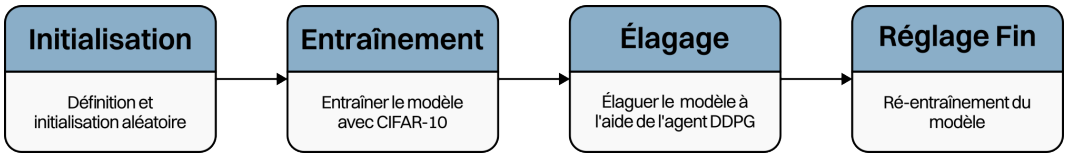

Fig. 3.2 : Schéma global du processus d'élagage.

3.3 Plongements des couches

Il existe plusieurs architectures de réseaux de neurones et les couches et les connexions entre elles diffèrent d'une architecture à l'autre. Dans ce travail, nous allons concentrer uniquement sur les réseaux de neurones de convolution et résiduels, qui utilisent deux types de couches seulement : les couches de convolution et les couches entièrement connectées.

Comme illustrée dans la figure **??**, l'agent DDPG reçoit le plongement de la couche L_n en entrée et fournit en sortie le taux d'élagage a_n de cette couche. La raison derrière l'utilisation du plongement de couche au lieu du plongement d'un graphe de calcul avec encodeur GCN (qui est utilisé dans la méthode de Douag et al. 2022) est que le plongement de couche est généralement plus simple à mettre en œuvre et plus rapide.

D'une part, le plongement de couche peut simplifier la représentation d'entrée et réduire la dimensionnalité par rapport à une représentation graphique, ce qui rend les données plus faciles à traiter et l'apprentissage plus rapide. D'autre part, lors de l'utilisation du graphe de calcul avec GCN, la construction et reconstruction du graphe entraînent une surcharge à chaque étape de l'apprentissage par renforcement. Nous devons toujours reconstruire le graphe de calcul en fonction du taux d'élagage actuel, ce qui peut être coûteux en termes de calculs et peut ralentir le processus d'élagage, tandis que dans les plongements de couches, nous trouvons qu'ils sont plus interprétables et nécessitent moins d'efforts pour les construire, visualiser et comprendre par rapport aux structures graphiques. Finalement, la complexité des méthodes basées sur GCN peut augmenter considérablement lorsque la taille du réseau de neurones augmente, ce qui n'est pas le cas pour les plongement de couches car ils sont plus évolutives et fonctionnent mieux dans des environnements à grande échelle.

Chaque plongemenet s_n est caractérisé par 11 caractéristiques qui sont essentielles pour que l'agent DDPG puisse distinguer une couche d'une autre. Ces caractéristiques sont les suivantes :

$$(n, d, c, h, w, stride, k, FLOPs[n], removed, remaining, a_{n-1}) \tag{3.1}$$

où :

n	: L'indice de la couche
$detc$	: Les canaux de sortie et d'entrée
h	: Le nombre de pixels dans la direction verticale
w	: Le nombre de pixels dans la direction horizontale
$stride$	: La taille du pas du filtre
k	: La taille du noyau
$FLOPs[n]$	: Les FLOPs de la couche L_n
$removed$	: Le nombre total de FLOPs supprimés dans les couches précédentes
$remaining$	: Le nombre de FLOPs restants dans les couches suivantes
a_{n-1}	: Le taux d'élagage de la couche précédente

Après l'extraction de ces caractéristiques, nous effectuons une normalisation afin de les préparer à être utilisées comme entrées dans l'agent DDPG. La normalisation garantit que les valeurs des caractéristiques extraites se situent dans une plage similaire. Ceci est important pour la stabilité de l'entraînement et la convergence de l'agent DDPG. Il existe plusieurs méthodes pour faire la normalisation telles que :

- **Min-Max scaling** : Cette méthode transforme les valeurs des caractéristiques en une plage prédéfinie, souvent [0, 1] ou [-1, 1]. Il faut d'abord Identifier les valeurs minimales et maximales de la caractéristique, puis soustrayez la valeur minimale et divisez par la plage (maximum moins minimum) pour chaque valeur de la caractéristique.
- **Z-Score (standardisation)** : Cette méthode met à l'échelle les valeurs pour avoir une moyenne de 0 et un écart type de 1. D'abord, on calcule la moyenne et l'écart type de la caractéristique. Puis, pour chaque valeur de la caractéristique, on soustrait la moyenne et on divise par l'écart type.
- **Feature Scaling** : Avec cette méthode, on met à l'échelle chaque caractéristique individuellement pour avoir une moyenne de 0 et une variance de 1. Ceci peut être réalisé en soustrayant la moyenne et en divisant par l'écart type pour chaque caractéristique séparément.

Dans notre cas, nous allons utiliser la méthode Min-Max scaling pour garantir que toutes les valeurs sont ajustées proportionnellement pour s'adapter à la plage [0, 1].

3.4 Choix des pourcentages d'élagage

Afin de choisir les meilleurs pourcentages d'élagage, nous disposons de certaines métriques qui guident ce processus. Ces métriques sont essentielles pour l'évaluation des performances des réseaux de neurones élagués. Les métriques que nous examinerons comprennent le temps d'inférence et la taille du réseau.

Le temps d'inférence est une mesure du délai nécessaire pour obtenir une sortie suite à la présentation d'une entrée au réseau, c'est à dire c'est le temps nécessaire pour faire une propagation vers l'avant. Une diminution de ce temps se traduit par une amélioration de la réactivité globale du réseau. De plus, la réduction de la taille du réseau apporte plus

d'avantages, allant de la réduction des besoins en espace de stockage à une consommation énergétique réduite.

Cependant, ces deux métriques sont liées à un paramètre important : le degré de sparsité du réseau, qui, lorsqu'il augmente, il peut engendrer des gains significatifs en termes de taille du réseau et de rapidité d'inférence. Ce paramètre mesure la proportion de paramètres qui ont été élagués par rapport au nombre total de paramètres dans le modèle d'origine. Toutefois, nous pouvons avoir une perte de performances du modèle lorsque nous augmentons extrêmement le degré de sparsité du réseau. Il est donc nécessaire de trouver un équilibre entre la réduction de la taille et le maintien des performances.

Pour augmenter le degré de sparsité du réseau, nous devons réduire/augmenter l'une des 4 mesures suivantes : les paramètres du réseau, les FLOPs, les FLOPS, ou les MAC.

Paramètres

Les paramètres font référence aux poids et aux biais qui sont ajustés pendant l'entraînement du réseau. Chaque connexion entre les neurones est associée à un poids, qui détermine l'importance de cette connexion pour le calcul de la sortie. En ajustant ces poids et biais lors de la phase d'entraînement, le réseau apprend à représenter les relations entre les entrées et les sorties. Le nombre total de paramètres dans un réseau est un facteur crucial qui, avec son augmentation, peut rendre le réseau plus grand en termes de taille et complexe à entraîner.

Prenons l'exemple d'un réseau de neurones convolutionnel (CNN). Chaque couche de ce réseau contient des filtres qui glissent sur l'image d'entrée pour extraire des caractéristiques. Chaque connexion entre un filtre et une région de l'image a un poids associé. Ces poids constituent les paramètres du réseau. Par exemple, si nous avons un filtre de taille 3x3, il y aura 9 poids (un pour chaque connexion) et un biais associé à ce filtre. Si le réseau a plusieurs de ces filtres dans une couche, le nombre total de paramètres augmentera en conséquence.

FLOPs

Les FLOPs (Floating Point Operations) sont une mesure du nombre total des opérations en virgule flottante effectués par le modèle, telles que les additions, les soustractions, les multiplications et les divisions. Un modèle avec un grand nombre de FLOPs peut nécessiter plus de ressources de calcul et de temps pour l'entraînement et l'inférence.

Souvent, le nombre de paramètres est utilisé pour mesurer la complexité du modèle, mais il ne se traduit pas directement en charge computationnelle lors de l'inférence. Dans ce travail, nous allons utiliser les FLOPs comme critère de choix des pourcentage d'élagage au lieu d'utiliser les paramètres. Les raisons de ce choix sont citées dans la section suivante.

FLOPS

Les FLOPS (Floating Point Operations per Second) sont le nombre d'opérations en virgule flottante qui peut être effectuer par seconde et ils sont utilisés pour évaluer la

capacité de calcul et les performances d'un dispositif ou des unités de traitement, telles que les processeurs et les GPU. Plus le nombre de FLOPS est élevé, plus le dispositif est capable de réaliser des calculs complexes rapidement et l'inférence sera plus rapide.

Par exemple, pour une carte graphique (GPU) qui a une capacité de calcul de 10 téraflops ($10 \cdot 10^{12} FLOPS$), elle peut effectuer environ 10 billions d'opérations en virgule flottante par seconde. Cela serait particulièrement utile pour accélérer l'entraînement des réseaux très profonds.

MAC

Un MAC (Multiply-Accumulate Computations) combine une multiplication suivie d'une addition de produits résultants. Il est utilisé pour calculer les sorties des neurones en multipliant les entrées par les poids associés, puis en accumulant ces produits pour obtenir la sortie finale du neurone. Généralement, on considère 1 MAC = 2 FLOPs.

3.4.1 Raisons d'utilisation des FLOPs

Dans les réseaux CNN, la réduction du nombre de paramètres ne se traduit pas nécessairement par une réduction proportionnelle en termes d'opérations en virgule flottante (FLOPs). Ces dernières dépendent de plusieurs facteurs au-delà du simple nombre de paramètres, tels que :

- **Taille des filtres** : La taille des filtres de convolution affecte le nombre de calculs. Les filtres plus grands nécessitent plus de calculs par couche.
- **Taille de l'entrée** : La taille de l'entrée d'une couche affecte également le nombre de calculs. Des dimensions d'entrée plus grandes nécessitent plus de calculs pour être traitées.
- **Stride** : Le stride (pas) détermine comment les filtres de convolution se déplacent à travers les données d'entrée. Les pas plus grands réduisent le nombre de calculs, car ils sautent certaines régions de l'entrée.
- **Pooling** : Les couches de regroupement maximal (maxpool) ou de regroupement moyen (avgpool) sont utilisées dans les CNN pour réduire les dimensions spatiales. Ces couches affectent également la charge de calcul globale.
- **Non-linéarité** : Les fonctions d'activation telles que ReLU introduisent des calculs supplémentaires lors des passes avant et arrière.
- **Connexions sautées et blocs résiduels** : Les connexions sautées ou les blocs résiduels peuvent introduire des calculs supplémentaires.

Prenons un exemple simple avec deux couches de convolution dans le réseau. Supposons que les deux couches ont le le même filtre et stride mais la première couche possède une entrée plus grande que la deuxième, c'est a dire que la deuxième couche nécessitera moins de FLOPs par rapport à la première couche. Nous avons donc deux scénarios :

1. **Élagage en pourcentage de paramètres** : Dans ce scénario, si on élague 25% de la première couche et 75% de la deuxième couche, on va élaguer donc 50% des paramètres ($\frac{25+75}{2}$). Cela réduit le nombre total de paramètres, mais ne correspond pas nécessairement à une réduction proportionnelle des calculs car la première couche a plus de calculs que la deuxième couche (la réduction des FLOPs pourrait être bien inférieure à 50%).
2. **Élagage en pourcentage de FLOPs** : Contrairement à la réduction de paramètres, si on élague 50% de FLOPs, alors on va élaguer 50% des calculs, ce qui est plus avantageux.

3.4.2 Calcul des FLOPs

Comme indiqué précédemment, le nombre des FLOPs peut être influencé par plusieurs facteurs, tels que la taille d'entrée, le stride, la taille du filtre, etc. L'estimation du nombre de FLOPs ne tient peut-être pas compte de toutes les complexités des opérations spécialisées, des optimisations matérielles et des détails d'implémentation, mais elle fournit une estimation approximative de la complexité de notre modèle.

Pour calculer le nombre des FLOPs dans un modèle, nous avons des règles différentes pour chaque type de couche :

- **Couche de convolution**

$$FLOPs = 2 \cdot C_I \cdot C_O \cdot K \cdot O \tag{3.2}$$

- **Couche entièrement connectée**

$$FLOPs = 2 \cdot I \cdot O \tag{3.3}$$

où :

C_I : Le nombre de canaux d'entrée
C_O : Le nombre de canaux de sortie
K : La forme du noyau
I : La taille de l'entrée
O : La taille de la sortie

Supposons que nous avons le modèle suivant qui effectue une classification sur le jeu de données MNIST, où :

- La taille de l'entrée est 28x28x1 (niveaux de gris)
- Nous commençons par 2 convolutions de 5 noyaux de taille (3x3)
- Nous exécutons ensuite une couche entièrement connectée de 128 neurones
- Nous terminons avec une couche entièrement connectée de 10 neurones

Donc, le nombre de FLOPs pour les 4 couches de notre modèle est calculé comme suit :

- **Première convolution** : $2{\times}1{\times}5{\times}(3{\times}3){\times}26{\times}26 = 60840FLOPs$
- **Deuxième convolution** : $2{\times}5{\times}5{\times}(3{\times}3){\times}24{\times}24 = 259200FLOPs$
- **Première couche FC** : $2{\times}(24{\times}24{\times}5){\times}128 = 737280FLOPs$
- **Deuxième couche FC** : $2{\times}128{\times}10 = 2560FLOPs$

Le modèle fera donc $60840 + 259200 + 737280 + 2560 = 1060400$ opérations.

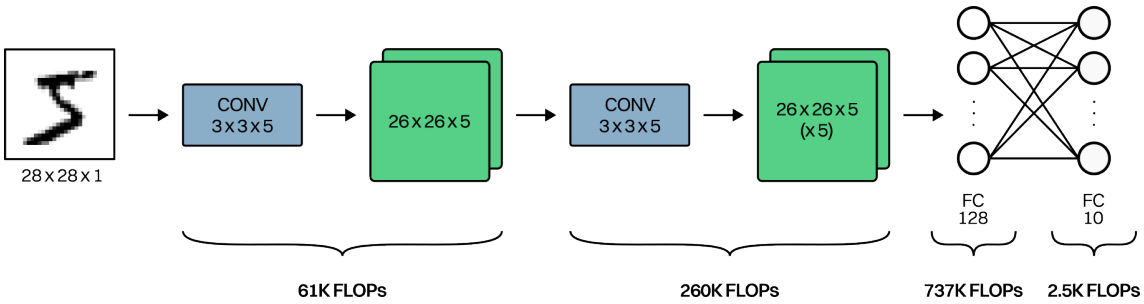

Fig. 3.3 : Nombre des FLOPs dans le modèle.

3.5 Choix du type d'élagage

Lors de la conception de notre méthode d'élagage, le choix entre l'élagage structuré et non structuré a été une considération essentielle. L'élagage structuré cible des motifs spécifiques, tels que des neurones entiers, des canaux ou des couches, tandis que l'élagage non structuré élimine des poids individuels sans tenir compte de leur emplacement dans le réseau. Chaque approche offre des avantages distincts et des compromis, influençant leur pertinence pour différentes situations d'élagage. Dans cet section, nous explorons les deux types d'élagage et nous sélectionnons un pour l'utiliser dans notre travail.

L'élagage structurée consiste à supprimer des unités ou blocs entiers du réseau de neurones, tels que des filtres, des canaux (plusieurs filtres) ou des couches entières, tout en conservant l'architecture globale du réseau. Cela permet d'avoir une compression plus importante de notre modèle car des structures entières sont supprimées, ce qui peut conduire à un déploiement très efficace sur des appareils avec des ressources de calculs et de stockage limitées. Cependant, l'élagage structuré peut affecter négativement la précision du réseau puisque certains poids importants peuvent être éliminés, et il pourrait nécessiter un ajustement fin pour récupérer les performances perdues.

D'autre part, l'élagage non structuré permet de supprimer des poids individuels ou des connexions du réseau, ce qui permet potentiellement de mieux conserver la précision. Toutefois, il ne conduit pas toujours à des gains significatifs en termes de mémoire et de temps d'inférence puisque le nombre de calculs reste le même (nous avons le même nombre de filtres et les poids éliminés sont seulement remplacés pare des zéros).

Le choix entre ces deux techniques dépend de facteurs tels que l'architecture spécifique du réseau, le matériel disponible et le compromis souhaité entre la compression et les performances. Dans notre cas, nous souhaitons déployer les modèles élagués dans des appareils avec des ressources limitées, c'est pourquoi d'élagage structuré est la meilleure approche à utiliser. Nous utilisons la norme L1 pour identifier et supprimer les canaux les moins importants. La norme L1 est simplement la somme des valeurs absolues des poids $|w_i|$ du canal (**??**). Pour chaque canal, la norme L1 des poids du canal est calculée et les canaux avec des valeurs plus faibles sont considérés comme moins importants et sont sélectionnés puis supprimés de la couche de convolution.

$$L1Norm = \sum |w_i| \tag{3.4}$$

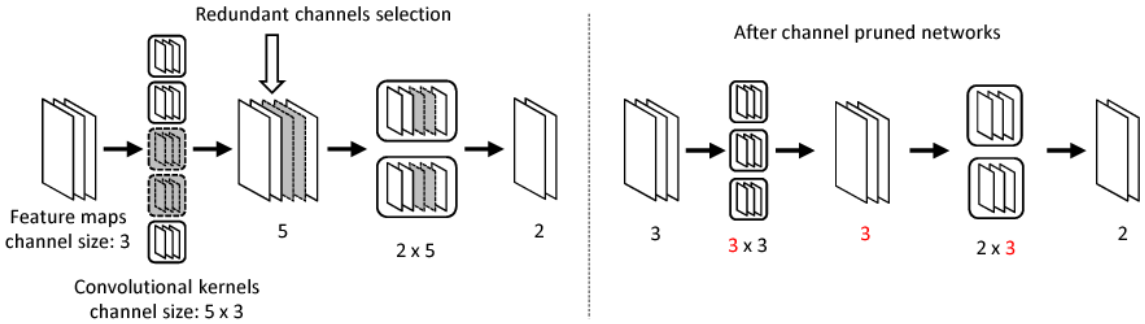

Fig. 3.4 : Illustration du mécanisme d'élagage des canaux [Yamamoto et al. 2018].

Après avoir retiré les canaux, le réseau peut subir un ajustement (ré-entraînement) pour rétablir les performances perdues. Cela peut impliquer l'entraînement du réseau élagué, en gardant les valeurs finales des poids après l'élagage, avec un taux d'apprentissage plus petit sur les exemples d'entraînement restants.

3.6 Implémentation de l'apprentissage par renforcement

Comme illustré sur la figure **??**, l'agent reçoit le plongement s_n de la couche L_n, puis génère un pourcentage d'élagage de cette couche en tant qu'action a_n. Ensuite, La couche L_n est élaguée avec le pourcentage a_n à l'aide de algorithme d'élagage de canaux. Après l'élagage de la couche L_n, l'agent passe à la couche suivante L_{n+1} et reçoit son plongement s_{n+1} en entrée. Après avoir élagué toutes les couches, la récompense est évaluée puis renvoyée à l'agent. Dans cette section, nous allons présenter en détail la théorie et l'implémentation de l'algorithme Deep Deterministic Policy Gradient (DDPG) dans notre méthode d'élagage automatique. Nous avons choisi d'utiliser DDPG au lieu de PPO (qui est utilisé dans la méthode présentée par Douag et al. 2022) parceque DDPG a tendance à converger plus rapidement que PPO dans de nombreux cas. Cette vitesse de convergence est attribuée en partie au fait que DDPG utilise des politiques déterministes et PPO utilise des politiques stochastiques, c'est-à-dire qu'à chaque étape, l'agent DDPG choisit une action spécifique sans aucune composante aléatoire, tandis que l'agent PPO choisit une

action avec une composante aléatoire. Cette stochasticité peut ralentir la convergence de l'agent PPO car il doit explorer différentes actions pour déterminer quelle est la meilleure.

3.6.1 Agent DDPG

Dans l'apprentissage par renforcement, les algorithmes de gradient de politique sont utilisés avec une fonction de politique stochastique $\pi(s)$. Cela signifie que, pour un état donné, il y aura une distribution de probabilité pour chaque action dans l'espace d'action. Dans le cas de l'algorithme DDPG (Deep Deterministic Policy Gradient), on utilise une politique déterministe $\mu(s)$ au lieu de la politique stochastique $\pi(s)$. Pour un état s donné, il y aura une décision déterministe $\mu(s)$ au lieu d'une distribution sur les actions.

Cet algorithme (DDPG) est un algorithme acteur-critique hors politique (off-policy) qui vise à résoudre des problèmes de contrôle continu, où les actions à prendre sont des valeurs continues plutôt que discrètes. Cet algorithm est une extension de l'apprentissage Q profond et l'algorithme DPG (Deterministic Policy Gradient) qui intègre des réseaux de neurones profonds pour mieux gérer les espaces d'actions et d'états complexes. L'apprentissage Q profond fonctionne dans un espace d'action discret et le DPG l'étend à l'espace d'action continu tout en apprenant une politique déterministe. Donc, Il apprend simultanément une fonction Q et une politique déterministe où il utilise des données hors politique et l'équation de Bellman (**??**) pour apprendre la fonction Q, et puis il utilise la fonction Q pour apprendre la politique.

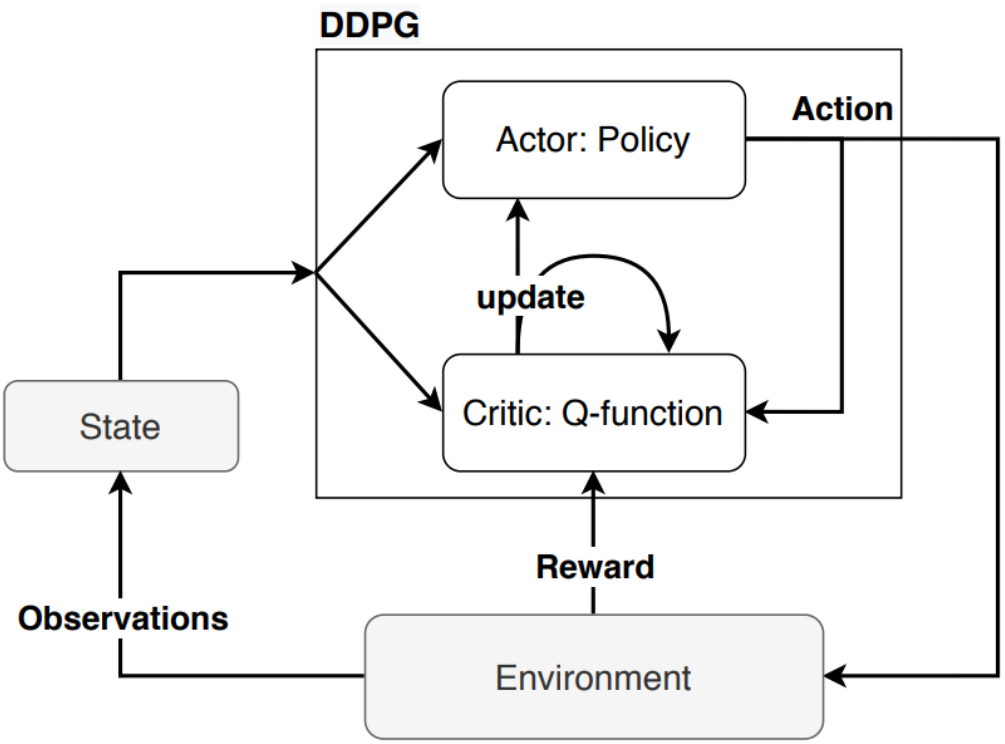

Fig. 3.5 : Schéma de présentation de l'algorithme DDPG [Handaoui et al. 2020].

L'une des difficultés dans l'apprentissage par renforcement est l'exploration de l'espace d'actions pour découvrir de bonnes stratégies. Dans DDPG, on utilise une politique

stochastique pour introduire de l'exploration où on ajoute un bruit additif dans les actions choisies par l'acteur. Dans notre travail, nous utilisons une distribution normale (politique gaussienne) pour le bruit additif (**??**).

$$\mu^{'}(s_n) \sim N(\mu(s_n|\Theta_n^{\mu}), \sigma^2, 0, 1) \tag{3.5}$$

Pendant l'exploitation, le bruit σ est initialisé à 0,5 et diminué de façon exponentielle après chaque épisode. Pour une exploration rapide, nous calculons la récompense sans réglage fin pour avoir une bonne approximation de la précision après le réglage fin.

Politique gaussienne

Pour permettre à l'agent de découvrir de nouvelles stratégies potentiellement meilleures, nous allons ajouter du bruit gaussien aux actions choisies par le réseau d'acteur. Ce bruit, également appelé distribution normale ou bruit blanc, est une distribution de probabilité caractérisée par sa moyenne μ et son écart type σ. Le bruit ajouté à chaque action est tiré de cette distribution et il permet d'explorer différentes parties de l'espace des actions. La formule **??** représente la fonction de la distribution gaussienne.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{3.6}$$

Le degré d'exploration peut être contrôlé en ajustant les paramètres de la distribution gaussienne. Nous initialisons l'écart type à une valeur plus élevée (0,5 dans notre cas), ce qui entraîne plus d'exploration, car le bruit ajouté aux actions devient plus important et imprévisible. Après chaque épisode, nous allons diminuer sa valeur de façon exponentielle. Pour la moyenne μ, elle peut être fixée à zéro ou à une petite valeur. Elle n'affecte pas significativement l'exploration, car elle décale le bruit seulement sans altérer son caractère aléatoire.

Ce bruit d'exploration équilibre le compromis entre l'exploration et l'exploitation. Au début, lorsque l'agent apprend, l'exploration est importante pour découvrir différentes actions. À mesure que l'entraînement progresse, le bruit sera progressivement réduit pour se concentrer davantage sur l'exploitation de la politique apprise. Pour ajouter ce bruit aux actions, nous utilisons un générateur de nombres aléatoires pour tirer des échantillons d'une distribution gaussienne

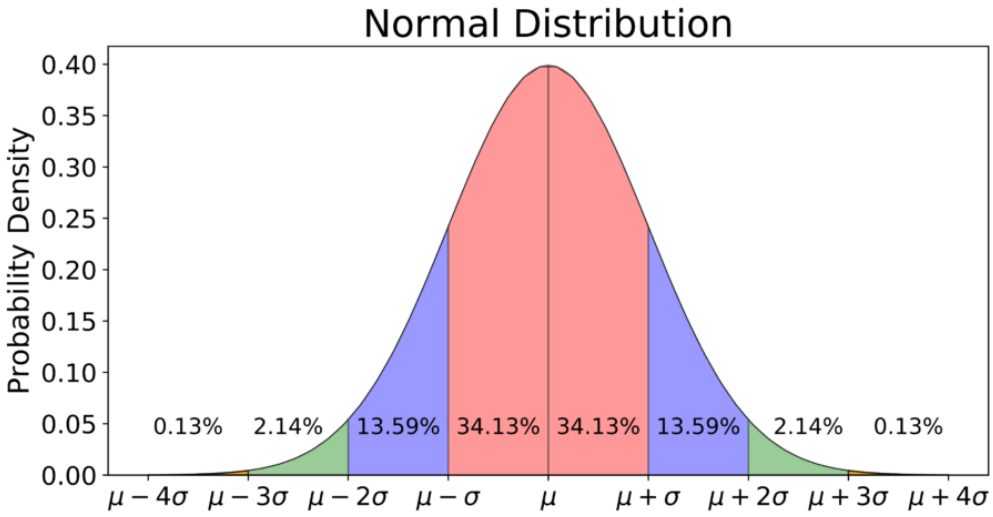

FIG. 3.6 : Courbe de la distribution gaussienne [YOSEPH et al. 2019].

3.6.2 Équations Clés

Valeur cible (Target Value) pour le Critique

Le réseau de critique a pour objectif de minimiser l'écart entre la valeur prédite et la récompense réelle et estimer la récompense cumulative attendue pour une paire état-action donnée. La valeur cible y_i pour le réseau de critique, qui est un élément crucial dans la mise à jour du réseau de critique, est calculée en fonction de la récompense immédiate r_i reçue à l'instant i de la valeur estimée de la prochaine paire état-action.

$$y_i = r_i - b + \gamma \cdot Q(s_{i+1}, \mu(s_{i+1})|\Theta^Q) \tag{3.7}$$

où $Q(s_{i+1}, \mu(s_{i+1})$ représente la récompense cumulée estimée pour le prochain état s_{i+1} et l'action proposée par le réseau d'acteurs cibles μ, γ représente le facteur de remise (discount factor), b est la récompense de base et μ représente le réseau d'acteur cible qui donne l'action suggérée pour le prochain état. La récompense de base b est soustraite pour réduire la variance de l'estimation du gradient, qui est une moyenne exponentielle des récompenses précédentes, tandis que le facteur de remise γ est fixé à 1 pour éviter de donner la priorité aux récompenses à court terme.

Mise à jour du Critique

Le réseau de critique est mis à jour en minimisant la perte moyenne entre la valeur prédite $Q(s_i, a_i)$ et la valeur cible y_i. La perte L est calculé pour chaque pas de temps, et le but est d'ajuster les paramètres du réseau critique pour minimiser cette perte.

$$L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i|\Theta^Q))^2 \tag{3.8}$$

Cette mise à jour aide le réseau critique à mieux se rapprocher de la véritable récompense cumulée attendue.

Mise à jour de l'Acteur

L'objectif de l'acteur est d'apprendre une politique qui maximise la récompense cumulée attendue telle qu'estimée par le réseau critique. En d'autres terms, il s'agit de maximiser les valeurs prédites du critique pour les actions prises. La mise à jour du réseau d'acteurs consiste à calculer le gradient de la récompense cumulée attendue par rapport aux paramètres de l'acteur Θ_μ. Ce gradient est approximé comme suit :

$$\nabla_{\Theta_\mu} J \approx \mathbb{E}_{s_i}[\nabla_a Q(s_i, a)|_{a=\mu(s_i)} \nabla_{\Theta_\mu} \mu(s_i)] \tag{3.9}$$

où J est la fonction d'objectif de l'acteur, Θ_μ sont les paramètres du réseau d'acteur, $\mu(s_i)$ est l'action produite par le réseau d'acteur pour l'état s_i et $\nabla_a Q(s_i, a)$ représente le gradient de la valeur du critique par rapport à l'action.

La mise à jour des paramètres Θ_μ de l'acteur est effectuée en utilisant ce gradient pour déplacer la politique dans la direction qui augmente la récompense cumulative attendue. La règle de mise à jour des paramètres de l'acteur est donnée par la formule **??**.

$$\Delta\Theta_\mu = \alpha \nabla_{\Theta_\mu} J \tag{3.10}$$

où α est le taux d'apprentissage.

3.6.3 Acteur-Critique

L'algorithme DDPG utilise deux types de réseaux de neurones profonds : un réseau d'acteur (Actor Network) et un réseau de critique (Critic Network).

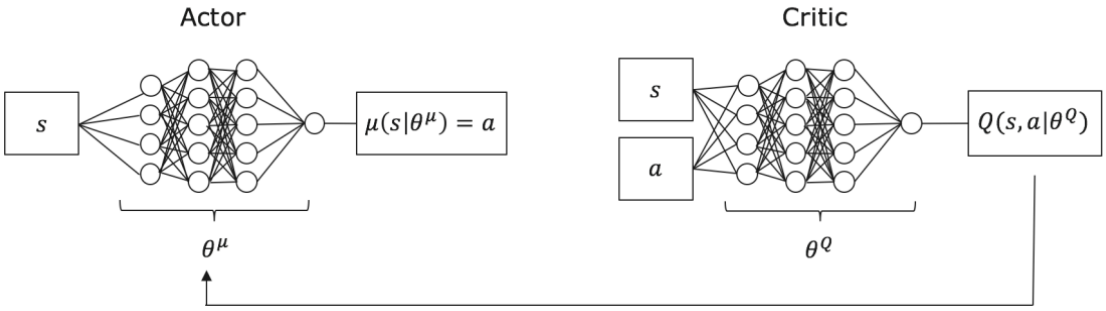

Fig. 3.7 : Structure de base de l'agent acteur-critique DDPG. L'acteur prend l'état s comme entrée et produit une action basée sur la politique déterministe μ comme résultat. Le critique prend à la fois l'état et l'action choisie par l'acteur comme entrée et fournit en sortie une valeur Q [Liessner et al. 2018].

Réseau d'acteur (Actor Network)

Le réseau d'acteur est responsable d'apprendre et d'améliorer la politique qui associe des états à des actions continues. Son objectif est de fournir des actions qui maximisent la récompense cumulative attendue sur le temps. Il prend un état actuel s_i (le plongement

de la couche i) en entrée et génère une action continue a_i (le pourcentage d'élagage de la couche i) en sortie qui correspond à l'action que l'agent devrait prendre dans l'état actuel. La caractéristique principale du réseau d'acteur dans l'algorithme DDPG est qu'il apprend une politique déterministe. Pour un état donné, il produit toujours la même action. Ce qui signifie qu'il apprend à produire des actions optimales pour chaque état, ce qui n'est pas le cas pour les politiques stochastiques où l'agent produit une distribution de probabilité sur les actions possibles.

Réseau de critique (Critic Network)

Le réseau de critique permet d'approximer la récompense cumulative attendue (aussi appelée fonction de valeur état-action) pour une paire état-action donnée. En d'autres termes, il évalue à quel point une action est bonne dans un état particulier selon la politique actuelle. Il guide le réseau d'acteur en évaluant la qualité des actions choisies. Il prend à la fois un état s_i (le plongement de la couche i) et une action a_i (le pourcentage d'élagage de la couche i) en entrée et prédit la récompense cumulative attendue $Q(s_i, a_i)$. Cette valeur prédite est fournie au réseau d'acteur pour mettre à jour les paramètres de ce dernier afin d'améliorer la politique, ce qui permet d'obtenir une politique plus performante au fil de l'apprentissage.

3.6.4 Protocole de recherche

Afin de déployer un réseau de neurones très profond sur des appareils avec des ressources matériel limitées, nous devons réduire significativement le nombre de FLOPs et la taille de ce réseau et pour rendre notre méthode plus rapide et plus efficace dans le processus d'élagage, nous allons limiter l'espace d'action de notre agent DDPG, c'est à dire que nous allons limiter le pourcentage d'élagage de chaque couche dans le réseau par la définition d'une borne supérieure a_{max} (on peut prendre $a_{max} = 0.8$). En limitant l'espace d'action, l'agent peut faire une exploration plus efficaces et il peut découvrir des configurations qui optimisent les compromis entre la précision et l'utilisation des ressources.

Pour permettre à l'agent de découvrir de nouvelles stratégies potentiellement meilleures, nous allons ajouter du bruit gaussien aux actions choisies par le réseau d'acteur. Cependant, lors de la mise en œuvre des limites de l'espace des actions pour l'agent DDPG, il est important de veiller à ce que le processus d'exploration de l'agent respecte ces limites. Les stratégies d'exploration telles que l'ajout de bruit aux actions doivent toujours produire des actions dans les bornes spécifiées.

L'algorithme suivant illustre le processus de prédiction du pourcentage d'élagage a_i de la couche L_i.

Algorithm 1 : Prédiction du pourcentage d'élagage a_i de la couche L_i

```
   /* Initialisation de la taille du modèle élagué                          */
1  if i = 0 then
2      T_elague ← 0
   /* Calcul de l'action                                                    */
3  a_i ← μ'(s_i)
   /* Limitation de l'action avec le pourcentage d'élagage maximum          */
4  a_i ← min(a_i, a_max)
   /* Calcul de la taille du modèle                                         */
5  T_total ← Σ_k T_k
   /* Calcul de la taille des couches suivantes                             */
6  T_suivant ← Σ_{k=i+1} T_k
   /* Calcul du nombre de paramètres à réduire dans la couche L_i si toutes les
      couches suivantes sont élaguées avec le pourcentage d'élagage maximal. α
      représente le pourcentage d'élagage cible du modèle.                  */
7  T_cible ← α · T_total − a_max.T_suivant − T_elague
   /* Limitation de l'action si elle est trop petite pour atteindre la réduction
      de taille souhaitée                                                   */
8  a_i ← max(a_i, T_cible/T_i)
   /* Mise à jour de la taille du modèle élagué                             */
9  T_elague ← T_elague + a_i · T_i
10 return a_i
```

3.6.5 Mémoire de l'agent

Dans l'algorithme DDPG, l'agent stocke les expériences passées dans une mémoire sous forme de transitions. Chaque transition dans une épisode est de la forme (s_i, a_i, R, s_{i+1}), où s_i représente l'état actuel (le plongement de la couche actuelle), a_i et a_{i+1} sont les pourcentages d'élagage de la couche actuelle et la couche suivante respectivement et R est la récompense après l'élagage du réseau. L'agent utilise ensuite ces transitions pour mettre à jour les paramètres des réseaux de neurones d'acteur et de critique.

L'utilisation de cette mémoire permet d'améliorer la stabilité de l'apprentissage et la convergence des réseaux en permettant de mélanger les transitions provenant de différentes étapes temporelles afin d'éviter les problèmes de corrélation temporelle, où les transitions consécutives sont fortement liées. De plus, la mémoire permet une meilleure généralisation de l'apprentissage en stockant et en réutilisant des transitions passées. L'agent donc peut acquérir une compréhension plus large de l'environnement et développer des stratégies qui fonctionnent bien dans divers scénarios.

3.6.6 Processus d'entraînement de l'agent

Dans cette section, nous expliquons le processus d'entraînement de l'agent DDPG. Comme illustré dans la figure **??**, ce processus est composé principalement de cinq étapes principales.

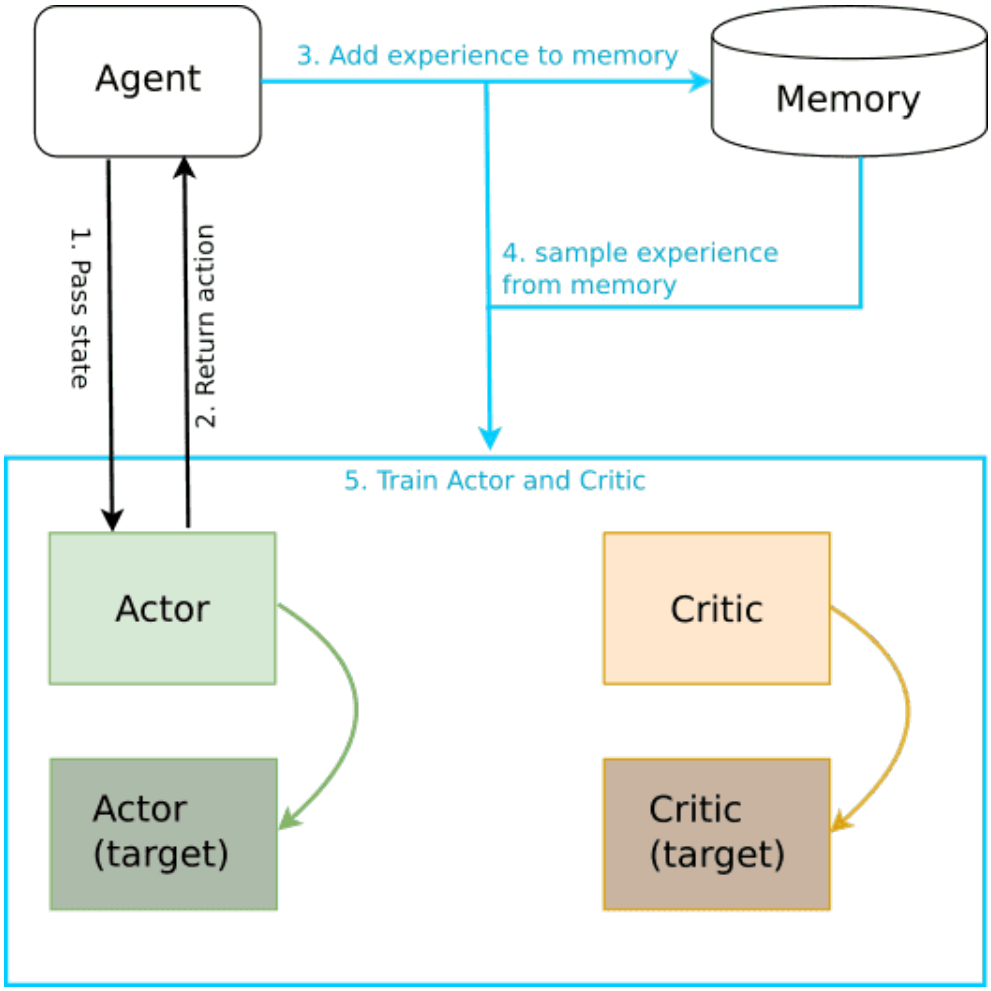

FIG. 3.8 : Schéma représentant le processus d'entraînement de l'agent DDPG. L'agent est entraîné pour un nombre fixe d'épisodes et, dans chaque épisode, un nombre fixe de pas de temps (timesteps).

1. **Passage de l'état à l'acteur** : L'agent reçoit un état s_i de l'environnement, puis il passe cet état au réseau d'acteur (μ). Le réseau d'acteur génère une action a_i en fonction de l'état s_i. Cette action est choisie pour maximiser la valeur estimée $Q(s_i, a_i$, qui représente la somme des récompenses futures prévues.

2. **Retour de l'action à l'agent** : L'agent reçoit l'action générée a_i du réseau d'acteur, puis il l'envoie à l'environnement. Ce dernier évolue à l'étape i et fournit une récompense et un nouvel état s_{i+1} à l'agent.

3. **Ajout des expériences à la mémoire** : L'agent crée une expérience en combinant l'état s_i, l'action a_i, la récompense et le nouvel état s_{i+1} et il l'ajoute à la mémoire.

4. **Échantillonnage des expériences de la mémoire** : l'agent échantillonne un lot (batch) d'expériences de la mémoire. Ces expériences échantillonnées forment un ensemble varié de données sur lequel l'agent va apprendre.

5. **Entraînement de l'acteur et du critique** : L'agent utilise les expériences échantillonnées dans l'etape précédente pour mettre à jour le réseau de critique (Q) en

minimisant la différence entre les valeurs prédites et les valeurs cibles basées sur l'équation de Bellman et il calcule le gradient à partir du réseau de critique pour l'utiliser pour mettre à jour le réseau d'acteur (μ). Cette mise à jour du réseau d'acteur vise à maximiser les valeurs Q prédites.

3.7 Réglage fin après l'élagage

A la fin de l'exécution de l'algorithme DDPG, nous aurons un modèle élagué qui va avoir une baisse de performance en raison de la perte d'informations qui peut être causée pendant l'élagage puisque nous utilisons l'élagage structuré. On ré-entraîne donc ce modèle afin d'améliorer sa précision pour atteindre une précision proche de celle du modèle original. C'est là qu'interviennent des stratégies clés telles que l'initialisation aléatoire, le rewinding, et le réglage fin (fine-tuning). Ces stratégies sont utilisées pour mettre à jour les paramètres du réseau avant l'entraînement. Dans l'initialisation aléatoire, on affecte des valeurs aléatoires aux paramètres du réseau élagué, tandis que dans le cas du rewiniding, on sauvegarde les paramètres du réseau original avant le premier entraînement pour les utiliser pour initialiser le réseau élagué. Dans le fine tuning, on initialise le réseau élagué par les valeurs des paramètres pré-élagage. La meilleure stratégie pour avoir la plus grande précision est le rewinding, mais elle ne converge pas rapidement [Douag et al. 2022]. C'est pourquoi nous avons choisi d'utiliser le fine tuning après l'élagage car il permet d'avoir une convergence plus rapide que les deux autres stratégies [Douag et al. 2022].

Dans cette étape, nous allons ré-entraîner le réseau élagué sur les mêmes données d'entraînement tout en maintenant les poids non supprimés fixés (nous gardons les valeurs des poids résultantes de l'élagage). Cela permet au modèle de réapprendre les relations importantes entre les caractéristiques des données et les étiquettes cibles, en se concentrant sur les parties restantes du réseau. Cependant, nous devons maintenir les poids du réseau qui ont été préservés pendant l'élagage fixés pour garantir que le réseau ne perd pas les connaissances apprises initialement.

Dans notre méthode, nous commençons le ré-entraînement avec un taux d'apprentissage plus faible que celui utilisé dans l'entraînement initial afin d'éviter des mises à jour qui pourraient perturber les poids restants.

3.8 Conclusion

Dans ce chapitre, nous avons présenté les différentes étapes suivies pour concevoir notre propre méthode automatique d'élagage. Nous avons commencé par une vue global de la méthode, où nous l'avons décomposée en quatre étapes principales : **initialisation**, **entraînement**, **élagage** et **réglage fin**. Dans la partie d'élagage, nous avons utilisé un agent DDPG qui prend en entrée un plongement d'une couche et fournit en sortie le pourcentage d'élagage de cette couche. Après l'élagage, nous avons fait un réglage fin pour améliorer la précision du modèle élagué. Nous aovns également présenté en détails la construction des plongements des couches, les métriques utilisées pour choisir les pourcentages d'élagage, le type d'élagage utilisée (élagage des couches) et l'implémentation de

l'algorithme DDPG.

Dans le chapitre suivant, nous allons évaluer et tester notre méthode sur les modèles VGG-19 et ResNet-34. Nous allons présenter d'abord les technologies et outils utilisés pour la conception, puis nous allons comparer les résultats d'élagage de notre méthode avec les résultats d'élagage de quelques méthodes automatique qui effectuent le même type d'élagage que notre méthode (élagage des canaux).

Chapitre 4

Réalisation et tests

4.1 Introduction

Après avoir exposé en détail notre méthode automatique d'élagage dans le chapitre précèdent, nous discuterons maintenant sur les technologies et outils qui ont été choisis pour matérialiser cette approche et garantir son déploiement efficace. De ce fait, il est essentiel d'examiner en détail les langages de programmation, les bibliothèques et les frameworks qui ont été sélectionnés pour concrétiser notre approche. Nous offrons également un aperçu des stratégies de test que nous avons employées pour évaluer la performance de notre solution. Ces tests ont été établis pour mesurer les performances des modèles VGG-19 et ResNet-34 et la précision de chacun.

4.2 Modèles et jeu de données utilisés

Notre méthode d'élagage a été conçue pour élaguer les réseaux de neurones comportant seulement deux types de couches : les couches de convolution et les couches entièrement connectées. C'est pourquoi nous avons choisi d'utiliser des modèles tels que VGG-19 et ResNet-34. Dans cette section, nous parlerons de l'architecture et de l'organisation de ces deux modèles ainsi que le jeu de données CIFAR-10 qui est utilisé pour les entraîner.

4.2.1 CIFAR-10

CIFAR-10 [1] est un jeu de données qui est souvent utilisé dans le domaine de la vision par ordinateur. Il se compose d'un total de 60 000 images en couleur de 32x32 pixels chacune, réparties en 10 classes différentes, avec 6 000 images par classe [Krizhevsky et al. 2009]. Ces 10 classes d'images comprennent : des avions, des automobiles, des oiseaux, des chats, des cerfs, des chiens, des grenouilles, des chevaux, des bateaux et des camions (**??**).

[1] acronyme qui représente le "Canadian Institute for Advanced Research" (Institut canadien de recherches avancées)

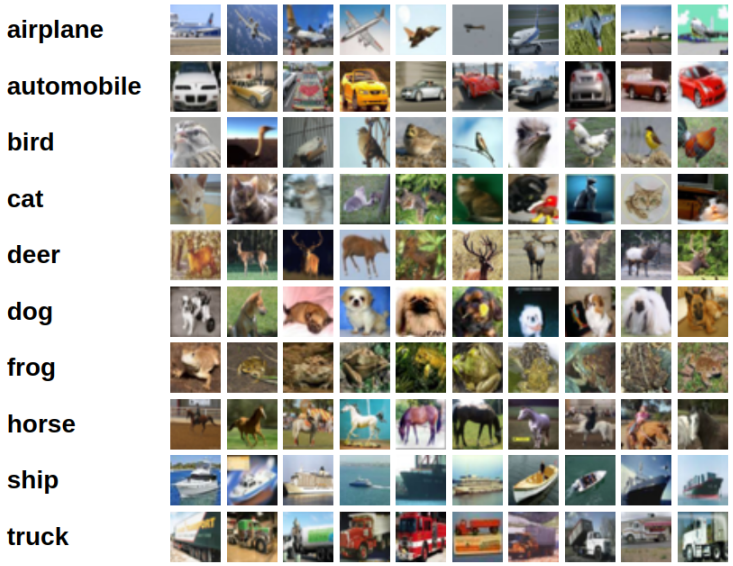

FIG. 4.1 : Les classes du jeu de données CIFAR-10.

Le dataset est divisé en deux ensemble : un ensemble d'entraînement et un ensemble de test. L'ensemble d'entraînement est composé de 50 000 images, tandis que l'ensemble de test est composé de 10 000 images. Dans ce dataset, les images sont de résolution relativement basse et présentent une variabilité dans les poses, l'éclairage, les arrière-plans et les détails, ce qui rend le dataset proche des défis du monde réel. Il est souvent utilisé pour l'entraînement, l'évaluation et la comparaison des performances des algorithmes de classification d'images et des réseaux de neurones convolutionnels (CNN), et il permet de reconnaître leur capacité à généraliser à partir d'un ensemble d'apprentissage limité [KRIZHEVSKY et al. 2009].

4.2.2 VGG-19

VGG-19 [2] est un réseau de neurones convolutif (CNN) très profond qui possède plus de 19,6 milliards de FLOPs et est utilisé fréquemment dans le domaine de la vision par ordinateur. Il est composé de 19 couches, comprenant 16 couches de convolution et 3 couches entièrement connectées, et les images en entrée sont généralement de taille 224x224 pixels. Le réseau commence par des couches de convolutions successives, qui sont suivies de couches de pooling (de type max pooling). Ces dernières permettent de réduire la dimension spatiale de la sortie en ne conservant que les informations les plus importantes [SIMONYAN et al. 2014].

Chaque couche de convolution est composée de plusieurs filtres (ou noyaux) 3x3 et un padding (stride) de 1 pour préserver les dimensions. Les filtres possèdent 64, 128, 256, 512 et 512 canaux respectivement. Le nombre de canaux augmente à mesure que l'on progresse dans le réseau. tandis que dans les couches de pooling, on a des fenêtres de 2x2 et un décalage de 2 pour réduire la résolution spatiale [SIMONYAN et al. 2014].

Une fois que les données sont réduites en termes de dimensions spatiales, elles sont

[2] acronyme de "Visual Geometry Group" (famille des modèles développés par l'Université d'Oxford)

passées à travers plusieurs couches entièrement connectées afin d'effectuer la classification, en associant les caractéristiques apprises aux classes d'objets. La dernière couche de sortie possède le même nombre de neurones que de classes dans le jeu de données. Elle utilise une fonction d'activation softmax pour obtenir des probabilités normalisées pour chaque classe. Dans les couches de convolution, les fonctions d'activation utilisées sont des fonctions ReLU (Rectified Linear Unit) [Simonyan et al. 2014].

ConvNet Confguration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input(224 x 224 RGB image)					
conv 3-64	conv 3-64 **LRN**	conv 3-64 **conv 3-64**	conv 3-64 conv 3-64	conv 3-64 conv 3-64	conv 3-64 conv 3-64
maxpool					
conv 3-128	conv 3-128	conv 3-128 **conv 3-128**	conv 3-128 conv 3-128	conv 3-128 conv 3-128	conv 3-128 conv 3-128
maxpool					
conv 3-256 conv 3-256	conv 3-256 conv 3-256	conv 3-256 conv 3-256	conv 3-256 conv 3-256 **conv 1-256**	conv 3-256 conv 3-256 **conv 3-256**	conv 3-256 conv 3-256 conv 3-256 **conv 3-256**
maxpool					
conv 3-512 conv 3-512	conv 3-512 conv 3-512	conv 3-512 conv 3-512	conv 3-512 conv 3-512 **conv 1-512**	conv 3-512 conv 3-512 **conv 3-512**	conv 3-512 conv 3-512 conv 3-512 **conv 3-512**
maxpool					
conv 3-512 conv 3-512	conv 3-512 conv 3-512	conv 3-512 conv 3-512	conv 3-512 conv 3-512 **conv 1-512**	conv 3-512 conv 3-512 **conv 3-512**	conv 3-512 conv 3-512 conv 3-512 **conv 3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Tab. 4.1 : Les configurations des réseaux VGG (illustrées dans des colonnes selon leurs profondeurs). La colonne E représente la configuration du modèle VGG-19 utilisé [Simonyan et al. 2014].

4.2.3 ResNet-34

ResNet-34 est un réseau de neurones résiduel profond, composé de 34 couches et possédant plus de 3,6 milliards de FLOPs. Ce type de réseau est caractérisé par l'utilisation de blocs résiduels, en introduisant des connexions résiduelles, également appelées connexions "skip", qui sautent une ou plusieurs couches. Cette caractéristique permet d'exploiter les avantages de la profondeur tout en évitant les problèmes de dégradation de performance qui surviennent lorsque les réseaux deviennent plus profonds [He et al. 2016].

Le bloc résiduel de base dans ResNet-34 comprend deux couches de convolution 3x3 accompagnées de fonctions d'activation ReLU, et intègre une connexion résiduelle qui agit comme un raccourci autour des couches de convolution, en ajoutant la sortie de la première couche de convolution à la sortie de la deuxième couche.

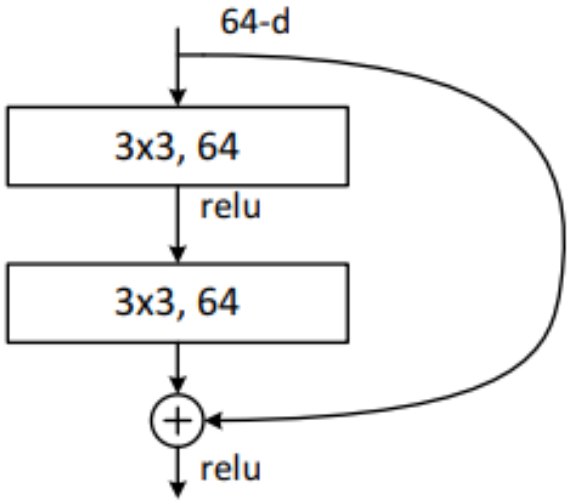

Fig. 4.2 : Un bloc résiduel de base comme sur la figure **??** pour ResNet-34 [He et al. 2016].

La fin du réseau ResNet-34 comprend une couche de classification entièrement connectée avec autant de neurones que de classes dans le jeu de données. Une fonction d'activation softmax est généralement utilisée pour obtenir les probabilités de classe normalisées.

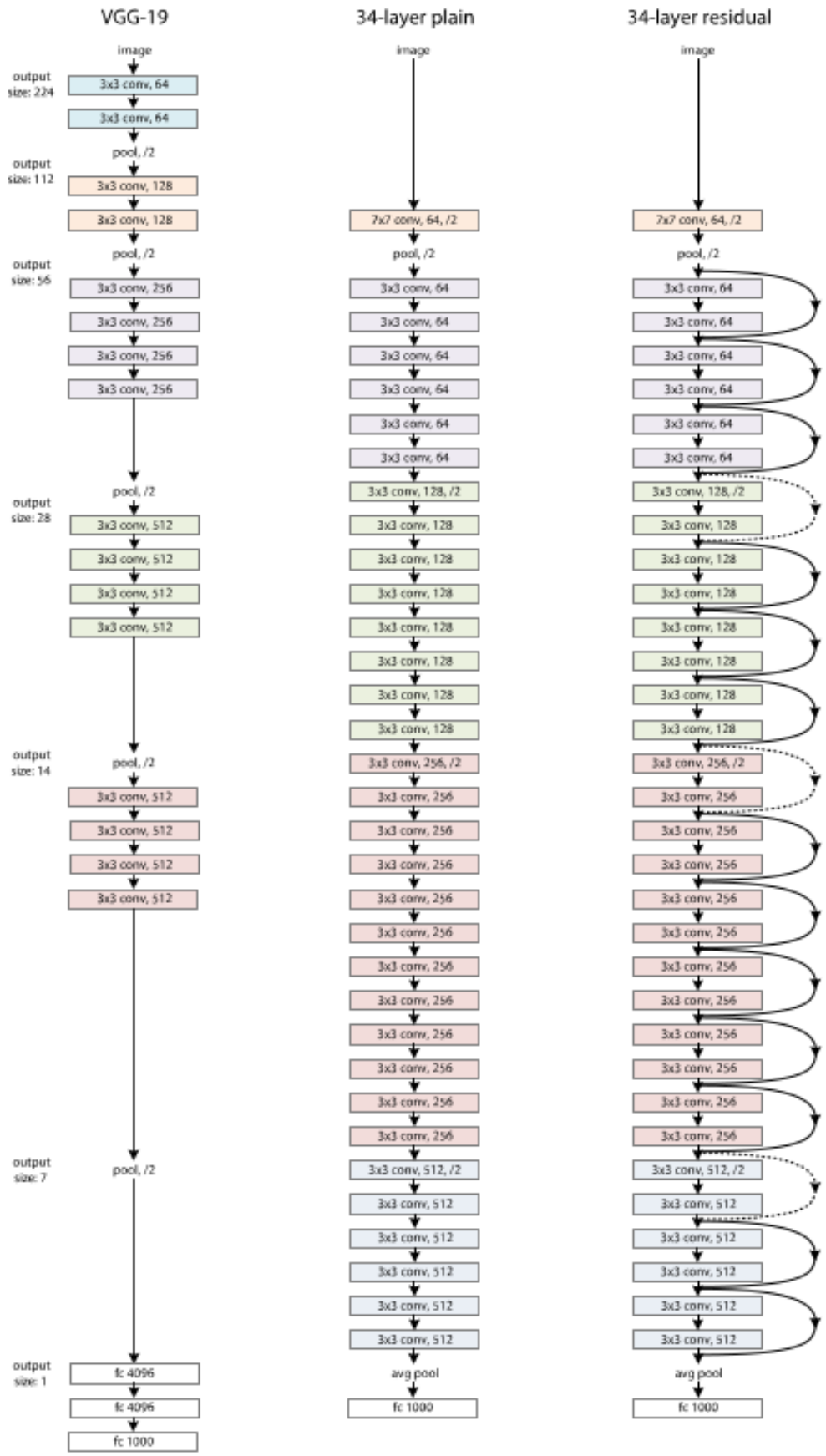

Fig. 4.3 : Les architectures des modèles utilisés. A gauche : le modèle VGG-19. Au milieu : un réseau simple de 34 couches. À droite : le modèle ResNet-34 [He et al. 2016].

4.3 Technologies utilisées

4.3.1 Python

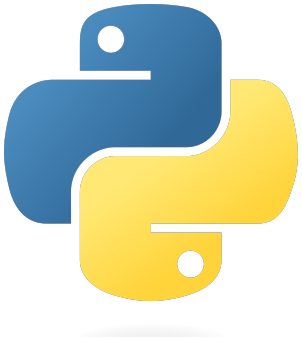

Fig. 4.4 : Python.

Python joue un rôle important dans le domaine de l'intelligence artificielle (IA) grâce à sa polyvalence, sa simplicité et sa richesse en bibliothèques spécialisées. En tant que langage de programmation, Python est devenu le premier choix pour de nombreux chercheurs, ingénieurs et développeurs travaillant dans le domaine de l'IA. D'une part, Python est connu pour sa syntaxe claire et lisible, ce qui permet aux nouveaux arrivants dans le domaine de l'IA de se familiariser rapidement avec les concepts de base. D'autre part, Python offre plusieurs bibliothèques et frameworks spécialisés dans l'IA, tels que TensorFlow, Keras, PyTorch et Scikit-learn. Ces bibliothèques sont souvent utilisées pour le développement de réseaux de neurones, d'algorithmes d'apprentissage automatique et d'autres techniques d'IA. Finalement, Python est un langage polyvalent qui permet aux développeurs de créer et tester différentes approches rapidement.

4.4 Bibliothèque utilisées

Python offre une riche collection de bibliothèques spécialisées dans plusieurs domaines, en particulier l'IA. Ces bibliothèques offrent des outils et des frameworks puissants pour développer des modèles d'apprentissage automatique, des réseaux de neurones, et bien plus encore. Dans cette section, nous allons explorer les bibliothèques que nous avons utilisé pour implémenter et tester notre méthode.

4.4.1 NumPy

Fig. 4.5 : NumPy.

NumPy [3] est une bibliothèque open source qui offre de nombreuses fonctionnalités pour le calcul numérique en Python. Elle offre un support puissant pour la manipulation de tableaux multidimensionnels, ainsi que pour l'exécution de calculs mathématiques complexes sur ces tableaux. Ces tableaux multidimensionnels permettent de stocker et de manipuler efficacement des données numériques sous forme de matrices et de vecteurs. Elle fournit également des fonctions mathématiques de base, des opérations d'algèbre linéaire, des opérations sur les tableaux, des fonctions statistiques et bien plus encore. Elle est largement utilisé en IA pour le traitement et la manipulation de données, la préparation de jeux de données, ainsi que pour la mise en œuvre d'algorithmes d'apprentissage automatique et de réseaux de neurones. La performance élevée de NumPy en calcul numérique en fait un bon choix pour les tâches intensives en termes de calcul.

4.4.2 Matplotlib

Fig. 4.6 : Matplotlib.

Matplotlib est une bibliothèque de visualisation en Python qui aide à créer des graphiques et des visualisations de données de manière interactive et statique. Cette bibliothèque offre un large éventail d'outils pour générer des graphiques de haute qualité à partir de données numériques. Son objectif est de permettre aux utilisateurs de représenter visuellement des données complexes de manière claire et compréhensible. Pour cela, elle propose une grande variété de types de graphiques, tels que les graphiques linéaires, les graphiques en barres, les graphiques à secteurs, les graphiques de dispersion, les graphiques 3D, etc. Elle permet également de personnaliser presque tous les aspects des

[3] acronyme de "Numerical Python"

graphiques, y compris les étiquettes, les couleurs, les styles de ligne, les titres, les axes et les légendes. L'utilisation de Matplotlib est essentielle dans l'analyse de données, la science des données et la recherche en général. En intelligence artificielle et en apprentissage automatique, Matplotlib est souvent employé pour visualiser les performances des modèles, les distributions de données, les tendances, les caractéristiques importantes, les matrices de confusion, les courbes d'apprentissage, etc.

4.4.3 Pytorch

Fig. 4.7 : Pytorch.

PyTorch est une bibliothèque open-source d'apprentissage automatique et d'intelligence artificielle en Python, développée principalement par Facebook's AI Research lab (FAIR). Elle repose sur un concept fondamental appelé "tenseur", qui est une structure de données multidimensionnelle similaire aux tableaux NumPy et elle est conçue pour faciliter le développement et la mise en œuvre de modèles de réseaux de neurones profonds. PyTorch est surtout distinguée par sa prise en charge des calculs automatiques de gradients, ce qui signifie qu'il est possible de définir des opérations mathématiques sur les tenseurs et que PyTorch peut automatiquement calculer les gradients de ces opérations qui sont nécessaires pour ajuster les poids dans les réseaux de neurones et ainsi minimiser une fonction de perte. Elle est utilisé pour la conception des modèles de réseaux de neurones profonds, y compris les réseaux de neurones convolutionnels (CNN), les réseaux de neurones récurrents (RNN), les transformeurs, etc.

4.4.4 Torchvision

Fig. 4.8 : Torchvision.

Torchvision est une bibliothèque qui fait partie de PyTorch. Elle est spécifiquement conçue pour faciliter le chargement et la transformation de jeux de données d'images couramment utilisés dans le domaine de l'apprentissage automatique et de la vision par ordinateur. Elle offre des outils pour prétraiter les données d'image, créer des ensembles de données, appliquer des transformations aux images et charger des ensembles de données préexistants. Elle propose des classes pour charger facilement des ensembles de données standard tels que MNIST, CIFAR-10, ImageNet, etc. Elle permet également d'appliquer diverses transformations aux images, telles que le redimensionnement, le recadrage, la normalisation, les rotations, les miroirs, etc. qui sont utiles pour augmenter la variabilité des données.

4.4.5 NNI

Fig. 4.9 : NNI (Neural Network Intelligence).

NNI (Neural Network Intelligence) est une bibliothèque open-source développée par Microsoft Research. Elle fournit un ensemble d'outils et de bibliothèques pour faciliter l'exploration et l'optimisation des espaces d'hyperparamètres, ainsi que pour la recherche automatique d'architectures de modèles. Elle permet aux utilisateurs de définir un espace de recherche pour les hyperparamètres, tels que les taux d'apprentissage, les tailles de lot, les architectures de couches, etc. NNI exécute ensuite des expériences en utilisant différentes configurations d'hyperparamètres et rapporte les résultats, y compris les performances du modèle. Elle est très utile dans le domaine de l'apprentissage automatique, car elle simplifie et automatise le processus d'ajustement des hyperparamètres et d'exploration des architectures, ce qui peut considérablement accélérer le développement de modèles performants.

4.5 Outils utilisés

4.5.1 Google Colab

Fig. 4.10 : Google Colab.

Google Colab (abrégé de Colaboratory) est une plateforme de notebooks interactifs basée sur le cloud, développée par Google. Elle permet aux utilisateurs d'écrire, d'exécuter et de partager du code Python de manière collaborative, sans nécessiter de configuration ou d'installation. Elle propose des notebooks interactifs qui permettent d'insérer des cellules de code exécutable et des cellules de texte et chaque notebook Colab s'exécute dans un environnement virtuel où les utilisateurs peuvent accéder à la puissance de calcul des processeurs graphiques (GPU) et des unités de traitement tensoriel (TPU) pour accélérer l'entraînement de modèles d'apprentissage automatique. Elle propose également de nombreuses bibliothèques préinstallées, mais les utilisateurs peuvent également installer et utiliser des bibliothèques tierces via des commandes simples.

4.5.2 Google Drive

Fig. 4.11 : Google Drive.

Google Drive est un service de stockage en ligne développé par Google pour stocker, synchroniser et partager des fichiers et des dossiers sur le cloud. Il offre une variété de fonctionnalités telles que le stockage en ligne, la synchronisation multi-appareils, le partage de fichiers, la collaboration en temps réel, etc. De plus, il peut être utilisé avec Google Colab.

4.6 Tests et résultats

Cette dernière section présente une analyse des résultats obtenus après avoir tester notre méthode d'élagage. Nous effectuons nos expérimentations sur CIFAR-10 avec deux réseaux profonds classiques : VGG-19 et ResNet-34. Les résultats obtenus sont examinés et comparés avec les résultats des autres méthodes d'élagage, permettant ainsi de dégager des conclusions quant à la performance et l'efficacité de notre méthode. Cette section donc vise à présenter de manière claire et précise les découvertes issues de ces tests.

Le processus d'exécution des tests est décrit comme suit :

1. Entraînement du modèle jusqu'à la convergence
2. Élagage du modèle avec les pourcentages d'élagage fournis par l'agent DDPG
3. Réglage fin du modèle
4. Mesure de la précision, la taille, le nombre de paramètres, etc. du modèle élagué
5. Comparaison du modèle élagué avec le modèle original

On élague les canaux dont les poids ont la plus petite valeur absolue. Le pourcentage d'élagage maximum a_{max} est fixé pour les couches de convolution à 0,8 et pour la couche entièrement connectée à 0,98. Cette limite supérieure a_{max} est utilisée uniquement pour accélérer la recherche. Nous pouvons simplement prendre $a_{max} = 1$ et nous aurons des résultats similaires. Le réseau d'acteurs μ comporte deux couches cachées, chacune comportant 300 neurones et la couche de sortie finale est une couche sigmoïde pour délimiter les actions dans la plage (0, 1). Le réseau critique Q comportait également deux couches cachées, chacune comptant 300 unités. Nous entraînons le réseau avec 64 comme batch size. L'agent DDPG explore d'abord 100 épisodes avec un bruit constant $\sigma = 0.5$, puis exploite 300 épisodes avec un bruit σ qui décroît de manière exponentielle.

Pour évaluer la performance de notre méthode, nous l'avons comparée avec les deux méthodes d'élagage citées dans la première partie du rapport : ABCPruner [Lin et al. 2020] et CCPrune [Chen et al. 2021]. Ces deux méthodes sont des méthodes automatique qui utilisent le même type d'élagage (élagage des canaux). ABCPruner est une méthode basée sur l'algorithme de colonie d'abeilles artificielles (ABC). Dans cette méthode, la recherche du réseau élagué optimal est formulée comme un problème d'optimisation et l'algorithme ABC est utilisé pour le résoudre de manière automatique afin de réduire les interférences humaines. CCPrune (Collaborative Channel Pruning) est une autre méthode qui utilise aussi l'élagage des canaux. Cette méthode introduit d'abord la régularisation sur les poids des couches de convolution et les facteurs d'échelle de la couche BN (Batch Normalization) respectivement, puis elle combine les poids de la couche de convolution et le facteur d'échelle de la couche BN pour évaluer l'importance du canal.

4.6.1 VGG-19

Le tableau suivant représente les résultats après l'application de notre méthode sur VGG-19 avec une comparaison aux deux autres méthodes d'élagage : ABCPruner et CCPrune.

Méthode	Précision (%)	FLOPs (%)	Paramètres (%)	Taille (MB)
Modèle original	93.71	-	-	548
ABCPruner	93.08	73.68	88.68	62.6
CCPrune	**93.78**	48.92	86.82	77.2
Notre méthode	90.6	91.6	90.2	**52.7**

Tab. 4.2 : Résultats d'élagage de VGG-19 sur CIFAR-10. La deuxième colonne indique la précision du modèle. Les troisième et quatrième colonnes indiquent le taux d'élagage des FLOPs et le taux d'élagage des paramètres. La dernière colonne indique la taille du modèle.

Les résultats dans le tableau **??** montrent que les techniques ABCPruner et CCPrune atteignent toujours une précision très proche du réseau original qui est aussi meilleure que celle de notre méthode. Cependant, notre méthode dépasse ces techniques quand nous parlons de la taille du modèle. Nous remarquons qu'avec notre méthode, nous pouvons d'avoir une grande réduction dans la taille du modèle (la taille est 10 fois moins que le modèle original) et dans le nombre de FLOPs et paramètres à cause des pourcentages d'élagage élevés. Cette réduction du nombre de FLOPs signifie que notre méthode effectue moins de calculs que les deux autres méthodes. Nous pouvons donc avoir le meilleur temps d'inférence avec notre méthode. Nous pouvons aussi remarquer que l'utilisation de l'élagage structuré engendre une petite dégradation de la précision des modèles élagués par rapport au modèle original. Cette dégradation est causée par la suppression de canaux entiers, ce qui peut causer une suppression de certains paramètres importants qui peuvent se trouver dans l'un des canaux éliminées.

La figure **??** montre les statistiques des canaux restants dans les couche de convolution dans le modèle VGG-19. Sur cette figure, Nous pouvons clairement comprendre la structure du réseau après l'élagage.

Dans VGG-19, 90% des poids sont dans les couches entièrement connectées. Dans notre méthode, nous avons utilisé l'élagage de canaux entiers en couches de convolution et cela a un effet secondaire intéressant en réduisant également la mémoire. Comme observé dans Molchanov et al. 2016, plus la couche est profonde, plus elle sera élaguée. Cela signifie que la dernière couche de convolution sera beaucoup élaguée et que de nombreux neurones de la couche entièrement connectée qui la suit seront également supprimés.

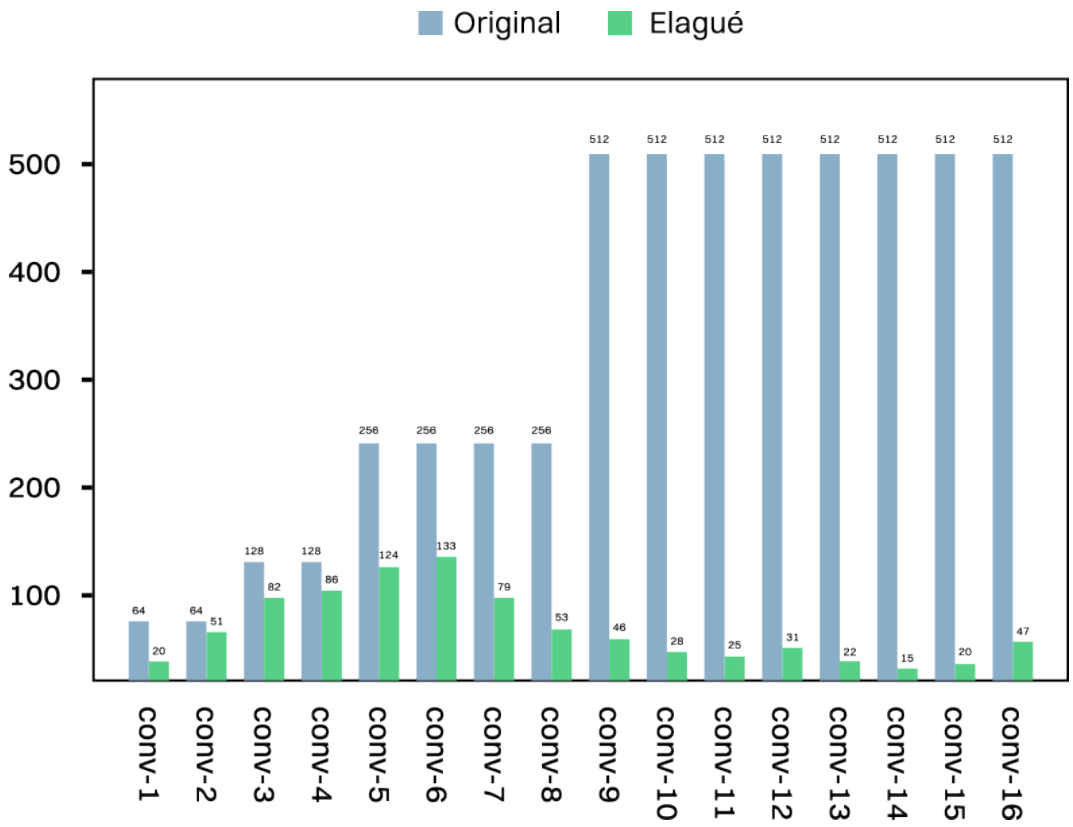

Fig. 4.12 : Statistiques des canaux restants dans les couches de convolution de VGG-19.

4.6.2 ResNet-34

Le tableau suivant représente les résultats après l'application de notre méthode sur ResNet-34 et une comparaison avec les deux autres méthodes d'élagage utilisées : ABC-Pruner et CCPrune.

Méthode	Précision (%)	FLOPs (%)	Paramètres (%)	Taille (MB)
Modèle original	91.45	-	-	81.4
ABCPruner	89.69	58.97	51.76	39.2
CCPrune	**90.76**	57.55	34.78	53.0
Notre méthode	86.21	58.09	67.63	**26.5**

Tab. 4.3 : Résultats d'élagage de ResNet-34 sur CIFAR-10. La deuxième colonne indique la précision du modèle. Les troisième et quatrième colonnes indiquent le taux d'élagage des FLOPs et le taux d'élagage des paramètres. La dernière colonne indique la taille du modèle.

Les résultats dans le tableau **??** sont les même que ceux du modèle VGG-19. Ils montrent toujours que la précision dans les techniques ABCPruner et CCPrune dépasse la précision dans notre méthode. Cependant, notre méthode dépasse ces deux techniques dans la compression de la taille du modèle et dans le nombre de FLOPs, ce qui nous donnera un meilleur temps d'inférence. Nous remarquons aussi que la taille du modèle élagué avec notre méthode est presque 3 fois moins que le modèle original, ainsi que le nombre de FLOPs et paramètres (les résultats sont résumés dans la figure **??**). Nous avons

aussi vu que l'élagage structuré réduit légèrement la précision des modèles élagués par rapport au modèle original et la cause de cette réduction est la même cause mentionnée dans l'interprétation de l'élagage structuré pour le modèle VGG-19.

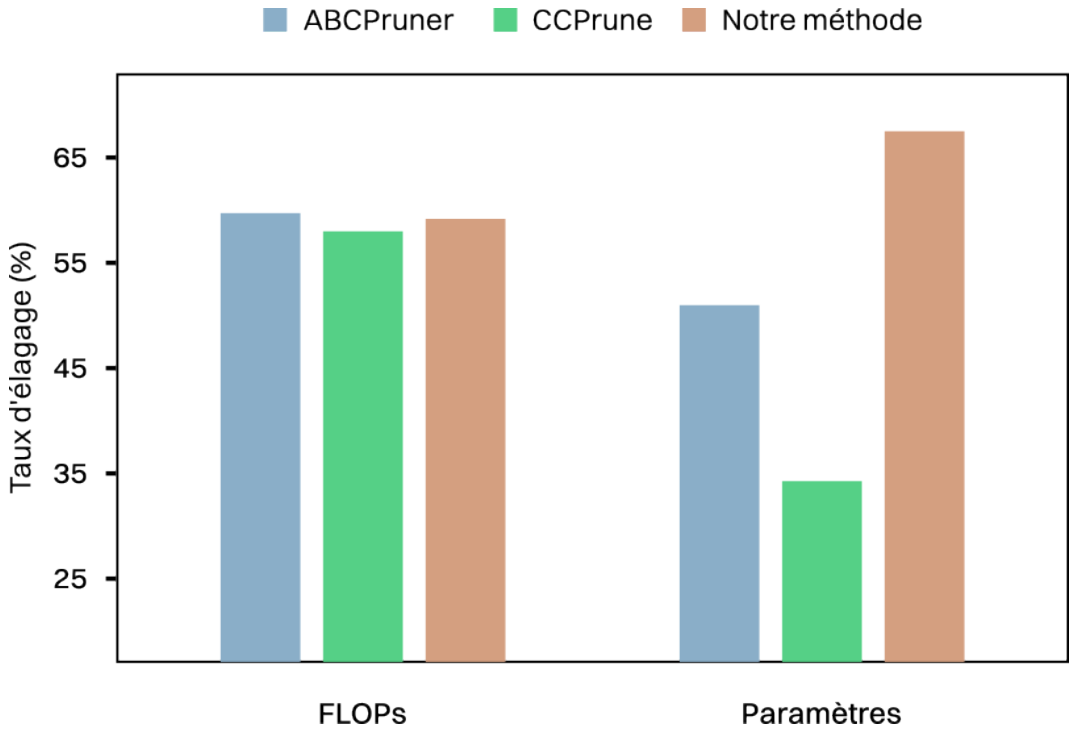

Fig. 4.13 : Comparaison entre les taux d'élagage des FLOPs et des paramètres des trois méthodes d'élagage pour le réseau ResNet-34.

4.7 Conclusion

En conclusion de ce chapitre dédié aux tests et résultats, l'analyse des résultats d'expérimentations menées sur les modèles VGG-19 et ResNet-34 prouvent l'efficacité de notre méthode d'élagage automatique. Nous avons trouvé que notre méthode permet de réduire significativement la taille des modèles et le nombre de FLOPs, en surpassant les deux méthodes utilisées pour la comparaison (ABCPruner et CCPrune). Ces réductions nous permettent d'avoir de très petits modèles qui sont également très performants et faciles à déployer sur des appareils limités en termes de ressources de calcul et de stockage. Cependant, il est important de souligner que le processus d'élagage n'est pas sans compromis. Les avantages en termes de taille sont parfois accompagnés d'une petite dégradation de la précision qui nécessite de faire un réglage fin après l'élagage afin de restaurer partiellement les performances du modèle initial. De plus, le taux d'élagage optimal peut varier en fonction du même jeu de données et des spécificités de la tâche.

On a également présentée au début du chapitre les modèles utilises pour tester la méthode (VGG-19 et ResNet-34) et le jeu de données utilisé (CIFAR-10) pour l'entraînement de ces modèles, ainsi que les différents technologies et outils utilisés pour la conception de notre méthode. Nous avons utilisé le langage de programmation Python avec plusieurs

bibliothèques telles que NumPy, Matplotlib, Pytorch, etc. Ces bibliothèques offrent des outils puissants pour développer et entraîner les différents types de réseaux de neurones.

Enfin, ce chapitre constitue une étape essentielle de ce rapport en fournissant des résultats d'implémentation des concepts et des théories énoncés dans le chapitre précédent. Les résultats obtenus offrent des orientations pratiques pour les applications et les améliorations futures de cette méthode pour élaguer des réseaux de neurones profonds. En considérant ces résultats, la rapport se tourne vers la section finale, où les conclusions et perspectives globales sont tirées.

Conclusion et perspectives

Avec l'augmentation de la profondeur des architecture des réseaux de neurones, le nombre de calculs et la taille des réseaux augmentent également, ce qui rend leurs déploiement sur des appareils dotés d'un matériel limité très difficile et compliqué. L'élagage a émergé comme une approche pour réduire la complexité de ces réseaux profonds. Cependant, cette approche prend beaucoup de temps et nécessite des experts humains afin de bien élaguer un réseau. En raison de ses défauts, des méthodes automatiques utilisant l'apprentissage par renforcement sont apparues. Ces méthodes fournissent des résultats exceptionnels et ont la capacité à s'adapter à une grande variété d'environnements en utilisant des configurations appropriées. Cependant, les algorithmes d'apprentissage par renforcement ne peuvent pas prendre en entrée un réseau profond complet car il est très complexe. Il est donc nécessaire d'utiliser des structures moins complexes comme entrée telles que le plongement de graphe, de chouche, de noeuds, etc. Le plongement est un vecteur unidimentionnel qui garde seulement les informations importantes dans le réseau. En transformant ces données en un vecteur unidimensionnel, cet outil de plongement facilite grandement la capacité de l'agent d'apprentissage par renforcement à appréhender et à interagir avec son environnement.

Dans ce rapport, nous avons découvert les différentes techniques d'élagage des réseaux de neurones, ainsi que les types de plongements et les différents aspects des algorithmes d'apprentissage par renforcement. Nous avons commencé le rapport par une introduction au domaine d'apprentissage profond où nous avons vu quelques définitions et concepts de base, tels que les poids, les connexions, les types de réseaux de neurones, les différents types d'apprentissage (supervisé, non-supervisé, semi-supervisé et par renforcement), etc. Ensuite, nous avons présenté quelques concepts et algorithmes de l'apprentissage par renforcement, tels que les processus de décision de Markov, l'apprentissage Q profond, etc. Nous avons également parlée sur les graphes et les différentes techniques de plongement, ainsi que l'hypothèse du ticket de loterie et les types d'élagage des réseaux de neurones.

On a parlé de tout cela juste pour acquérir suffisamment de connaissances pour pouvoir élaborer une nouvelle approche d'élagage des réseaux de neurones profonds, en utilisant les techniques de plongement de couches et un algorithme d'apprentissage par renforcement complexe pour nous fournir les pourcentage d'élagage du modèle, pour arriver enfin à un modèle de taille considérablement réduite et avec une réduction minimale de la précision. Nous avons commencé par la présentation des modèles testés et le jeux de données utilisée pour les entraîner. Puis, nous avons présenté une vue global de la solution et ensuite les détails de chaque étape de la solution, qui commence par la construction du plongement et finit par le réglage fin. Enfin, nous avons vu les différents résultats d'application de notre méthode sur les modèles et nous les avons comparés avec les résultats de quelques

méthodes performantes.

L'avantage le plus important de notre méthode est qu'elle permet de réduire considérablement la taille des modèles et le nombre de calculs, ce qui est idéal pour déployer ces modèles sur des appareils dotés d'un matériel limité. Toutefois, cet avantage est parfois accompagné d'une petite dégradation de la précision, ce qui nécessite de faire un réglage fin après l'élagage afin de restaurer partiellement la précision du modèle initial.

Même si notre méthode donne de très bons résultats, des améliorations sont encore possibles. Nous pouvons apporter ces améliorations à différentes parties de notre processus d'élagage. Par exemple, nous pouvons essayer de généraliser notre méthode à d'autres types d'architectures de réseau autres que les réseaux convolutifs et résiduels. Nous pouvons également modifier l'algorithme d'élagage de l'élagage des canaux vers un autre type d'élagage qui peut contribuer à augmenter la précision de nos modèles. Nous pouvons également essayer de faire d'autres types de plongement, tels que le plongement de l'ensemble du réseau, et utiliser d'autres algorithmes d'apprentissage par renforcement, tels que PPO ou A3C, ou même essayer un algorithme d'optimisation. Nous pouvons aussi essayer de faire l'élagage au début ou pendant l'entraînement afin d'éviter l'étape de réglage fin qui peut prendre du temps pour le ré-entraînement.

Bibliographie

Aggarwal, Charu C. (2018). *Neural networks and deep learning : A textbook*. Springer.

Alharbi, Nouf Fahad et Nabil M. Hewahi (2021). "Exploring deep neural network capability for intrusion detection using different mobile phones platforms". In : *International Journal of Computing and Digital Systems* 10.1, p. 1391-1406.

Amini, Alexander et al. (2018). "Spatial uncertainty sampling for end-to-end control". In : *arXiv preprint arXiv :1805.04829*.

Bishop, Christopher M. (2016). *Pattern recognition and machine learning*. Springer New York.

Chen, Yanming et al. (2021). "CCPrune : Collaborative channel pruning for learning compact convolutional networks". In : *Neurocomputing* 451, p. 35-45.

Dong, Peijie et al. (juin 2022). "Prior-guided one-shot neural architecture search". In : *arXiv.org*.

Douag, Lyes et Lokmane Sadani (2022). "Etude et implémentation de méthodes pour l'optimisation des architectures neuronales entièrement connectées". Master's thesis. ESI.

Feng, Jie et al. (2020). "Generative Adversarial Networks based on collaborative learning and attention mechanism for hyperspectral image classification". In : *Remote Sensing* 12.7, p. 1149.

Feng, Junxi et al. (2019). "Reconstruction of Porous Media from extremely limited information using conditional generative adversarial networks". In : *Physical Review E* 100.3.

Goodfellow, Ian, Yoshua Bengio et Aaron Courville (2016). *Deep Learning*. MIT Press.

Goodfellow, Ian et al. (2020). "Generative Adversarial Networks". In : *Communications of the ACM* 63.11, p. 139-144.

Handaoui, Mohamed et al. (2020). "Releaser : A reinforcement learning strategy for optimizing utilization of ephemeral cloud resources". In : *2020 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, p. 65-73.

He, Kaiming et al. (juin 2016). "Deep Residual Learning for Image Recognition". In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hu, Rong et al. (2022). "A multi-attack intrusion detection model based on mosaic coded convolutional neural network and centralized encoding". In : *PLOS ONE* 17.5.

Kimura, Nobuaki et al. (2019). "Convolutional neural network coupled with a transfer-learning approach for time-series flood predictions". In : *Water* 12.1, p. 96.

Kingma, Diederik P. et Max Welling (2012). “Auto-Encoding Variational Bayes”. In.

Krizhevsky, Alex et Geoffrey Hinton (2009). *Learning multiple layers of features from tiny images*. Rapp. tech. 0. Toronto, Ontario : University of Toronto.

Kumaraswamy, Balachandra (2021). “Neural Networks for Data Classification”. In : *Artificial Intelligence in Data Mining*, p. 109-131.

Liessner, Roman et al. (2018). “Deep Reinforcement Learning for Advanced Energy Management of Hybrid Electric Vehicles”. In : *International Conference on Agents and Artificial Intelligence*.

Lin, Mingbao et al. (2020). “Channel pruning via automatic structure search”. In : *arXiv preprint arXiv :2001.08565*.

McCulloch, Warren S. et Walter Pitts (1943). “A logical calculus of the ideas immanent in nervous activity”. In : *The Bulletin of Mathematical Biophysics* 5.4, p. 115-133.

Molchanov, Pavlo et al. (2016). “Pruning convolutional neural networks for resource efficient inference”. In : *arXiv preprint arXiv :1611.06440*.

Muniasamy, Anandhavalli et al. (2020). “Deep Learning for Predictive Analytics in Healthcare”. In : *The International Conference on Advanced Machine Learning Technologies and Applications (AMLTA2019)*. Sous la dir. d’Aboul Ella Hassanien et al. Cham : Springer International Publishing, p. 32-42.

Rosenblatt, F. (1958). “The Perceptron : A probabilistic model for information storage and organization in the brain.” In : *Psychological Review* 65.6, p. 386-408.

Simonyan, Karen et Andrew Zisserman (2014). “Very deep convolutional networks for large-scale image recognition”. In : *arXiv preprint arXiv :1409.1556*.

Vaswani, Ashish et al. (2017). “Attention is All you Need”. In : *Advances in Neural Information Processing Systems*. Sous la dir. d’I. Guyon et al. T. 30. Curran Associates, Inc.

Wiering, Marco et Martijn van Otterlo, éd. (2012). *Reinforcement Learning : State-of-the-Art*. Springer.

Yamamoto, Kohei et Kurato Maeno (2018). “PCAS : Pruning Channels with Attention Statistics”. In : *ArXiv* abs/1806.05382.

Yoseph, Fahed, Markku Heikkilä et Daniel Howard (2019). “Outliers Identification Model in Point-of-Sales Data Using Enhanced Normal Distribution Method”. In : *2019 International Conference on Machine Learning and Data Engineering (iCMLDE)*, p. 72-78.

Zhou, Jie et al. (2020). “Graph neural networks : A review of methods and applications”. In : *AI Open* 1, p. 57-81.