REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
École Nationale Polytechnique

Département d'Automatique

Mémoire de projet de fin d'études

En vue de l'obtention du diplôme d'ingénieur d'état

en Automatique

Commande d'un véhicule autonome basée sur le End to End deep learning

Présenté par : ARIBI Imène & SAYAD Khaled

Sous la direction de Mr. H.CHEKIREB (ENP)

Présenté(e) et soutenue publiquement le (24/06/2018)

Composition du Jury :

Président	Mr. L.ABELOUEL	Enseignant à ENP
Rapporteur	Mr. H.CHEKIREB	Professeur à ENP
Examinateur	Mr. R.ILLOUL	Maître de conférences à ENP

ENP 2018

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
École Nationale Polytechnique

Département d'Automatique

Mémoire de projet de fin d'études

En vue de l'obtention du diplôme d'ingénieur d'état

en Automatique

Commande d'un véhicule autonome basée sur le End to End deep learning

Présenté par : ARIBI Imène & SAYAD Khaled

Sous la direction de Mr. H.CHEKIREB (ENP)

Présenté(e) et soutenue publiquement le (24/06/2018)

Composition du Jury :

Président	Mr. L.ABELOUEL	Enseignant à ENP
Rapporteur	Mr. H.CHEKIREB	Professeur à ENP
Examinateur	Mr. R.ILLOUL	Maître de conférences à ENP

ENP 2018

Dédicace

A la mémoire de ma grand-mère, puisse Dieu l'accueillir dans miséricorde.

A celle et celui qui ont toujours garni mon chemin avec force et lumière, mes très chers parent.

A mon grand-père.

A toute ma famille pour leur amour et le respect qu'ils m'ont toujours accordé.

A mes amis et à tous les membres du Club d'Activités Polyvalentes (CAP).

A toute personne qui m'a aidé à franchir un horizon dans ma vie.

Imène

Dédicace

Je dédie ce modeste travail à :

A mes très chères parents : *aucune dédicace ne saurait être assez éloquente pour exprimer ce que vous méritez pour tous les sacrifices que vous n'avez cessé de me donner depuis ma naissance, durant mon enfance et même à l'âge adulte.*

A mes très chères Sœurs et Frères : *En témoignage de l'attachement, de l'amour et de l'affection que je porte pour vous.*

A mes très chères Nièces et Neveux : *pour l'espoir que vous gravez de jour en jour dans mon cœur.*

A mes Amis et camarades de L'École Nationale Polytechnique

Moncef Khaled

Remerciements

Avant toutes choses, nous tenons à exprimer notre très grande estime et notre gratitude à l'ensemble du personnel de cette école pour toutes ces années passées au sein d'un établissement convivial d'un très haut niveau intellectuel et académique.

Le jour final de l'accomplissement de ce parcours universitaire est arrivé et c'est avec un immense honneur et un grand plaisir que nous présentons nos remerciements et nos profondes reconnaissances à nos encadreurs Pr. Hachemi CHKIREB et M. Abdellah KHELLOUFI, pour leur patience et leur savoir-faire qui nous ont parmi de travailler dans un environnement approprié pour la préparation de ce mémoire.

Nos vifs remerciements vont également aux les membres du jury ; Mr. L.Abellouel enseignant chercheur à l'école nationale polytechnique et Mr. R.Illoul maitre-assistant à l'école nationale polytechnique.

Nos remerciements s'étendent aussi à l'ensemble des enseignants du département Automatique pour la richesse et la qualité de leur enseignement ainsi que nos familles pour leur soutien financier et moral.

Enfin nous remercions tous ceux qui ont contribué à la réalisation de ce travail de près ou de loin.

ملخص: تتمتع المركبات ذاتية الخدمة بميزات كبيرة لتحسين الحركة وتقليل التلوث. كانت الخوارزميات التقليدية القائمة على رؤية الكمبيوتر هي الطريقة الرئيسية للمالحة ذاتية الحركة للمركبات حيث كان صنع القرار نتيجة لسلوك تم إنشاؤه يدويًا. في السنوات اﻷلخيرة، أظهر التعلم العميق قدراته االستثنائية في التعرف البصري وصنع القرار في اﻷلنظمة المتكاملة. في هذه اﻷطروحة، سوف ندرس أداء استراتيجية التعلم E2E مع تقنية :CNN الشبكة العصبية الضيقة.

الكلمات المفتاحية: التعلم العميق ، رؤية الكمبيوتر ، المركبات ذاتية الحكم ، الشبكات العصبية التحويلية ، التعلم من طرف إلى طرف.

Abstract: Autonomous vehicles promise great benefits to humanity, such as significantly reducing traffic accidents, improving mobility and reducing pollution. Traditional algorithms based on computer vision were the main method of navigation of an autonomous vehicle where decision making was the product of manually constructed behaviors. In recent years, deep learning has demonstrated its extraordinary visual recognition and decision-making capabilities in end-to-end systems. In this thesis, we will study the performance of an E2E learning strategy combined with CNN technology, which are machine learning models that provide leading-edge results in various computer vision task

Key words: deep learning, computer vision, autonomous vehicles, convolutional neural networks, end to end learning.

Résumé : Les véhicules autonomes présentent de grands avantages pour l'amélioration de la mobilité et la réduction de la pollution. Les algorithmes traditionnels basés sur la vision par ordinateur ont été la principale méthode de navigation d'un véhicule autonome où la prise de décision était le produit d'un comportement construit manuellement. Ces dernières années, l'apprentissage profond a démontré ses extraordinaires capacités de reconnaissance visuelle et de prise de décision dans les systèmes end to end. Dans ce mémoire, nous étudierons la performance d'une stratégie d'apprentissage E2E combinée avec la technologie des CNN (convolutionnal neuron network).

Mots clés: apprentissage profond, vision par ordinateur, véhicules autonomes, réseaux de neurones de convolution, end to end learning.

TABLE DES MATIÈRES

Table des Figures

Introduction générale

Les véhicules modernes évoluent de plus en plus vers des véhicules intelligents en raison des développements fulgurants enregistrés dans le domaine de l'électronique numérique, de la technologie des capteurs et des logiciels.

Les véhicules intelligents ont la capacité de :

- comprendre l'état propre du véhicule et son environnement ;
- communiquer avec l'environnement ;
- planifier et d'exécuter des manœuvres.

Donc, un véhicule intelligent va pouvoir fournir des informations nécessaires à la commande totale du véhicule ou à aider le conducteur pour améliorer les taches de conduite du véhicule.

Il existe principalement deux approches pour le contrôle autonome d'un véhicule. Celles-ci font actuellement l'objet d'une recherche intensive dans le milieu académique et dans l'industrie. La première approche concerne les techniques habituelles issues de la vision par ordinateur qui sont bien comprises et bien appliquées cependant, leurs optimisations et la combinaison de diverses méthodes demeurent encore un sujet de recherche. La deuxième approche est basée sur l'utilisation de la perception, l'apprentissage automatique et les réseaux de neurones dits de convolution (CNN ou ConvNet). En effet, ce type de réseaux de neurone est apte à comprendre les scènes issues de la conduite pour contrôler un véhicule autonome. Ceci est obtenu grâce à l'apprentissage, automatiquement et sans intervention des ingénieurs, de plusieurs formes complexes.

Dans ce mémoire, nous avons choisi d'utiliser l'approche « end to end deep learning » pour contrôler le véhicule d'une manière autonome, cette nouvelle approche ne nécessite qu'une seule caméra frontale, un réseau CNN et la procédure d'apprentissage « end to end deep learning ». L'approche est bien avantageuse car elle est relativement simple et rentable par rapport à la norme actuelle dans la technologie des véhicules autonomes où est nécessaire le recours à la fusion des données issues de plusieurs capteurs (i.e caméras, radar et lidar).

Si on peut montrer que l'approche « end to end deep learning » est à même de piloter d'une manière satisfaisante une voiture dans une course automobile où les contraintes sont très fortes, elle est alors apte à supplanter largement la norme actuelle.

Pour ce faire, nous avons subdivisé notre mémoire en cinq chapitres. Au chapitre 1, on trouve surtout les définitions et une classification relatives à la conduite des véhicules. Un rappel succinct, relatif à l'apprentissage automatique et aux réseaux de neurones, est exposé au chapitre 2. C'est au chapitre 3 qu'on retrouve une présentation assez détaillée des réseaux de neurones dits de convolution (CNN ou ConvNet) où nous avons insisté sur la structure particulière de ces réseaux et leur construction. Nous détaillons également la structure du ConvNet que nous projetons d'exploiter pour la conduite d'un véhicule autonome. Le chapitre 4 est consacré aux techniques d'apprentissage profond des ConvNets et autre à la présentation en particulier de l'apprentissage « end to end deep learning ». Le chapitre 5 est le plus important puisque, nous exposons les résultats de notre travail relatif à la conduite d'un véhicule autonome. Ainsi, nous montrons de quelle manière nous recueillons les données relatives à l'entrainement et à la validation du réseau ConvNet proposé. Nous expliquons de quelle manière l'apprentissage est effectué et comment amélioré les performances du ConvNet. Par la suite, nous testons ce réseau dans des situations non présentées lors de l'apprentissage. Le mémoire se termine par une conclusion où nous résumons l'essentiel de notre travail et nous donnons quelques perspectives.

Chapitre I

Véhicules Autonomes : Introduction

Les voitures autonomes devraient faire le saut de la science et de la recherche vers les usines et les consommateurs dans les prochaines années dans les pays développés. En fait, un intérêt croissant pour cette nouvelle forme de mobilité existe, des constructeurs automobiles ainsi que de nouveaux joueurs économiques comme Google, investissent dans le développement des voitures autonomes. Des gouvernements offrent des mesures de soutien et adaptent leur cadre juridique pour préparer leurs pays à cette nouvelle industrie prometteuse. La prochaine révolution de la mobilité est dans l'air. On attend beaucoup du véhicule autonome : qu'il nous permette de nous déplacer partout, tout le temps, sans effort ni surcoût, qu'il soit disponible au simple claquement de nos doigts, qu'il nous transporte de manière sûre, rapide, et non polluante, d'un bout à l'autre de la ville ou du pays. Si cette révolution est une révolution des usages, c'est aussi et avant tout une révolution d'une industrie. L'industrie automobile, qui va être affectée de bout en bout par ce nouveau paradigme. Constructeurs, équipementiers, loueurs, géants du numérique : tous vont se battre férocement dans l'arène de l'autonomie. Laissant les vaincus exsangues et agonisants, portant les vainqueurs au sommet de la rentabilité et de la part de marché.

1 Histoire

L'idée du véhicule autonome a presque 75 ans, malgré des avantages évidents, ils n'ont jamais réalisé une percée majeure. La raison principale était que toutes les solutions passées nécessitaient d'immenses adaptations de l'environnement. Les exemples sont des fils de cuivre le long des rues ou des voies clôturées, débarrassées de tout objet inattendu et d'éventuels obstacles. Ces adaptations coûteuses de l'environnement souffrent des problèmes qui empêchent toute mise en œuvre au-delà des petits projets occasionnels.

Depuis quelques années, l'informatique moderne et la technologie des capteurs fournissent pour la première fois dans l'histoire les moyens techniques de créer des voitures réellement autonomes. Ces voitures roulent de manière autonome et libre dans des environnements ouverts et incertains créés pour les humains. Ce nouveau type de VA, qui est un pas de géant de la robotique autonome dans des environnements contrôlés et restreints, fait l'objet de ce mémoire. L'événement de décollage pour ces nouveaux véhicules autonomes a été classé premier dans le DARPA Grand Défi en 2004. Le défi DARPA et ses événements ont attiré l'attention des médias.

Les VA se sont soudainement trouvés dans le centre d'intérêt de divers instituts techniques et des grandes entreprises avec des divisions de R&D bien financées. Très récemment, c'est-à-

dire depuis une ou deux années, les gouvernements ont commencé à considérer explicitement les VA comme une nouvelle industrie stratégique possible. Ils ont commencé à fournir le cadre juridique nécessaire pour tester et utiliser les VA dans leurs rues aux États-Unis au Japon à titre d'exemple.

2 Impact des véhicules autonomes sur la mobilité

Le véhicule autonome va affecter l'ensemble des concepts de mobilité et la notion de transport elle-même va être modifiée. Dans cette section, on va discuter les avantages de l'usage des véhicules autonomes sur la mobilité dans les sociétés de demain.

- **La sécurité routière :** C'est un avantage majeur pour les passagers des véhicules et pour les passants: l'accidentologie devrait baisser de manière substantielle avec la progression des véhicules autonomes. Les études ont effectivement montré que la plupart des accidents sont principalement dus à une erreur humaine. Il est donc raisonnable d'espérer une chute d'accidentologie dans de telles proportions.

- **La pollution :** Les véhicules autonomes auront une conduite plus économique que celle des humains. Plusieurs facteurs montrent qu'il est plus facile de rendre autonome un véhicule électrique qu'un véhicule thermique, car toutes les commandes du véhicule (y compris le moteur) y sont électriques. Cet élément pourrait d'ailleurs être un argument pour convaincre les législateurs de l'intérêt à favoriser le développement du véhicule autonome et électrique.

- **Le confort :** Les conducteurs seront partiellement ou totalement libérés de la tâche de conduite, qui est une tâche fatigante, en particulier sur les longs trajets, les trajets récurrents, et dans les embouteillages. En plus, les véhicules autonomes auront probablement une conduite moins brusque que celle des humains, avec des freinages, des prises de virage, des accélérations standardisées et optimisées.

3 Le degré d'autonomie d'un véhicule autonome

L'agence fédérale américaine des États-Unis chargée de la sécurité routière NHTSA (National Highway Traffic Safety Administration) a adopté en janvier 2016 les standards du SAE International (anciennement SAE, Society of Automotive Engineers) quant à la définition des niveaux d'autonomie du véhicule. Depuis cette date, ces standards sont considérés comme des standards internationaux et l'ensemble des acteurs du véhicule autonome s'y réfère. De la Figure 1.1, ces standards comportent cinq niveaux d'automatisation :

- **Niveau 0** : aucune automatisation. Le conducteur humain est en permanence en charge de tous les aspects de la tâche de conduite dynamique, même s'il est aidé par des systèmes d'alertes ou d'intervention;

- **Niveau 1** : assistance au conducteur. Un système d'assistance à la conduite est en charge, spécifiquement en mode conduite, soit de la direction, soit de l'accélération/décélération, en utilisant des informations sur l'environnement de conduite, et en se reposant sur le conducteur humain pour s'occuper de tous les aspects restants de la tâche de conduite ;

- **Niveau 2** : automatisation partielle. Un ou plusieurs systèmes d'assistance à la conduite sont en charge, spécifiquement en mode conduite, du freinage et de l'accélération/décélération, en utilisant des informations sur l'environnement de conduite, et en en se reposant sur le conducteur humain pour s'occuper de tous les aspects restants de la tâche de conduite dynamique ;

- **Niveau 3** : automatisation conditionnelle. Un système de conduite autonome est en charge, spécifiquement en mode conduite, de tous les aspects de la tâche de conduite dynamique, et attend du conducteur humain qu'il répondra de manière appropriée à une requête d'intervention;

- **Niveau 4** : automatisation élevée. Un système de conduite autonome est en charge, spécifiquement en mode conduite, de tous les aspects de la tâche de conduite

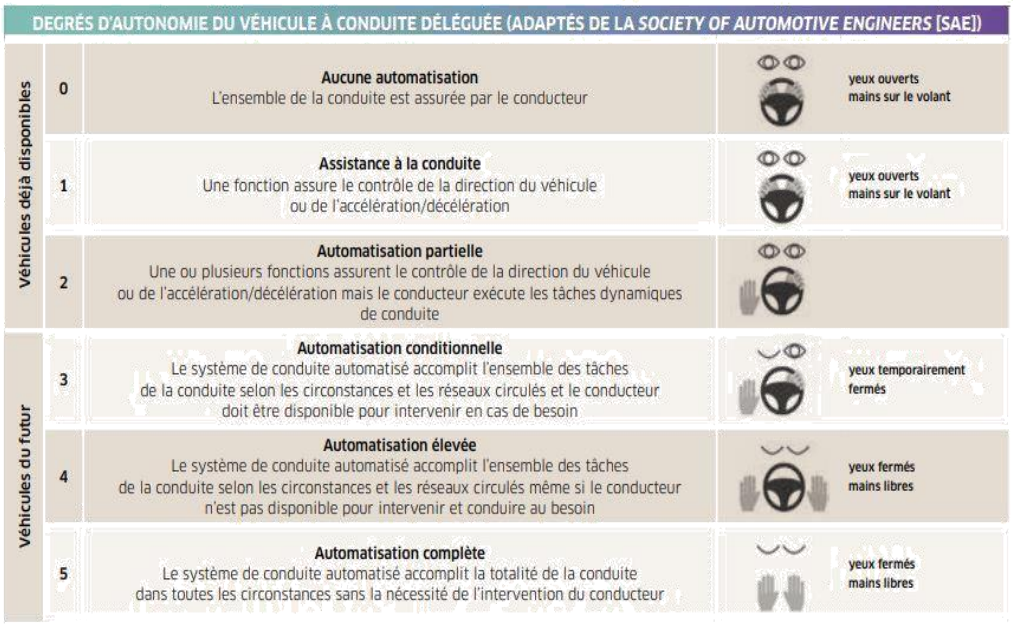


Fig.1.1 Degrés d'automatisation d'un véhicule autonome

dynamique, même si le conducteur humain ne répond pas de manière appropriée à une requête d'intervention;

- **Niveau 5** : automatisation totale, un système de conduite autonome est en charge, en permanence, de tous les aspects de la tâche de conduite dynamique, dans toutes les conditions de route et d'environnement qui peuvent être gérées par un conducteur humain.

4 Conduite d'un véhicule autonome

Pour devenir autonome, la voiture a besoin de diverses informations et d'une combinaison d'éléments. On peut décomposer le processus de conduite d'un véhicule autonome en 5 étapes :

- **Localisation et cartographie** afin de déterminer où de situer le véhicule par rapport à un repère donné.
- **Compréhension de la scène** permet de détecter les objets présents dans l'environnement où se situe le véhicule.
- **Planification de mouvement** assure le calcul de la trajectoire pour aller de A à B.
- **L'interaction Homme-Machine,** dans les véhicules autonomes, nécessite la connaissance de l'état du conducteur avant de procéder à toute une action.
- **La communication** permet, à un véhicule autonome, d'avoir la capacité de communiquer les actions avant d'interagir avec le monde extérieur.

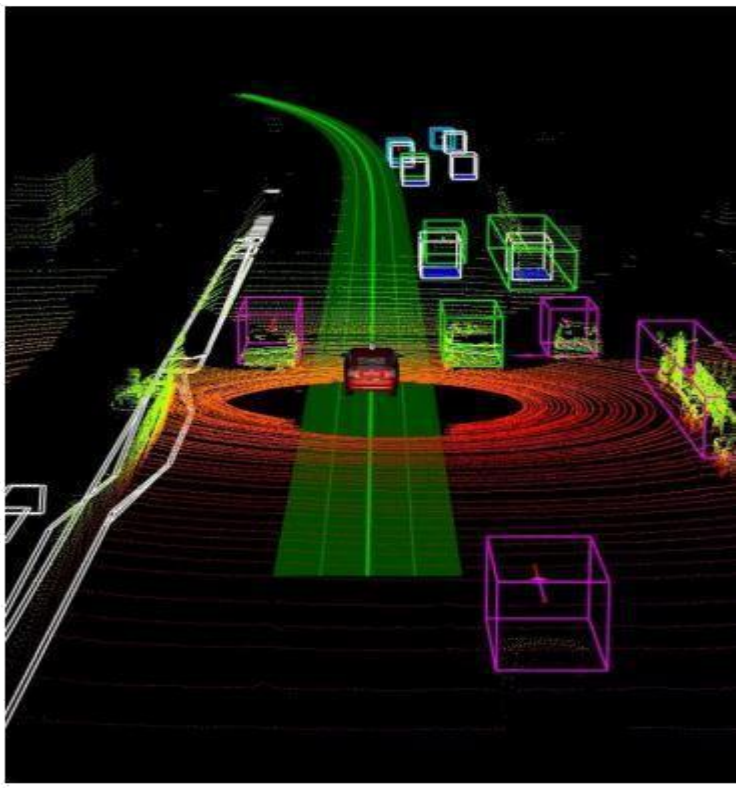


Fig.1.2 Illustration de conduite d'un véhicule autonome dans son environnement réel

A l'aide d'une caméra frontale, située sur le rétroviseur central, le véhicule identifie les couleurs, les panneaux et le marquage au sol et adapte ainsi son allure et ses arrêts. De nuit, une caméra infrarouge permet de déceler les piétons et les animaux sur la route.

Le véhicule est équipé de capteurs à ultrasons pouvant détecter des obstacles situés à quelques mètres. De plus, des radars et des capteurs à longue portée en avant et en arrière du véhicule déterminent, quelle que soit la météo, la position et la vitesse relative des autres véhicules et vont permettre le contrôle de la trajectoire et le maintien des distances de sécurité avec les autres véhicules. En outre, ces équipements gèrent aussi d'éventuels freinages brusques ou le stationnement du véhicule comme le font déjà certains véhicules ayant le Park assit.

En utilisant plusieurs lasers, le Lidar, situé sur le toit, établit une cartographie précise en 3D de l'environnement et des objets entourant la voiture. Celle-ci est mise à jour en permanence.

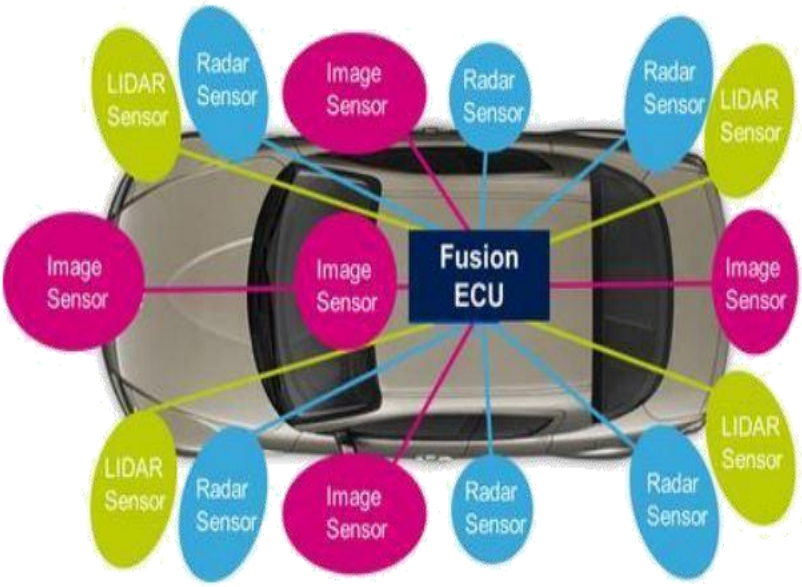


Fig.1.3 Capteurs équipant un véhicule autonome et leurs positions sur le véhicule

Les algorithmes implémentés au niveau de l'ordinateur central prennent en compte toutes ces données pour prendre des décisions dans le but de transporter les passagers en toute sécurité. Enfin, le système est complété d'un odomètre, appareil qui sert à mesurer la vitesse ou la distance parcourue par l'automobile.

Pour fonctionner correctement, un véhicule autonome doit parcourir des milliers de kilomètres pour alimenter sa base de données en images, expériences et interactions. Cette base de données va être utilisée dans la phase d'apprentissage. En fait, c'est le principe de l'apprentissage automatique.

Chapitre II
Introduction aux réseaux de neurones artificiels et leurs apprentissages automatiques

AlphaGo, un logiciel développé par Google a pu vaincre le champion du monde du jeu de GO, le plus compliqué des jeux de stratégie. En fait, les experts de l'intelligence artificielle n'ont jamais pensé qu'un tel développement peut se produire avant dix ans. Ce développement accéléré dans le domaine de l'apprentissage profond a influencé la mise en place de nouveaux champs de recherches : perception pour véhicules autonomes, reconnaissances des voix, images et des émotions…etc.

Dans ce chapitre, on va discuter les bases de l'apprentissage profond en commençant par placer l'apprentissage profond dans le contexte de l'intelligence artificielle. Ensuite, on va présenter les bases de l'apprentissage automatique. Les réseaux de neurones artificiels sont étudiés afin de présenter un modèle mathématique qui permet de mettre en œuvre les algorithmes d'apprentissage. Ensuite, on abordera les différents modèles d'apprentissage profond, supervisés et non supervisé et le développement du domaine ces dernières années.

1 Les bases de l'apprentissage automatique

1.1 Définition

Le terme « apprentissage automatique » ou bien « machine learning » a été introduit pour la première fois par Arthur Samuel en 1959. Un algorithme d'apprentissage automatique est un algorithme capable d'apprendre un ensemble de règles à partir d'un jeu de donnés. Schématiquement, un programme apprend d'une expérience E, en respectant une certaine classe de tâches T et une mesure de performance P, si sa performance P vis-à-vis de la tâche T s'améliore au fur et à mesure avec l'expérience E [2]. Ainsi, on peut imaginer l'existence de plusieurs variétés d'expériences E, des taches T et des performances P, à partir de ces variétés, on peut donc construire un algorithme d'apprentissage automatique spécifique à chaque problématique.

1.2 Quelques exemples de tâches

Le processus d'apprentissage ne représente pas la tâche à effectuer mais plutôt, la façon par laquelle on procède pour effectuer la tache demandée. Par exemple, si on veut qu'un robot se déplace dans une direction, la tâche à prendre en charge serait un problème de navigation. Ainsi, on peut soit programmer le robot pour lui apprendre comment se déplacer soit, écrire un programme qui spécifie comment le faire déplacer manuellement.

Les tâches sont spécifiées selon la façon avec laquelle on veut que le programme d'apprentissage automatique traite l'exemple. Un exemple peut être un ensemble de caractéristiques mesurées quantitativement tirées d'une expérience ou un objet qu'on veut le programme d'apprentissage automatique traite. Un exemple est généralement représenté par un vecteur dont chaque composante est une caractéristique. Par exemple, les caractéristiques d'une image sont les valeurs des pixels dans la même image [2].

Ils existent plusieurs types des tâches qu'on peut effectuer avec les algorithmes d'apprentissage automatique, parmi lesquelles on distingue :

- **La classification**

Effectuer une tâche de classification par un algorithme d'apprentissage profond revient à attribuer une catégorie parmi catégories auxquelles l'entrée peut appartenir. Pour résoudre ce type de problème, l'algorithme d'apprentissage doit générer une fonction qui génère une distribution de probabilité sur les catégories. Par exemple, si on injecte une image d'un chiffre grec et on demande de reconnaitre le chiffre puis de générer son équivalent en chiffre arabe, ici la fonction est telle que
[0 , 0.3 , 0.2 , 0] et la sortie serait de type [0, 0.4 , 0 , 0.1 , 0 , 0].

- **La régression**

Dans ce type de tâche, on fait apprendre à l'algorithme comment attribuer une valeur numérique étant donnés des entrées. Pour résoudre ce problème, l'algorithme d'apprentissage doit générer une fonction. En fait, ce type de tâche est similaire à la classification, mais il diffère par le type de sortie. Un exemple typique de cette tâche concerne la génération de la commande d'un robot mobile en se basant sur les images de l'environnement comme entrée.

- **La détection d'anomalie**

Dans ce type de tâche, l'algorithme d'apprentissage est exposé à un jeu de données avec l'objectif de détecter une anomalie ou bien un évènement étrange. Par exemple, dans la détection des emails indésirables (SPAM), l'algorithme détecte une anomalie représentée par un message dont le contenu ou bien l'expéditeur sont différents des contacts usuels.

- **La Traduction**

Ici, l'algorithme doit être apte à transformer une chaine de caractères d'une langue en une chaine de caractères d'une autre langue. Ce type de tâche est utilisé dans le traitement des langages naturels (Natural Language Processing).

1.3 La mesure de performance

Afin d'évaluer un algorithme d'apprentissage, on fait appel à une mesure quantitative de performance P. Cette mesure de performance est spécifique à la tache T en cours d'exécution par l'algorithme. Par exemple, pour une tache de classification, on s'intéresse particulièrement à « L'exactitude » du modèle laquelle est représentée par la proportion d'exemples pour laquelle l'algorithme génère des résultats exacts [2].

On teste l'algorithme d'apprentissage généralement sur un jeu de données différent de celui de la phase d'apprentissage afin de bien simuler les conditions réelles car on s'intéresse plus au comportement de l'algorithme lorsqu'on l'expose à des données nouvelles.

Le choix de la mesure de performance P est souvent difficile et compliqué. La complexité réside dans le fait qu'on ne peut pas préciser la quantité à mesurer au préalable et cela varie d'une application à une autre. Par exemple, pour une tache de régression, on peut pénaliser le modèle s'il commit fréquemment des petites erreurs ou bien s'il commit rarement des grandes erreurs. Par contre, dans d'autres applications, on connait au préalable la quantité à mesurer mais sa mesure est impossible. Par exemple, pour un modèle probabiliste, le calcule des probabilités associées à des points dans l'espace se fait implicitement d'où l'impossibilité de mesurer la probabilité actuelle d'un point de l'espace [2].

1.4 Les Différents types d'apprentissage

Un algorithme d'apprentissage automatique peut être classé selon l'expérience E qu'on le permettre d'avoir durant la phase d'apprentissage sur un jeu de données. Un jeu de données (« dataset » en anglais) est un ensemble des exemples qui varient d'une application à une autre (images, fichiers audio, expressions facials…) et qui sert à entrainer le modèle d'apprentissage.

On peut classifier les paradigmes d'apprentissage en 3 catégories : apprentissage : supervisé, non supervisé et par renforcement.

i. Apprentissage non supervisé

Dans ce paradigme d'apprentissage, l'algorithme doit extraire des classes (caractéristiques) à partir d'un jeu de données non étiquetées. L'absence d'étiquetage entraine l'absence des moyens de validation des résultats de l'algorithme d'apprentissage. Donc, le modèle apprend à extraire des caractéristiques communes entre des individus présents dans le jeu de données.

On juge la performance du modèle par sa capacité à extraire et à classifier toutes les caractéristiques cachées dans un jeu de données.

ii. Apprentissage supervisé

Dans ce paradigme d'apprentissage, l'algorithme doit extraire des classes (caractéristiques) à partir d'un jeu de données étiquetées. Les données étiquetées sont traitées et validées, donc le modèle doit générer des règles à partir des exemples vérifiés.

iii. Apprentissage par renforcement

Dans ce paradigme d'apprentissage, l'algorithme d'apprentissage reçoit une information sur l'exactitude du résultat à chaque itération afin d'optimiser une récompense quantitative au cours du temps. Donc, l'algorithme doit interagir avec l'environnement pour avoir un retour d'état sur les performances instantanées.

On peut aussi définir ces paradigmes d'apprentissage en termes de probabilité. Dans l'apprentissage non supervisé, l'algorithme traite des données présentes dans un vecteur aléatoire et cherche à apprendre la probabilité $P(x)$. Pour l'apprentissage supervisé, l'algorithme observe un vecteur aléatoire x et un autre vecteur associé y et cherche à trouver la probabilité $P(y \vee x)$ (la probabilité d'apparition de sachant). Donc, on peut remarquer que pour l'apprentissage supervisé l'intervention d'un superviseur qui fixe y d'où le terme supervisé.

1.4 Apprentissage automatique via la descente du gradient stochastique

La descente du gradient est une procédure d'optimisation utilisée dans l'apprentissage automatique. Cette méthode est appliquée lors de la maximisation ou la minimisation d'une fonction à plusieurs variables.

Pour toute fonction $f(x_1, x_2, x_3 \ldots, x_n)$ différentiable à n-variables, le gradient est un vecteur g défini tel que :

$$g(x_1, x_2, x_3 \ldots, x_n) = \nabla f(x_1, x_2, x_3 \ldots, x_n)$$

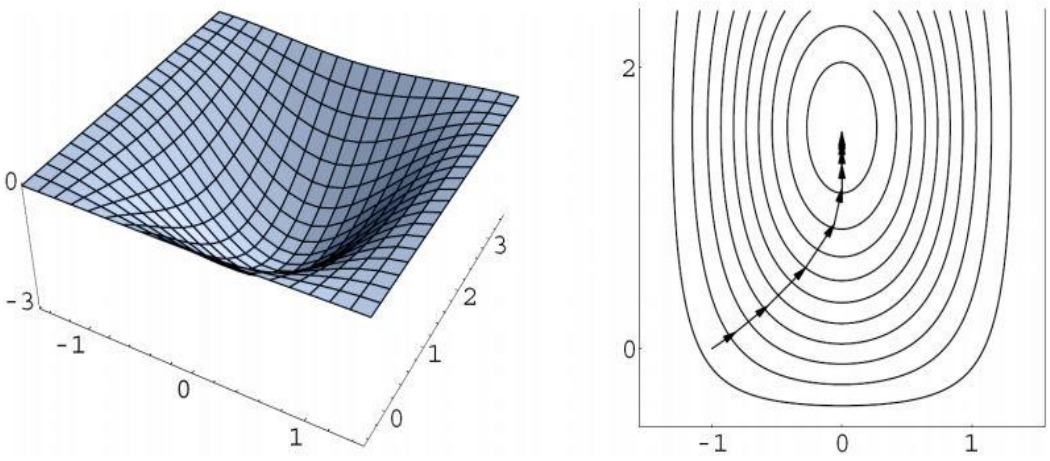


Fig 2.1 Visualisation de la procédure de descente du gradient pour une fonction à 2 variables

Pour trouver le minimum local d'une fonction donc pour assurer sa décroissance, on effectue l'exploration tel que le point considéré est pris dans le sens de l'évolution de la décroissance du gradient de la fonction (i.e : dans le même sens que $-|g|$ au point courant). Donc, durant la procédure du gradient, on cherche à effectuer la plus profonde descente à partir d'un point de l'espace comme le montre la Figure 2.1.

La descente du gradient stochastique est une méthode issue des méthodes de la descente du gradient. Dans le cas de l'apprentissage automatique, on s'intéresse au problème de minimisation d'une fonction de coût $L(w)$ qui est prise dans la forme d'une somme de fonctions différentiables :

$$L(w) = \sum_{i=1}^{n} L_i(w)$$

Donc, on cherche une estimation de w qui minimise $L(w)$ et chaque fonction $L_i(w)$ est associée avec la $i^{iéme}$ observation de jeu de données utilisé pour l'apprentissage.

Durant la procédure de descente du gradient stochastique, la valeur du gradient est approchée par le gradient d'une seule des fonctions L_i. L'algorithme est comme suit :

a) Choisir le point de départ (un vecteur initial du vecteur des paramètres w) et un taux d'apprentissage η.

b) Répéter jusqu'à ce qu'un minimum soit atteint ou approché :
   - Mélanger aléatoirement les échantillons de jeu de données.
   - Pour $i \in N$ faire : $w = w - \eta \nabla F_i(w)$

1.5 Problème de Sur-apprentissage (Overfitting) et de sous-apprentissage (Underfitting)

Les informations utilisées par l'algorithme d'apprentissage automatique pour construire le modèle optimal d'apprentissage sont tirées de plusieurs jeux de données, chaque jeu de données est utilisé étape par étape pour créer le modèle.

D'abord, l'apprentissage se fait sur un jeu de données d'entrainement (training dataset) afin de fixer les paramètres du modèle. Le modèle est testé sur un jeu de données d'entrainement, le résultat est comparé à une valeur réelle (l'objectif) pour ajuster les paramètres du modèle. Ensuite, le modèle est utilisé pour estimer les valeurs d'observations sur un autre ensemble, qui est le jeu de données de validation. Ce dernier teste donne une évaluation impartiale du modèle tout en réglant ses hyper-paramètres.

Les hyper-paramètres d'un modèle d'apprentissage automatique sont les paramètres qu'on fixe avant le lancement du processus d'apprentissage, les autres paramètres étant fixés durant la phase d'entrainement.

L'avantage d'utiliser le jeu de données de validation est d'éviter le sur-apprentissage (overfitting), qui est un problème dû à l'augmentation de l'erreur de validation alors que

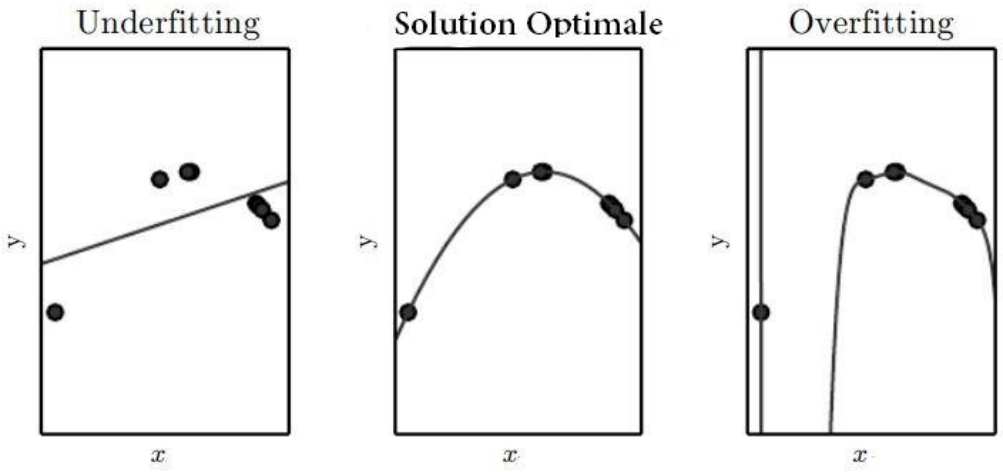


Fig. 2.2 Illustration des problèmes d'overfitting et d'underfitting

l'erreur d'apprentissage diminue, ce problème est provoqué par un mauvais dimensionnement de la structure du modèle. L'utilisation d'un jeu de données de validation permet d'introduire l'opération de régularisation qui est un mécanisme consistant à ajouter de l'information durant le processus d'apprentissage pour minimiser l'overfitting.

L'underfitting apparrait lorsque le modèle est incapable de minimiser l'erreur sur le jeu de données d'entrainement, pour remédier à ce problème on doit augmenter la taille du réseau (en ajoutant des couches cachées).

1.6. L'apprentissage profond dans le contexte de l'intelligence artificielle

Depuis l'antiquité, l'homme a essayé d'inventer des machines dotées d'intelligence. Les premières apparitions d'une envie de concevoir une intelligence artificielle remontent aux mythes Grecs. En effet dans l'Iliade d'Homère, Héphaïstos, le dieu du feu chargé de la métallurgie, conçoit des femmes faites d'or et qui peuvent agir comme un humain [1]**.** De nos jours, l'intelligence artificielle représente un domaine de recherche très actif avec plusieurs applications : reconnaissance des voix et des images, véhicules autonomes, diagnostic médicale …etc.

Tout à fait au début, les projets d'intelligence artificielle étaient majoritairement basés sur une connaissance programmée par des langages formels. Donc, un ordinateur peut raisonner automatiquement en utilisant des règles d'inférence logique ; cette approche est appelée « la base de connaissance pour l'intelligence artificielle », mais il a été prouvée qu'elle est

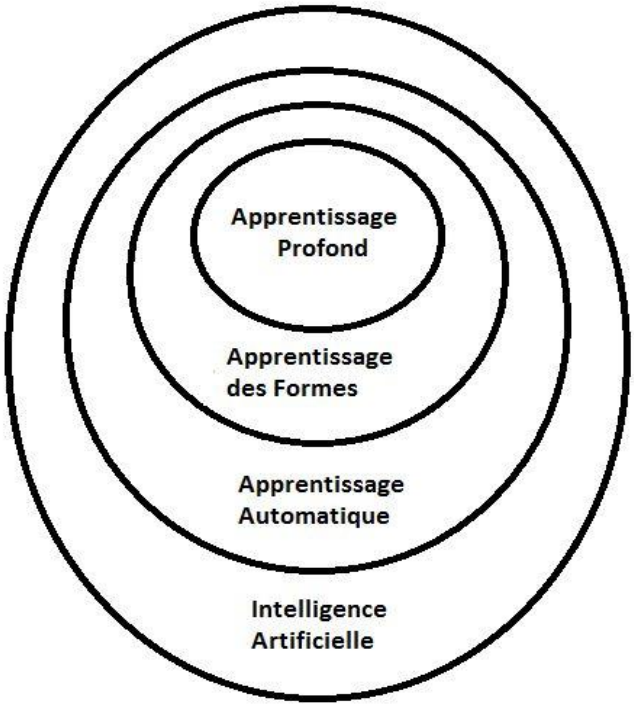


Fig. 2.3 Place de l'apprentissage profond selon le diagramme de Venn

inefficace pour de nombreuses applications. Pour remédier à ce problème, les chercheurs se sont orientés vers la conception de machines capables d'acquérir leurs connaissances d'une façon autonome, d'où l'apparition de « l'apprentissage automatique » connu aussi sous le vocable anglophone « machine learning ».

Les machines auront donc la possibilité d'acquérir de l'expérience et de ce fait, leurs apprentissages n'imposent pas à leur concepteur de fournir toutes la connaissance sous un langage formel. Par ailleurs, la hiérarchisation des concepts permet aux machines d'apprendre des concepts compliqués à partir de concepts plus simples. Si on représente le graphe de la hiérarchie de ces concepts, on obtient un graphe à plusieurs couches, la couche la plus interne (profonde) correspond au concept de «l'apprentissage profond » [2]**.** Donc, il apparait clairement que l'apprentissage profond est une méthode de l'intelligence artificielle qui exploite les techniques de l'apprentissage automatique en autres une forme particulière de l'apprentissage des formes.

Plus précisément, les méthodes de l'apprentissage profond représentent un sous-espace des méthodes de l'apprentissage des formes (Feathers learning); ces méthodes se distinguent par un apprentissage des formes d'une façon hiérarchique. En fait, l'apprentissage d'une forme

complexe est concrétisé par un apprentissage des éléments simples constituants cette forme complexe.

2 Les réseaux de neurones artificiels

L'idée de l'apprentissage profond est inspirée du fonctionnement du cortex humain. En effet, l'apprentissage profond est souvent nommé « réseau de neurones artificiels ».

Les réseaux de neurones artificiels représentent une modélisation mathématique des réseaux biologiques neuronaux. Le but ultime était de simuler le fonctionnement du cerveau humain par une machine. Actuellement, les réseaux de neurones ont prouvé leurs efficacités dans le domaine de la classification et de la reconnaissance d'image et de la voix. Ces architectures sont aussi utilisées pour approximer des fonctions non linéaires pouvant dépendre d'un grand nombre d'entrées.

2.1 Aspects biologiques des réseaux de neurones

Le cerveau humain est constitué de neurones qui se connectent entre eux moyennant des signaux chimiques et électriques. Ces signaux passent d'un neurone à un autre via le potentiel d'action qui est un évènement durant lequel le potentiel électrique du neurone augmente puis chute brusquement.

On peut qualifier le potentiel d'action par un évènement « tout ou rien ». En effet de la Figure 2.4, on remarque que le potentiel d'action est soit au niveau haut soit au niveau bas, il n'existe pas d'états intermédiaires. De plus, les connexions entre les neurones ont un poids :

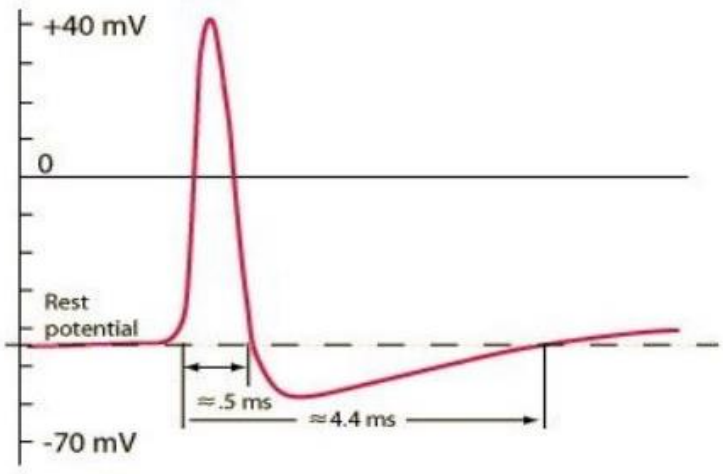


Fig. 2.4 Représentation du potentiel d'action

deux neurones connectés entre eux dont le poids de la connexion est important s'excitent entre eux ce qui permet la propagation du potentiel d'action.

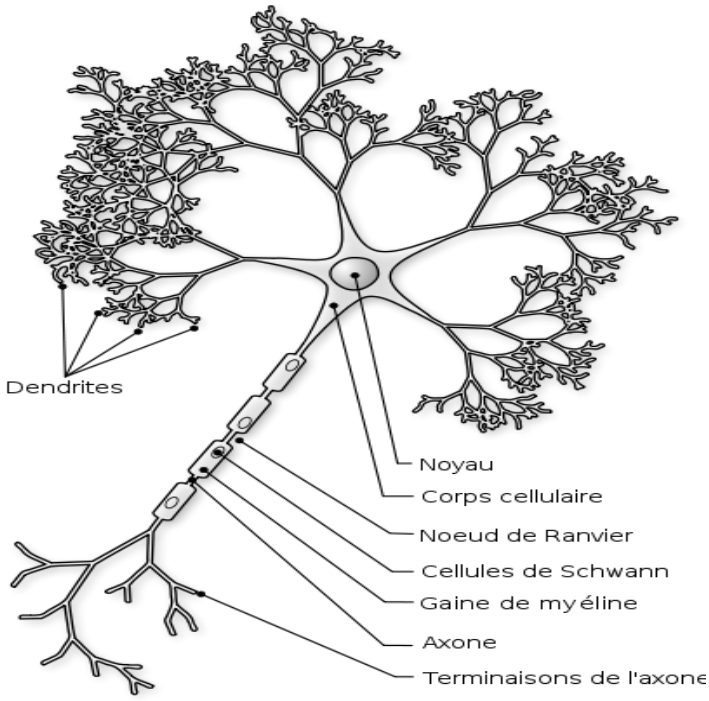


Fig.2.5 Représentation schématique d'un neurone biologique [3]

On déduit qu'un neurone représente une entité logique : il est excité ou bien repos, ce qui fait des neurones un modèle parfait pour résoudre des problèmes dont le résultat est du type binaire ce qu'on appelle communément la classification binaire.

2.2 Modèle formel des réseaux de neurones

D'après le paragraphe précédent, la sortie d'un neurone peut être l'entrée de plusieurs neurones. En outre, ces connexions entre neurones ont des poids différents ce qui contrôle la propagation du potentiel d'action.

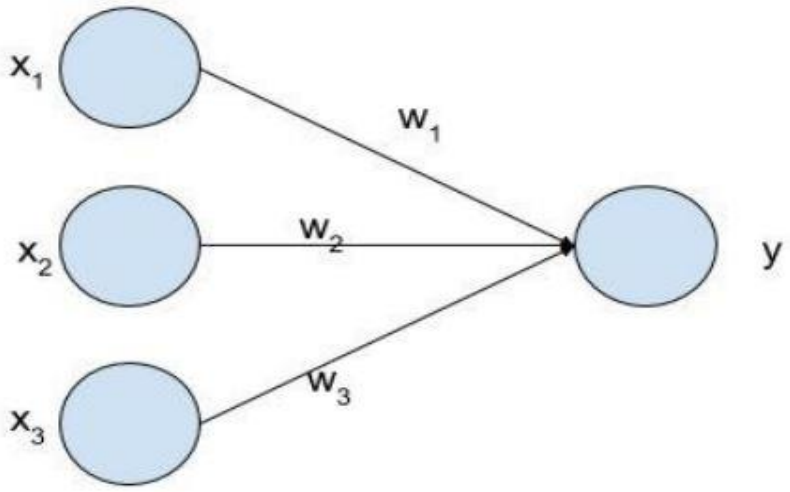


Fig.2.6 Représentation simplifiée d'un neurone connecté à son entrée avec trois neurones via des connexions de poids différents

La Figure 2.6 représente l'un des premiers modèles de neurones artificiels proposé, en 1943, par McCulloch et Pitts. Ce modèle défini la sortie du neurone comme étant une variable y fonction de n variables d'entrée x multipliés par leurs poids (ou paramètres) associés (à la Fig.2.4, n=3):

$$y = f\left(\sum_{1}^{n} w_i x_i\right)$$

Où f est la fonction d'activation où dans le premier modèle proposé, celle-ci était simplement représentée par une fonction seuil :

$$f(x) = \begin{cases} 1 \; si \; x > 0 \\ 0 \; sinon \end{cases}$$

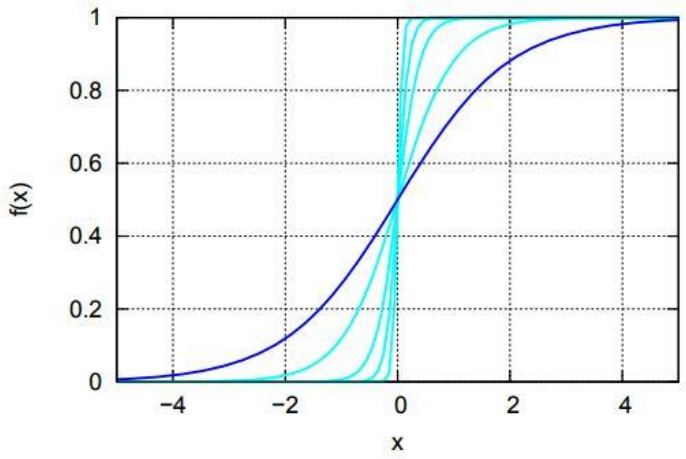
La fonction d'activation défini l'état de basculement du neurone de l'état actif à l'état inactif si l'entrée dépasse le seuil d'activation.

En plus de la fonction d'activation binaire introduite ci-dessus, ils existent d'autres fonctions d'activations entre autres : la fonction sigmoïde (ou de Fermi), la fonction tangente hyperbolique, la fonction linéaire redréssée.

a) **La Fonction Sigmoïde** est défini comme suit :

$$f(x) = \frac{1}{1 + exp^{-x}}$$

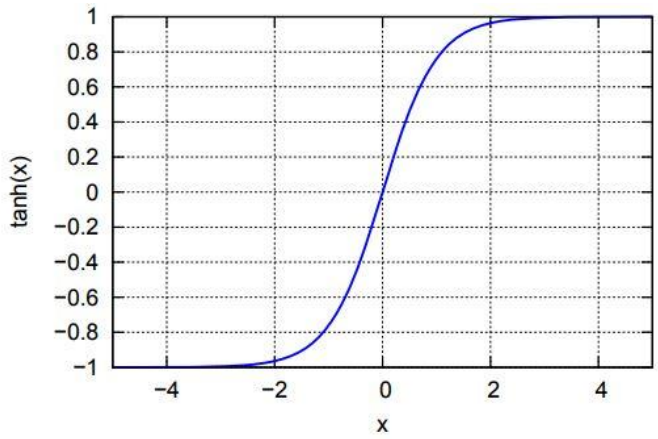
où sa sortie est comprise entre 0 et 1 comme indiqué à la figure ci-dessous :



b) La fonction Tangente Hyperbolique est déterminé par :

$$f(x) = tanhx$$

Sa sortie est comprise entre -1 et 1 comme montre la figure ci-dessous.



c) La fonction linéaire redressée est définie telle que :

$$f(x) = max(0, x)$$

2. 3 Les principaux types de réseaux de neurones

Après avoir étudié les éléments qui constituent un réseau de neurone, on va présenter dans cette section les topologies les plus courantes des réseaux de neurones artificiels.

- **Les réseaux « Feedforward » :** ce type des réseaux est constitué d'une couche d'entrée, n couches cachées et une couche de sortie, de telle sorte qu'un neurone dans une couche a une connexion directe avec tous les neurones de la couche suivante.

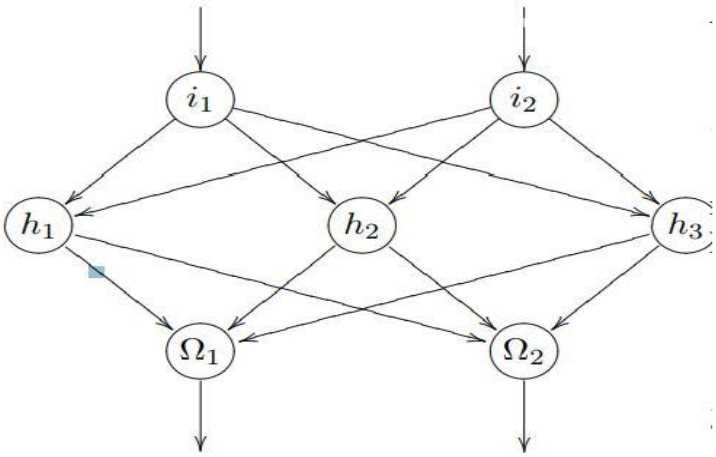


Fig.2.7 Réseau de neurones « feedforward »

- **Les réseaux récurrents :** dans ce type des réseaux, les neurones s'influence sur eux même. En plus, on ne peut pas préciser les couches d'entrée et de sortie explicitement. On distingue trois types des réseaux de neurones récurrents, les réseaux à récurrence directe où les connexions commencent et se terminent sur le même neurone (Fig 2.8) ce qui pousse les neurones à atteindre leur limite d'activation.

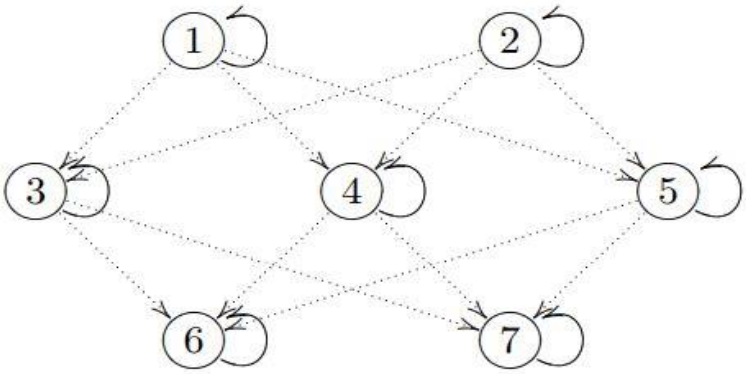


Fig.2.8 Réseau de neurones à récurrence directe

Un deuxième type des réseaux de neurones récurrents concerne ceux à récurrence indirecte, où chaque récurrence influence le neurone de départ via un détour par exemple, un neurone *j* influence les neurones de la couche suivante, et ces derniers influencent le neurone *j*.

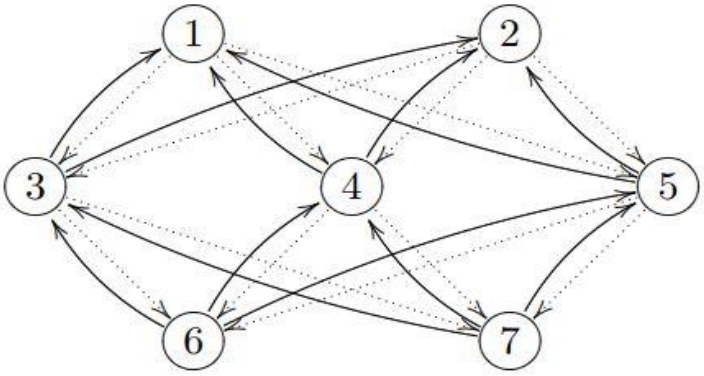


Fig.2.9 Réseau de neurones à récurrence indirecte

Enfin, le troisième type des réseaux de neurones récurrents se distingue par une récurrence latérale, où chaque neurone peut avoir une connexion avec un neurone de la même couche (Fig.2.10).

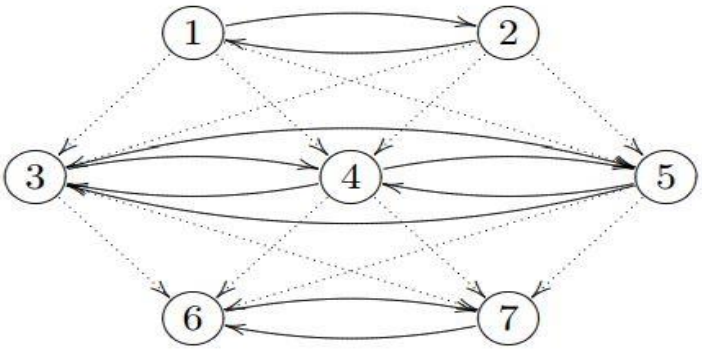


Fig.2.10 Réseau de neurones à récurrence latérale

- **Les réseaux complètement connectés** dans ce type de réseaux, un neurone peut se connecter à n'importe quel neurone dans le réseau sauf lui-même (il ne y'a pas une récurrence directe dans le réseau) :

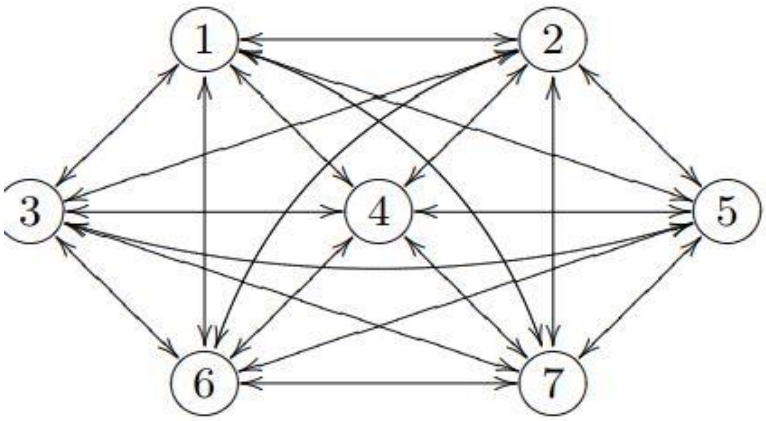


Fig.2.11 Réseau de neurones complètement lié

2.4 Apprentissage d'un réseau de neurones

Le réseau de neurone change lorsqu'un de ces composants change, donc on peut dire que l'apprentissage d'un réseau de neurone se fait en :

- Développant de nouvelles connexions
- Supprimant des connexions
- Changeant les poids des connexions
- Changeant les seuils d'activations des neurones
- Variant les fonctions d'activation des neurones
- Générant des nouveaux neurones
- Supprimant des neurones existants

Généralement pour effectuer l'apprentissage d'un réseau de neurones, on procède par un changement des poids des connexions durant le processus d''apprentissage.

3 Algorithme du perceptron

Pour entrainer un réseau de neurones à effectuer une tache de classification, Rosenblatt avait proposé l'algorithme du perceptron [4]. L'entrainement de neurone artificiel est équivalent au calcul des paramètres P_i par rapport à un ensemble de données d'entrainement à caractère binaire. Cet algorithme calcule les nouveaux paramètres à chaque itération. Soit r^j une donné réelle de base, on peut définir les étapes de l'algorithme comme suit :

i) Initialisation des paramètres w_i aléatoirement

ii) Calcule de la sortie relative à j^{ième} entrée (x^j) , soit P^j la sortie prédite :

$$P^j = f\left(x^j\right)$$

iii) Les poids sont mis à jour en fonction de la différence entre la sortie prédite et la donné réelle :

$$w_i(t+1) = w_i(t) + \left(r^j - P^j\right){x^j}_i$$

L'espace des fonctions que l'algorithme de perceptron peut simuler est limité, ceci a conduit à chercher de nouveaux algorithmes plus efficaces.

4 Algorithme de la Backpropagation

Pour remédier au problème de limitation de l'algorithme du perceptron, une autre architecture de réseau de neurones à plusieurs couches a été proposée.

Comme le montre la Figure 2.12, cette nouvelle architecture est composée de succession de plusieurs couches : une couche d'entrée, une couche cachée et une couche de sortie. Chaque couche est composée de N_i neurones dont l'entrée est la sortie des N_{i-1} neurones présents sur la couche précédente.

Une fois les données sont injectées au niveau de la couche d'entrée, le modèle du réseau calcule l'activation des neurones dans toutes les couches. Le calcul se fait au niveau de chaque neurone où la somme de ses entrées est calculée puis cette valeur passe à travers la fonction d'activation pour produire sa sortie.

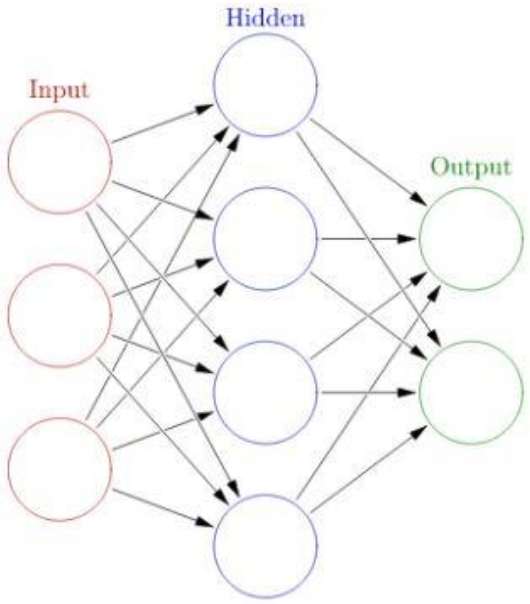


Fig.2.12 Structure d'un réseau de neurones à plusieurs couches

En général, l'activation du neurone x_j^c (le neurone j de la couche$c$) est calculée en fonction de l'activation x_j^{c-1} (du neurone j de la couche$c-1$) à partir de la relation suivante

$$x_j^c = f\left(\sum_{i=1}^{n^{c-1}} x_j^{c-1}\, w_{ij}^{c,c-1}\right)$$

Où :

- $w_{ij}^{c,c-1}$est le poids associé à la connexion entre les neurones x_j^{c-1} et x_j^c
- f est la fonction d'activation
- n^{c-1}est le nombre de neurones sur la couche $c-1$

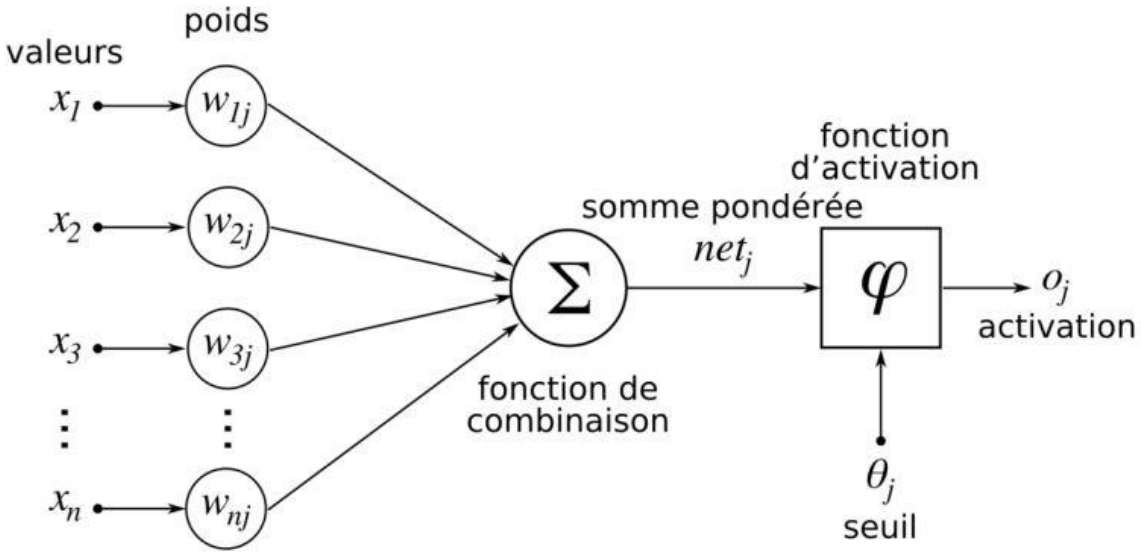


Fig.2.13 Étapes de calcul de l'activation d'un neurone

L'entrainement du modèle consiste à trouver la configuration optimale de poids compte tenu des données d'entrainement. Pour ce faire, on utilise l'algorithme de la « descente du gradient stochastique » car il a été démontré qu'il est le meilleur algorithme d'optimisation pour ce type réseau [5].

L'algorithme de la descente du gradient stochastique « DGS » est basé sur l'actualisation de chaque poids particulier $w_{ij}^{c,c-1}$ itérativement dans le sens opposé à l'évolution du gradient de la fonction de coût L par rapport à ce poids $w_{ij}^{c,c-1}$:

$$\Delta w_{ij} = -\eta \frac{\partial L}{\partial w_{ij}^{c,c-1}}$$

η**:** est le taux d'apprentissage contrôlant la vitesse d'apprentissage. C'est un hyper paramètre (un paramètre dont la valeur est définie avant le lancement de l'apprentissage). La fonction de coût L est aussi un hyper paramètre, une fonction différentiable dont la dérivée partielle $\frac{\partial L}{\partial w_{ij}^{c,c-1}}$ est calculée séquentiellement, couche par couche, de la couche de sortie vers la couche de l'entrée, donc l'erreur est propagée en arrière dans le réseau d'où l'appellation Backpropagartion (propagation arrière). Pour améliorer la convergence de l'algorithme de DGS, d'autres méthodes ont été proposés dont l'optimisateur d'ADAM.

5. Configuration d'un réseau de neurones multi couche

Dans cette section on va présenter les étapes et les règles à suivre afin d'implémenter un réseau de neurones multi couche.

5.1 Le nombre de couches

Dans cette étape, on précise le nombre des couches à utiliser dans le réseau. Généralement, on a au moins trois couches : une d'entrée, une autre de sortie et une couche cachée, ce perceptron à multi couches peut approximer n'importe quelle fonction.

Dans cette étape il faut discuter de la représentabilité du réseau qui est l'aptitude du réseau multi couche à approximer une fonction. Il faut aussi discuter de la capacité d'apprentissage du réseau. D'après les expériences, il a été prouvé que deux ou trois couches cachées sont suffisantes pour respecter les deux conditions de représentabilité et la capacité d'apprentissage car avec une seule couche cachée on tombe souvent sur des problèmes d'impossibilité de capacité d'apprentissage [6].

On commence toujours avec une seule couche cachée et si cela ne convient pas, on ajoute une deuxième couche cachée. Il est à noter que l'ajout des couches cachées augmente les erreurs sub-minimales dans le réseau, mais avec le développement des moyens de calcule, on peut procéder avec des réseaux profonds (deep networks) sans problèmes.

5.2 Le nombres des neurones dans les couches cachées

Au nombre des neurones dans le réseau correspond le nombre des paramètres indépendants du réseau. Le but ultime étant d'avoir un minimum de paramètres indépendants, il est préférable de commencer avec un nombre réduit de neurones et l'augmenter à chaque entrainement du réseau jusqu'à ce qu'on obtienne les résultats désirés.

5.3 Sélection de la fonction d'activation

Pour les neurones de la couche d'entrée, la fonction d'activation est l'identité car ces neurones d'entrée ne traitent pas les informations.

Pour la couche de sortie, la fonction d'activation peut être la même que celle des neurones des couches cachées comme elle peut être différente. Par exemple, on peut fixer la fonction tangente hyperbolique pour les couches cachées et la fonction linéaire redressée pour la couche de sortie.

L'activation des neurones de la couche de sortie peut causer des problèmes d'optimisation (saut des minimums locaux par exemple), on peut limiter ces problèmes en fixant un très faible taux d'apprentissage pour les neurones de la couche de sortie.

5.4 Initialisation des poids des connexions

Avant de commencer le processus d'apprentissage, on doit initialiser les poids du réseau avec de petites valeurs aléatoires car si on initialise avec 0 les poids ne changeraient jamais de plus, si on initialise avec les mêmes valeurs, toutes les connexions auraient les mêmes poids à chaque itération.

Il est préférable de choisir des valeurs aléatoires dans les intervalles $[-0.5 \quad 0[$ et $]0 \quad +0.5]$. Ce choix des petites valeurs aléatoires est avantageux dans le fait qu'il permet d'avoir des impulsions d'apprentissage dès le départ car les valeurs proches de 0 permettent d'achever des dérivations plus grandes [6].

6. L'apprentissage profond (Deep learning)

L'apprentissage profond est une classe de techniques (méthodes) de l'apprentissage automatique exploitant plusieurs couches pour le traitement de l'information. Ce traitement s'appuie sur des fonctions non linéaires pour effectuer les taches d'apprentissage supervisé ou non.

L'apprentissage profond est basé sur un apprentissage à plusieurs niveaux de représentation des données, à travers les couches du réseau profond. En effet, on passe des paramètres de bas niveau aux paramètres de plus haut niveau, de telle sorte qu'à chaque niveau est apprise une abstraction ou bien une représentation plus compliquée que la précédente.

6.1 Les techniques (modèles) d'apprentissage profond

Plusieurs techniques d'apprentissage profond ont été développées pour faire face aux problèmes rencontrés dans le processus d'apprentissage, ces techniques sont classées selon le type d'apprentissage, supervisé et non supervisé.

- **Les modèles discriminants profonds (Deep descriminative models) :**

Cette classe comprend :

i. **Les réseaux de neurones profonds.** Les réseaux de neurones profonds sont des réseaux multi couches, dont les neurones sont complètement liés. Leurs poids sont initialisés avec un prétraitement supervisé ou non supervisé.

ii. **Les réseaux de neurones récurrents :** dans ce type de réseau les neurones ont un retour sur leur état donc chaque neurone s'auto-influence.

iii. **Les réseaux de neurones de convolution:** Cette architecture se distingue par la présence des couches de convolution (voir Chapitre 3 pour plus de détails).

- **Les modèles génératifs et non supervisés (Deep generative/unsupervised models) :**

Cette classe comprend :

i) Machine de Boltzmann restreinte : une machine de Boltzmann est un réseau à neurones liés symétriquement dont la fonction d'activation est stochastique. Une machine de

Boltzmann restreinte est un type spécial des machines de Boltzmann dont le réseau est constitué d'une couche d'entrée et des couches cachées à noter que les neurones de la même couche ne sont pas liés.

ii) Deep belief networks : c'est un modèle probabiliste composé de plusieurs couches cachées avec des variables stochastiques. Les deux premières couches ont des connexions symétriques entre eux, et les dernières couches ont des connexions top-down dont chaque couche a une connexion directe avec la couche en amont.

6.2 Développement de l'apprentissage profond

Simuler les mécanismes du traitement de l'information humaine (vision, audio, traitement de la parole..) par une machine exige l'utilisation des architectures profondes pour l'extraction des informations complexes et construire une représentation interne des informations. Par exemple, le système visuel humain est hiérarchique par nature.

L'entrainement des réseaux de neurones artificiels ayant plusieurs couches cachées a posé plusieurs problèmes comme la recherche du meilleur algorithme qui peut trouver les paramètres (les poids) optimaux en un temps minimum. Hinton et al. [5], ont proposé un algorithme itératif pour l'apprentissage basé sur un modèle à plusieurs couches avec l'introduction des couches de la machine de Boltzmann restreinte qui est couramment utilisée pour avoir une estimation de la distribution probabiliste d'un jeu de données. Ces changement ont permis au modèle profond d'apprendre les représentations complexes hiérarchiquement et d'une manière efficace car chaque couche peut apprendre une représentation plus réduite et moins redondante que la couche précédente.

En effet, la profondeur du réseau de neurones (le nombre des couches cachées) est devenue un critère essentiel pour avoir plus de performances. Ce développement de l'apprentissage profond a motivé l'introduction d'une nouvelle architecture, notamment : le réseau de neurones de convolution. Ce dernier a été développé pour faire face aux données qui ont une certaine topologie spatiale comme l'image, la voix, la vidéos...etc. En 2012, Krizhevsky et al [7] ont réussi à avoir une erreur de 15.3% lors de la compétition ImageNet (l'olympiade de la reconnaissance des images) et depuis lors, l'apprentissage profond en général et les réseaux de neurones de convolution ont largement contribué, ces dernières années, à un rapide essor de l'intelligence artificiel.

6.3 L'utilisation des GPU pour l'apprentissage profond

L'un des facteurs les plus importants qui ont contribués ay*u développement de l'apprentissage profond est la computation parallèle caractérisée par l'utilisation des processeurs graphiques (GPGPU : General Purpose Graphical Processing Unit) dans la phase d'apprentissage à cause de leurs performances et rapidité par rapport aux unités de traitement classiques comme les CPU. La caractéristique majeure des GPGPU est leur parallélisme qui est une opération qui consiste à mettre en œuvre des architectures d'électronique numérique permettant de traiter les informations de manière simultanée ce qui permet d'effectuer des milliers d'opérations de multiplication matricielles en temps minimal.

Dans notre projet qui consiste à utiliser l'approche end to end deep learning, on a opté d'utiliser cette technologie des GPGPU pour le traitement durant la phase d'apprentissage. Pour ce faire, on a fait appel au CUDA (Computer Unified Device Architecture) qui est une technologie des GPGPU développée par NVIDIA qui permet de programmer des GPU pour effectuer un calcul intensif.

On a utilisé pour l'apprentissage un ordinateur avec une carte graphique NVIDIA GEFORCE GTX 960m qui a une mémoire de 4GB et une vitesse de traitement des données de 1260 GHz.

6.4 Les outils de programmation

Après l'installation des outils de CUDA afin de bénéficier des performances de la carte graphique dans la phase d'apprentissage de réseau de neurones, un appel aux bibliothèques de programmation dédiées aux applications de l'intelligence artificielle a été fait. Les bibliothèques comme KERAS et TENSORFLOW programmées avec le langage PYTHON rendent la mise en œuvre des réseaux de neurones plus simple.

- **Python :** est un langage de programmation objet largement utilisé dans le domaine de l'apprentissage profond à cause de sa simplicité et la possibilité d'implémenter des fonctions mathématiques complexes en quelques lignes de code.
- **Tensorflow :** est une bibliothèque open-source développée par l'équipe Google Brain qui l'utilisait initialement en interne. Elle implémente des méthodes d'apprentissage automatique basées sur le principe des réseaux de neurones profonds

- **Keras :** est une librairie Python qui encapsule l'accès aux fonctions proposées par plusieurs librairies de l'apprentissage automatique, en particulier Tensorflow. De fait, Keras n'implémente pas nativement les méthodes. Elle sert d'interface avec Tensorflow simplement.

7 Conclusion

Ce chapitre a été consacré à une brève introduction aux réseaux de neurones artificiels et leurs apprentissages automatiques. Nous avons présenté, dans une première partie de ce chapitre, les généralités relatives aux bases de l'apprentissage automatique où nous avons particulièrement mis l'accent sur l'apprentissage profond.

En deuxième partie, nous avons exposés l'essentiel sur les réseaux de neurones artificiels où nous avons présentés les éléments suivants : le modèle formel d'un neurone, les structures courantes des réseaux de neurones et la descente du gradient stochastique.

Une troisième partie de ce chapitre a été consacrée à une présentation plus détaillée de la méthode d'apprentissage par l'algorithme de la Backpropagation et des techniques utilisées dans l'apprentissage profond.

La dernière partie a été consacrée aux outils informatiques (unité de traitement et outils de programmation) qu'on va utiliser dans la suite de ce mémoire afin de mettre en œuvre les résultats théoriques.

Chapitre III
Réseaux de neurones de convolution

De nos jours, les réseaux de neurone de convolution aussi appelés ConvNets sont partout. C'est sans doute l'architecture d'apprentissage profond la plus populaire. La récente vague d'intérêt pour l'apprentissage profond est due à l'immense popularité et l'efficacité des convnets.

Les ConvNets sont maintenant le modèle incontournable pour tous les problèmes liés aux images. En termes de précision, ils sont les plus fiables. Ils sont également appliqués avec succès aux systèmes de recommandation, au traitement du langage naturel et plus encore. Le principal avantage des ConvNets par rapport à leurs prédécesseurs est qu'ils détectent automatiquement les caractéristiques importantes sans aucune surveillance humaine. Par exemple, étant donné de nombreuses photos de chats et de chiens, ils apprennent des caractéristiques distinctives pour chaque classe par eux même.

1 Introduction aux réseaux de neurones de convolution

Les réseaux de neurones de convolution (ConvNets ou CNN) sont des réseaux neuronaux artificiels profonds utilisés principalement pour classer des images (par exemple, nommer ce qu'ils voient), les regrouper par similarité (recherche de photos), et effectuer une reconnaissance d'objet dans des scènes. Ce sont des algorithmes qui permettent d'identifier les visages, les individus, les signes de rue, les tumeurs, les ornithorynques et de nombreux autres aspects des données visuelles.

Les réseaux de convolution peuvent également effectuer la reconnaissance optique de caractères (ROC) pour numériser du texte et rendre possible le traitement en langage naturel sur des documents analogiques et manuscrits, où les images sont des symboles à transcrire. Les CNN peuvent également être appliqués au son lorsqu'il est représenté visuellement sous forme d’un spectrogramme. Plus récemment, les réseaux de convolution ont été appliqués directement à l'analyse textuelle ainsi qu'aux données graphiques.

L'efficacité des réseaux de convolution dans la reconnaissance d'image est l'une des principales raisons pour lesquelles le monde s'est réveillé à l'efficacité de l'apprentissage profond. Ce dernier a permis des avancées majeures dans le domaine de la vision par ordinateur (CV), laquelle a des applications évidentes pour les voitures autonomes, la

robotique, les drones, la sécurité, les diagnostics médicaux et les traitements pour les malvoyants.

1.2 Historique des réseaux de neurones de convolution

- En 1957, Frank Rosenblatt a développé la machine Mark I Perceptron, qui fut la première implémentation de l'algorithme du perceptron. La machine était connectée à une caméra qui utilisait des cellules photoélectriques 20 × 20 pour produire une image de 400 pixels. L'idée consistait à calculer une fonction de score en utilisant la fonction suivante :

$$f(x) = \begin{cases} 1 & si\ w.x + b > 0 \\ 0 & ailleurs \end{cases}$$

Les sorties sont soit zéro soit un, avec une règle de mise à jour des poids w_i. Cette règle consistait à prendre des poids initiaux puis les ajuster dans la direction de la cible.

- En 1959, Hubel et Wiesel, deux Biologistes qui cherchaient à comprendre le mécanisme de traitement visuel chez les mammifères, ont donc choisi d'étudier le cerveau d'un chat qui est plus ou moins similaire au cerveau humain d'un point de vue du traitement visuel. L'expérience consistait à coller des électrodes dans l'arrière du cerveau du chat qui est l'endroit où est située la zone du cortex visuel primaire et d'observer la réponse à des stimuli. L'expérience a montré que les neurones dans l'arrière du cortex répondent fortement. Cette expérience leur a permis de savoir qu'il existe de nombreux types de cellules dans cette partie du cerveau mais que la plupart d'entre elles sont de type simple. Celles-ci répondent en présence de bords orientés se déplaçant dans certaines directions. Les deux chercheurs ont surtout découvert que le traitement visuel commence par une structure simple du monde visuel (les bords orientés) puis, l'information suit le processus du traitement visuel et c'est le cerveau en fin de compte qui construit la complexité de l'information visuelle jusqu'à ce qu'il puisse restituer le monde visuel complexe.

- En 1986, le principe de la backpropagation fut introduit pour la première fois par Rumelhart où des équations avec des règles de mise à jour et la règle de la chaine apparurent. De ce fait, on dispose pour la première fois d'une méthode pour entraîner les réseaux de neurones. Toutefois, cette avancée dans le domaine de l'apprentissage, était insuffisante pour

prendre en charge le problème de l'apprentissage des réseaux de neurones de grande taille, et ainsi on observa une période de latence quant à l'utilisation de ce type de réseaux.

- En 2006, Geoff Hinton et Ruslan Salakhutdinov avaient publié un article où ils montraient qu'il est possible d'entraîner efficacement un réseau de neurones profond. En effet, la première étape était un pré-entraînement. Ce dernier consistait à modéliser chaque couche cachée à travers une machine de Boltzmann restreinte, et ainsi il fut possible d'obtenir une initialisation des poids de ces couches cachées en entraînant d'une manière itérative chacune d'elle. Puis, une fois passée l'étape d'initialisation de toutes les couches cachées du réseau complet, il était alors possible d'appliquer la backpropagation cependant il faudrait une initialisation très prudente afin de pouvoir exploiter la backpropagation.

- En 2012, Alex Krizhevsky présenta la première architecture de réseau de neurones de convolution (ConvNets) apte à fournir de très bons résultats concernant la classification d'image « ImageNet » et a été en mesure de réduire considérablement l'erreur de classification. Depuis lors, les CNN ont largement investi toutes sortes d'applications.

1.3 Bref Aperçu sur l'architecture des réseaux de neurones de convolution

- **Les réseaux de neurones standards**

Comme nous l'avons vu dans le chapitre précédent, les réseaux de neurones standards reçoivent une entrée (sous forme d'un seul vecteur) et la transforment à travers une série de couches cachées. Chaque couche cachée est constituée d'un ensemble de neurones, où chaque neurone est entièrement connecté à tous les neurones de la couche précédente de plus, les neurones sur une même couche fonctionnent de manière totalement indépendante et ne partagent aucune connexion. La dernière couche entièrement connectée est appelée «couche de sortie» dont les valeurs des paramètres de sortie représentent des scores d'appartenance d'un ou plusieurs objets à une classe de classification.

En outre, ces réseaux de neurones standards (ou conventionnels) ne sont pas bien adaptés au traitement des images complètes. En effet, si on considère le réseau de traitement d'image CIFAR-10, pour ce réseau, les images sont uniquement de taille 32x32x3 pixels (32 larges, 32 hautes, 3 canaux de couleur), donc un seul neurone entièrement connecté dans une première couche cachée d'un réseau neuronal standard aurait 32 * 32 * 3 = 3072 poids. Ce nombre de poids semble encore gérable, mais il est clair que cette structure entièrement connectée ne

s'adapte pas à des images de plus haute résolution. Par exemple, une image de taille plus respectable serait de 200x200x3 pixels, et conduirait à un neurone ayant 200 * 200 * 3 = 120 000 poids. De plus dans un réseau, il a y plus d'un neurone (peut-être même des milliers), ce qui conduit à un nombre explosif de paramètres ! Cette connectivité totale est gaspilleuse et ce grand nombre de paramètres conduirait rapidement à un sur-apprentissage.

- **Réseaux de neurones en volumes 3D**

Les réseaux de neurones de convolution sont spécialement conçus pour le traitement d'images ceci contraigne certainement leur architecture. En particulier, contrairement à un réseau neuronal conventionnel, les couches d'un ConvNet ont des neurones disposés selon les 3 dimensions: largeur, hauteur et profondeur. Ici, Il y a lieu de noter que le mot profondeur renvoie à la troisième dimension d'un volume d'activation, et non pas à la profondeur d'un réseau neuronal complet, qui lui renvoie au nombre total de couches dans un réseau. Par exemple, les images d'entrée pour le

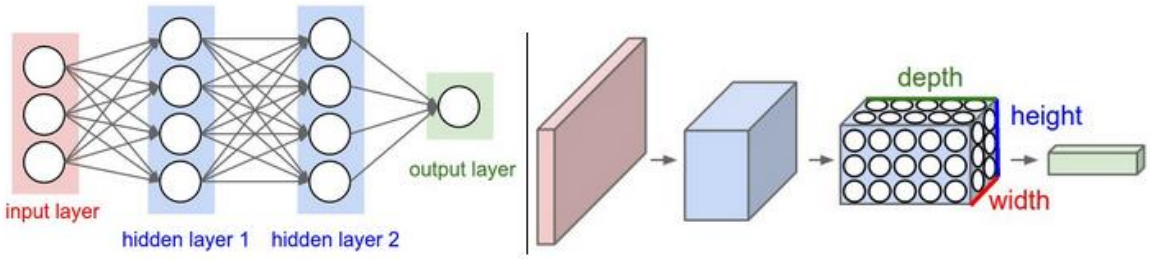


Fig.3.1 Réseaux de neurones standard et réseau ConvNets. À gauche: un réseau de neurones standard à trois couches. A droite : un ConvNet

CIFAR- 10 constituent un volume d'entrée d'activation, et ce volume a les dimensions suivantes : 32x32x3 (largeur, hauteur et profondeur respectivement). Comme nous le verrons dans les paragraphes à suivre, les neurones d'une couche ne seront connectés qu'à une petite région de la couche d'avant, appelée champ réceptif, au lieu qu'ils soient complètement connectés. De plus, la couche de sortie finale pour CIFAR-10 aurait des dimensions 1x1x10, car à la fin de l'architecture ConvNet, nous réduirons l'image complète en un seul vecteur de scores de classe, disposés le long de la dimension de la profondeur.

A titre de comparaison, la Figure 3.1 illustre la structure d'un réseau conventionnel et celle d'un réseau ConvNets. Il est clair que dans un réseau ConvNets, les neurones sont disposés selon les trois dimensions (largeur, hauteur et profondeur), tels que visualisés dans l'une des

couches. Chaque couche d'un ConvNet transforme donc un volume d'entrée 3D en un volume de sortie 3D avec l'activation de certains neurones (équivalent à une couche des réseaux standards). Dans cet exemple, la couche d'entrée rouge contient l'image, donc sa largeur et sa hauteur seraient les dimensions de l'image, et la profondeur serait 3 (pour les trois canaux : rouge, vert, bleu).

2. Les types de couches constituants les ConvNets

2.1 Préliminaire

Comme nous l'avons décrit ci-dessus, un réseau ConvNet est une séquence de couches, et chaque couche d'un réseau ConvNet transforme un volume d'activation en un autre via une fonction différentiable. Nous utilisons principalement quatre types de couches pour construire les architectures ConvNet: la couche d'entrée (INPUT layer), la **Couche de convolution** (Convolutional layer), la **Couche de mise en commun** (Pooling layer) et la **Couche entièrement connectée** (Fully-Connected layer). Puis, exactement comme dans les réseaux de neurones standards, nous allons empiler ces couches pour former une architecture ConvNet complète.

Nous allons détailler la question relative à l'architecture d'un ConvNet en exposant l'exemple relatif au cas d'une simple classification à base d'un ConvNet du type CIFAR-10. L'architecture de ce réseau pourrait avoir la structure suivante : [INPUT - CONV - RELU - POOL - FC].

Nous explicitons ci-dessous le rôle de chaque couche.

- La couche INPUT par exemple de taille [32x32x3] contient les valeurs brutes des pixels de l'image et, dans le cas de l'exemple cet image admet une largeur de 32 pixels, une hauteur de 32 pixels de plus, elle possède trois canaux de couleur R, G, B (red, green,blue).
- La couche CONV calcule la sortie des neurones connectés aux régions locales de l'entrée, chacun calculant un produit scalaire entre leurs poids et une petite région à laquelle ils sont reliés dans le volume d'entrée. Cela peut conduire à un volume tel que [32x32x10] si nous avons décidé d'utiliser 10 filtres.
- La couche RELU applique une fonction d'activation élémentaire seuillage à zéro :***max (0, x).*** Cette opération laisse la taille du volume inchangé ([32x32x12]).

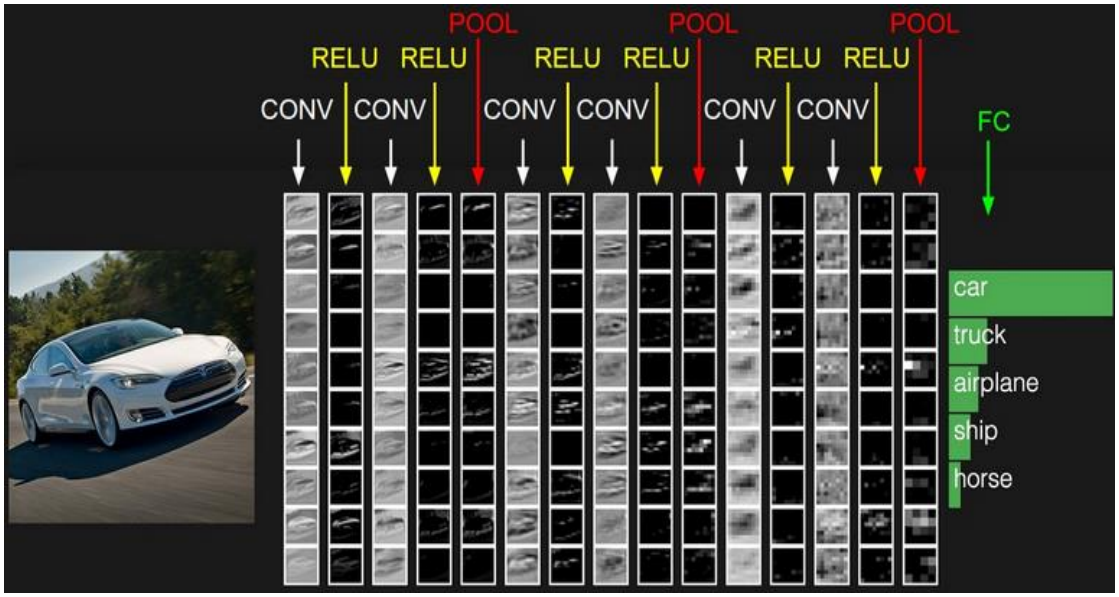


Fig.3.2 Représentation d'un ConvNet sous forme de colonne de volume

- La couche POOL effectue une opération de sous-échantillonnage le long des dimensions spatiales (largeur et hauteur), résultant en un volume plus réduit de taille [16x16x12].
- La couche FC (c'est-à-dire entièrement connectée) calcule les scores de classe, résultant en un volume de taille [1x1x10], où chacun des 10 valeurs correspond à un score de classe. Comme avec les réseaux de neurones ordinaires et comme son nom l'indique, chaque neurone de cette couche est connecté à tous les neurones d'un volume précédent.

Doté de cette structure, un ConvNets transforme l'image d'origine (les valeurs de pixels d'origine) couche par couche en une partition de classe finales. Il y a lieu de noter que certaines couches contiennent des paramètres et d'autres pas. En particulier, les couches CONV/FC effectuent des transformations dépendantes non seulement des activations relatives au volume d'entrée mais aussi, des paramètres (les poids et les biais des neurones). Par contre, les couches RELU/POOL implémentent une fonction fixe. Les paramètres des couches CONV/FC sont déterminés par apprentissage sur la base de l'algorithme de la descente du gradient de sorte que les scores de classe générés par le ConvNet soient cohérents avec les étiquettes de l'apprentissage pour chaque image.

La Figure 3.2 illustre un réseau ConvNet sous la forme de colonnes de volume. Le volume d'entrée (1^{ière} colonne) stocke les pixels de l'image brute (à gauche) et le dernier volume (dernière colonne) stocke les scores de classe (à droite). Chaque volume d'activation le long du chemin du traitement est affiché sous la forme d'une colonne. Puisqu'il est difficile de

visualiser les volumes 3D, les tranches de chaque volume ont été découpées en rangées. Le volume de la dernière couche contient les scores de chaque classe, mais ici nous avons visualisés uniquement les scores du top 5 triés et nous imprimons les étiquettes de chacun.

2.2 Couche de convolution

2.2.1 Structure

La couche Conv est le bloc de construction principal d'un réseau de convolution. Il effectue la plus grande partie de la charge lourde de calcul, son objectif est d'extraire les caractéristiques (*feature maps*) du volume d'entrée.

Tout d'abord, nous entamons l'analyse de la couche CONV sans analogie cerveau/neurones. Les paramètres de la couche CONV consistent en un ensemble de filtres pouvant être appris. Chaque filtre est de dimensions spatiales réduites (le long de la largeur et de la hauteur), mais s'étend sur toute la profondeur du volume d'entrée. Par exemple, un filtre typique sur une première couche d'un ConvNet peut avoir une taille 5x5x3 (c'est-à-dire 5 pixels de largeur et de hauteur, et 3 parce que les images ont une profondeur 3 relative aux canaux de couleur). Pendant le passage vers l'avant (the forward pass), nous faisons glisser (plus précisément, convoluons) chaque filtre sur la largeur et la hauteur du volume d'entrée et calculons les produits scalaires entre les entrées du filtre et l'entrée à n'importe quelle position. Lorsque nous faisons glisser le filtre sur la largeur et la hauteur du volume d'entrée, nous produisons une carte d'activation bidimensionnelle qui donne les réponses de ce filtre à chaque position spatiale. Intuitivement, le réseau renseigne en quelque sorte les filtres pour qu'ils s'activent lorsqu'ils détectent un type de caractéristique visuelle tel qu'un bord d'orientation ou une tache de couleur sur la première couche, ou finalement des motifs en nid d'abeilles ou en forme de roue sur des couches supérieures du réseau. Maintenant, nous aurons un ensemble complet de filtres dans chaque couche CONV (par exemple 12 filtres), et chacun d'eux produira une carte d'activation à 2 dimensions séparée. Nous empilerons ces cartes d'activation le long de la cote de profondeur et ce qui constitue le volume de sortie de la couche Conv.

- **Feature maps**

Comme nous l'avons cité dans le paragraphe précédent, les filtres de la première couche convolutive détectent les formes (features) simples telles que les arêtes et les courbes. Comme on peut l'imaginer, afin de prédire si une image appartient à une classe d'objet, nous avons

besoin que le réseau soit capable de reconnaître des formes plus complexes telles que les mains, les pattes ou les oreilles.

L'entrée de la première couche est l'image originale. Cependant, pour la deuxième couche CONV, l'entrée est la carte d'activation (activation map) qui résulte de la première couche. Ainsi, chaque couche du volume d'entrée décrit essentiellement les emplacements de certaines formes simples dans l'image d'origine. Maintenant, quand on applique un ensemble de filtres sur le dessus (passez le dans la deuxième couche CONV), la sortie sera des activations qui représentent des formes de niveau supérieur (feature maps). Les types de ces caractéristiques peuvent être des demi-cercles (combinaison d'une courbe et d'une arête droite) ou des carrés (combinaison de plusieurs arêtes droites). Lorsqu'on parcourt plusieurs couches CONV, on obtient des cartes d'activation qui représentent des formes de plus en plus complexes. À la fin du réseau, vous pouvez avoir des filtres qui s'activent lorsqu'il y a de l'écriture manuscrite dans l'image, des filtres qui s'activent quand ils voient des objets roses, etc.

Par ailleurs du point de vue du cerveau, chaque partie du volume de sortie 3D peut être interprétée comme une sortie d'un neurone qui pointe une petite région dans l'entrée et partage les paramètres avec tous les neurones à gauche et droite dans l'espace.

A ce stade, nous abordons les détails relatifs à la connectivité des neurones, à leur disposition dans l'espace et leur procédure de partage des paramètres.

2.2.2 Caractérisation d'une Couche de convolution

- **Connectivité locale**

Lorsque nous traitons des entrées de grande dimension telles que les images, comme nous l'avons vu ci-dessus, il est impossible de connecter les neurones d'un volume colonne à tous les neurones du volume précédent. Au lieu de cela, nous allons seulement connecter chaque neurone à une région locale du volume d'entrée. L'étendue spatiale de cette connectivité est un hyper-paramètre appelé le champ réceptif du neurone (de manière équivalente, c'est la taille du filtre). L'étendue de la connectivité le long de l'axe de profondeur est toujours égale à la profondeur du volume d'entrée. Il est important de souligner à nouveau cette asymétrie dans la façon de traiter les dimensions spatiales (largeur et hauteur) et la dimension de profondeur. En effet, les connexions sont locales dans l'espace (largeur et hauteur), mais toujours pleines sur toute la profondeur du volume d'entrée.

Exemple 1

Supposons que le volume d'entrée soit d'une taille de [32x32x3] (par exemple une image CIFAR-10 RGB). Si le champ réceptif (ou la taille du filtre) est 5x5, alors chaque neurone dans la couche de convolution aura des poids à une région de taille [5x5x3] dans le volume d'entrée, pour un total de 5 * 5 * 3 = 75 poids (et +1 paramètre de polarisation). Notez que l'étendue de la connectivité le long de l'axe de profondeur doit être de 3, puisqu'il s'agit de la profondeur du volume d'entrée.

Exemple 2

Supposons qu'un volume d'entrée ait une taille [16x16x20]. Puis, en utilisant par exemple un champ réceptif de taille 3x3, chaque neurone dans la couche de convolution aura maintenant un total de 3 * 3 * 20 = 180 connexions au volume d'entrée. Notez que, de nouveau, la connectivité est locale dans l'espace (par exemple 3x3), mais elle est complète le long de la profondeur d'entrée (20).

La Figure 3.3 montre une structure en volume d'un ConvNet. On peut voir à gauche: un exemple de volume d'entrée en rouge (par exemple une image CIFAR-10 32x32x3), et un exemple de volume de neurones dans la première couche de convolution. Chaque neurone dans la couche de convolution est connecté uniquement à une région locale dans le volume d'entrée l'espace 3D, mais sur toute la profondeur (c'est-à-dire pour tous les canaux de couleur). Notez qu'il y a plusieurs neurones (5 dans cet exemple) le long de la profondeur, tous pointant la même région dans l'entrée (voir la discussion des colonnes de profondeur dans le texte ci-dessous). A droite: les neurones calculent un produit scalaire de leurs poids avec l'entrée suivie d'une non-linéarité, mais leur connectivité est maintenant restreinte pour être locale dans l'espace.

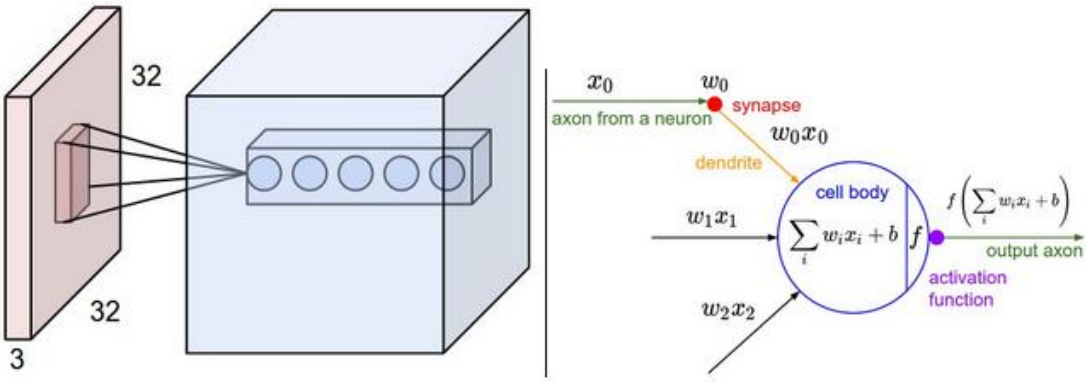


Fig.3.3 Réseaux de Neurones de volume 3D

- **Aménagement d'espace**

Nous avons expliqué la connectivité de chaque neurone de la couche de convolution au volume d'entrée, mais nous n'avons pas encore discuté du nombre de neurones présents dans le volume de sortie ou de leur disposition. Trois hyper-paramètres contrôlent la taille du volume de sortie: **la profondeur** (depth), **le pas** (stride) et **le remplissage par zéro** (zero-padding).

i. La profondeur du volume de sortie est un hyper-paramètre: il correspond au nombre de filtres que nous aimerions utiliser, chacun apprenant à chercher quelque chose de différent dans l'entrée. Par exemple, si la première couche de convolution prend en entrée l'image brute, alors différents neurones le long de la dimension de profondeur peuvent s'activer en présence de divers bords orientés, ou des taches de couleur. Nous nous référerons à un ensemble de neurones qui pointent tous vers la même région de l'entrée en tant qu'une colonne de profondeur (depth column) (certaines personnes préfèrent également le terme fibre).

ii. Nous devons spécifier le pas avec lequel nous faisons glisser le filtre. Lorsque le pas est égal à 1, nous déplaçons le filtre d'un pixel à la fois. Lorsque le pas est égal à 2 (ou exceptionnellement à 3 ou plus, bien que cela soit rare en pratique), le filtre saute 2 pixels à la fois lorsque nous le faisons glisser. Cela conduit à des volumes de sortie dans l'espace plus réduits.

iii. Par ailleurs, il serait parfois pratique de remplir le volume d'entrée avec des zéros autour de la bordure. La taille de ce zéro-padding est aussi un hyper-paramètre. La caractéristique intéressante de ce remplissage se manifeste par un contrôle de la taille spatiale des volumes de sortie (le plus souvent comme nous le verrons, nous allons l'utiliser pour que la taille spatiale du volume d'entrée soit exactement identique à celle du volume de sortie).

Il est possible de prévoir la taille spatiale du volume de sortie en fonction de la taille du volume d'entrée (**W**), de la taille du champ réceptif des neurones de la couche de convolution (**F**), du pas de glissement appliqué (**S**), et enfin en fonction du nombre (**P**) de zéro de remplissage (zero padding) utilisé sur la bordure. La taille du volume de sortie selon la largeur et la hauteur est obtenue en nombre de neurones « équivalents » par la relation suivante :

$$(W-F + 2P) / S + 1$$

Par exemple, pour une entrée 7x7 et un filtre 3x3 avec un pas égale à 1 et un zero-padding de 0, nous obtiendrions dans ce cas (pour W=7, F=3, P=0 et S=1) une sortie 5x5. Avec les mêmes données précédentes et un pas de 2, nous aurions une sortie 3x3.

- **Exemple d'utilisation du zero-padding**

On considère un réseau de taille en entrée W = 5x5 ayant un champ réceptif de taille F = 3x3 et on utilise comme zéro-padding P=1.

i. Si le champ réceptif balaye l'entrée avec un pas S = 1, on obtient en sortie selon l'axe des x une taille équivalente de (5 - 3 + 2) / 1 + 1 = 5 (idem selon l'axe des y).

ii. Si le champ réceptif balaye l'entrée avec un pas S = 2, on obtient en sortie selon l'axe des x une taille équivalente à (5 - 3 + 2) / 2 + 1 = 3 (idem selon l'axe des y).

Il y a lieu de noter que le pas S = 3 ne peut être utilisé car il n'entrerait pas parfaitement dans le volume. En termes d'équation, ceci peut être déterminé puisque (5 - 3 + 2) = 4 n'est pas divisible par 3.

Dans l'exemple ci-dessus, pour une dimension à l'entrée de 5x5, la dimension de sortie était également de 5x5. Cela a bien fonctionné car le champ réceptif était de 3x3 et nous avons utilisé un zero-padding de 1. S'il n'y avait pas de zero-padding, alors le volume de

sortie aurait seulement une dimension spatiale de 3, parce que c'est le nombre de neurones qui correspondrait à l'entrée d'origine.

En général, pour garantir au volume de sortie d'avoir la même taille que le volume d'entrée, on impose un zero-padding, pour un pas S est égale à 1, donné par :

$$P = (F-1) / 2$$

Notez encore que les hyper-paramètres d'arrangement spatial ont des contraintes mutuelles. Par exemple, lorsque l'entrée a la taille W = 10, aucun zero-padding n'est utilisé P = 0, et la taille du filtre est F = 3, alors il serait impossible d'utiliser le pas S = 2, puisque (W- F + 2P) / S + 1 = (10- 3+ 0)/2+1=4.5, n'est pas un nombre entier, indiquant que les neurones ne "s'ajustent" pas de façon nette et symétrique à travers l'entrée. Par conséquent, cette valeur de l'hyper-paramètre est considérée comme invalide. Comme nous le verrons dans la section sur les architectures ConvNet, le dimensionnement approprié des ConvNets, assurant un bon ajustement de toutes les dimensions peut être difficile à obtenir, d'où le recours à l'emploi du zero-padding.

- **Partage de paramètres.**

La technique de partage des paramètres est utilisée dans les couches de convolution pour contrôler le nombre de paramètres. Si on prend par exemple un volume de sortie de la première couche de convolution de taille [55x55x96] qui utilise une taille du champ réceptif de [11x11x3], nous voyons qu'il y a 55 * 55 * 96 = 290 400 neurones , et chacun a 11 * 11 * 3 = 363 poids et 1 biais. Ensemble, cela équivaut à 290400 * 364 = 105 705 600 paramètres sur la première couche du ConvNet seul. Il est évident que ce nombre est très élevé.

Il s'avère que nous pouvons réduire considérablement le nombre de paramètres en faisant l'hypothèse raisonnable suivante: si une caractéristique est utile pour le calcul relatif à une position spatiale (x, y), alors il devrait être aussi utile pour le calcul relatif à une position différente (x2, y2). En d'autres termes, en indiquant une tranche de profondeur bidimensionnelle comme une tranche de profondeur (par exemple un volume de taille [55x55x96] à 96 tranches de profondeur, chacune de taille [55x55]), nous allons contraindre les neurones dans chaque tranche de profondeur à utiliser les mêmes poids et biais. Avec ce schéma de partage de paramètres, la première couche de convolution, dans notre exemple,

aurait seulement 96 ensembles de poids uniques (un pour chaque tranche de profondeur), pour un total de 96 * 11 * 11 * 3 = 34 848 poids uniques, soit 34 944 paramètres (+96 biais). Alternativement, tous les 55 * 55 neurones dans chaque tranche de profondeur vont maintenant utiliser les mêmes paramètres. En pratique pendant la rétro-propagation, chaque neurone dans le volume calculera le gradient pour ses poids, mais ces gradients seront additionnés à travers chaque tranche de profondeur et ne mettront à jour qu'un seul ensemble de poids par tranche.

Notez que si tous les neurones dans une seule tranche de profondeur utilisent le même vecteur de poids, le passage vers avant de la couche convolutive peut dans chaque tranche de profondeur être calculée comme une convolution des poids du neurone avec le volume d'entrée (d'où le nom: Convolutional Layer). C'est pourquoi il est courant de se référer aux ensembles de poids comme un filtre (ou un noyau), qui est convolué avec l'entrée.

Notez que parfois l'hypothèse de partage de paramètres peut ne pas avoir de sens. C'est particulièrement le cas lorsque les images d'entrée vers un réseau ConvNet ont une structure centrée spécifique, où l'on devrait s'attendre, par exemple, à ce que des caractéristiques complètement différentes soient apprises d'un côté de l'image par rapport à un autre. Un exemple pratique est lorsque les entrées sont des visages qui ont été centrées dans l'image. Vous pourriez vous attendre à ce que différentes caractéristiques propres aux yeux ou aux cheveux puissent (et devraient) être apprises dans différents endroits de l'espace. Dans ce cas, il est courant de relâcher le schéma de partage de paramètres et, au lieu de cela, d'appeler simplement la couche une couche localement connectée.

En résumé, la couche de convolution :

Accepte un volume de taille $W_1 \times H_1 \times D_1$

Nécessite quatre hyper-paramètres:

- Nombre de filtres K
- Leur étendue spatiale F
- Le pas S
- Le nombre de zero-padding P

Produit un volume de taille $W_2 \times H_2 \times D_2$ où:

- $W_2 = (W_1\text{-}F + 2P) / S + 1$
- $H_2 = (H_1\text{-}F + 2P) / S + 1$ (c'est-à-dire que la largeur et la hauteur sont calculées également par symétrie)
- $D_2 = K$

Avec le partage de paramètres, il introduit $F*F*D_1$poids par filtre, pour un total de $(F*F*D_1)*K$ poids et K biais.

Dans le volume de sortie, la n-ième tranche de profondeur (de taille $W_2 \times H_2$) est le résultat de la réalisation d'une convolution valide du n-ième filtre sur le volume de l'entrée avec un pas S, puis compensée par n-ième biais.

En général, les valeurs courantes des hyper-paramètres sont tels que : F = 3, S = 1, P = 1. Cependant, il existe des conventions et des règles générales qui justifient ces hyper-paramètres.

2.3 Couches de mise en commun (Pooling Layers)

- **Description de la procédure de mise en commun**

Il est courant d'insérer périodiquement une couche Pooling entre des couches de convolution successives dans une architecture ConvNet. Sa fonction est de réduire progressivement la taille spatiale de la représentation afin de réduire la quantité de paramètres et de calculs dans le réseau, et donc de contrôler également le sur-apprentissage. La couche Pooling fonctionne indépendamment sur chaque tranche de profondeur de l'entrée et la redimensionne spatialement, en utilisant l'opération MAX. La forme la plus commune est une couche de regroupement avec des filtres de taille 2x2 appliquée avec un pas de 2 sous-échantillonnant chaque tranche de profondeur dans l'entrée par 2 le long de la largeur et la hauteur, rejetant ainsi 75% des activations. Chaque opération MAX prendrait dans ce cas un maximum de 4 chiffres (petite région 2x2 dans une tranche de profondeur). La dimension de profondeur reste inchangée.

Plus généralement, la couche de pooling:

Accepte un volume de taille $W_1 \times H_1 \times D_1$

Nécessite deux hyper-paramètres:

- L'étendue spatiale F
- Le pas S

Produit en sortie un volume de taille $W_2 \times H_2 \times D_2$ où:

- $W_2 = (W_1\text{-}F) / S + 1$
- $H_2 = (H_1\text{-}F) / S + 1$
- $D_2 = D_1$

La couche de pooling introduit zéro paramètre car elle calcule une fonction fixe de l'entrée ; il est à noter qu'il n'est pas courant d'utiliser un zero-padding pour les couches Pooling.

Il est important de noter qu'en pratique on rencontre uniquement deux variantes de la couche de mise en commun maximale : une couche de pooling avec F = 3 et S = 2 (aussi appelé regroupement de chevauchement), et plus communément F = 2, S = 2. Les tailles de regroupement avec des champs réceptifs plus important sont trop destructrices.

- **Mise en commun générale**

En plus de la mise en commun recourant à la fonction maximale, les unités de mise en commun peuvent également exploiter d'autres fonctions, telles que la mise en commun basée sur la fonction moyenne ou sur la norme L2. Souvent, la mise en commun moyenne a été historiquement utilisée, mais récemment elle a été est abandonnée en faveur de l'opération de mise en commun maximale, qui s'est avérée plus appropriée dans la pratique.

La couche de regroupement sous-échantillonne le volume spatialement, indépendamment dans chaque tranche de profondeur du volume d'entrée (Fig.3.4). Sur cette Figure 3.4, on peut voir à gauche que, dans cet exemple, le volume d'entrée de taille [224x224x64] est regroupé, avec un filtre de taille 2 et un pas de 2, dans un volume de sortie de taille [112x112x64]. Notez que la profondeur du volume est préservée. A droite, de cette même Figure 3.4, est illustrée l'opération de regroupement exploitant l'opérateur max, ce qui donne lieu à un pool

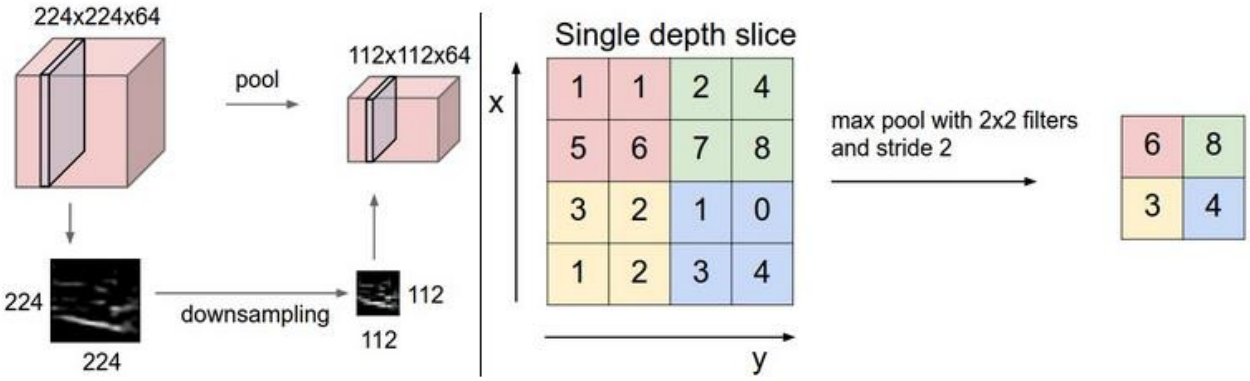


Fig.3.4 Illustration de la mise en commun basée sur la fonction max

maximum, ici représenté avec un pas de 2, c'est-à-dire que chaque max est pris sur 4 nombres (petit carré 2x2).

- **Non nécessité de la mise en commun**

La plupart des concepteurs rejettent le recours à l'opération de mise en commun et pensent qu'il est possible de s'en passer. Par exemple, Striver for Simplicity: The All Convolutional Net propose de rejeter la couche de mise en commun au profit d'une architecture constituée uniquement d'une répétition de couches de convolution. Pour réduire la taille de la représentation, ils suggèrent d'utiliser de temps à autre un plus grand pas dans la couche de convolution. Il est fort probable que les futures architectures comporteront très peu ou pas du tout de couches de regroupement.

2.3 Couche de normalisation (Normalization Layer)

De nombreux types de couches de normalisation ont été proposés pour qu'ils soient utilisés dans les architectures ConvNet, parfois avec l'intention de mettre en œuvre des schémas d'inhibition observés dans le cerveau biologique. Cependant, ces couches sont tombées en désuétude car, dans la pratique, leur contribution s'est avérée minime, s'il en a.

2.4 Couche entièrement connectée (Fully-connected layer)

Les neurones dans une couche entièrement connectée ont des connexions complètes à toutes les activations dans la couche précédente. Le but de cette couche est d'utiliser ces fonctions pour classer l'image d'entrée dans différentes classes (par exemple : Dog, Cat, etc.)

en fonction de l'ensemble des données d'apprentissage. Ici, pour sur-aplatir les résultats, un réseau de neurones artificiel complet (Artificial Neural Network) est appliqué pour obtenir les valeurs de sortie. En effet, la somme des probabilités de sortie de la couche entièrement connectée est toujours 1. Ceci est assuré en utilisant la fonction Softmax comme fonction d'activation dans la couche de sortie de la couche entièrement connectée. En outre, la fonction Softmax traite un vecteur de scores de sortie réels arbitraires (à la Figure 3.5 est donné un exemple de quatre valeurs en sortie) et les écrase ensuite en un vecteur de valeurs entre zéro et un et dont la somme est égale à un.

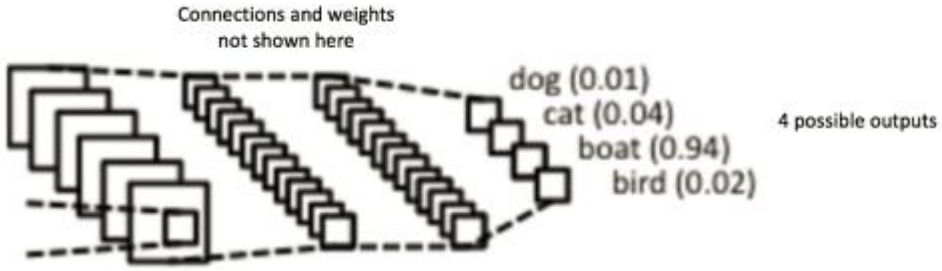


Fig.3.5 Illustration de la génération des classes

2.5 Conversion de couches entièrement connectées en couches de convolution

Il est à noter que la seule différence entre les couches entièrement connectées et les couches de convolution réside dans le fait que d'une part les neurones dans les couches de convolution sont connectés uniquement à une région locale de l'entrée, et d'autre part de nombreux neurones dans un volume CONV partagent les mêmes paramètres. Cependant, les neurones des deux couches calculent encore des produits scalaires, leur forme fonctionnelle est donc identique. Par conséquent, il s'avère qu'il est possible de convertir les couches entièrement connectées en des couches de convolution et inversement.

- Pour toute couche de convolution**,** il existe une couche entièrement connectée qui implémente la même fonction. La matrice de pondération serait une grande matrice qui est la plupart du temps nulle, sauf pour certains blocs (en raison de la connectivité locale) où les poids dans de nombreux blocs sont égaux (en raison du partage des paramètres).

- Inversement, toute couche entièrement connectée peut être convertie en une couche de convolution. Par exemple, une couche entièrement connectée avec K = 4096 et qui pointe un un volume d'entrée de taille 7 × 7 × 512 peut être exprimé de manière équivalente comme une couche de convolution avec F = 7, P = 0, S = 1, K = 4096. En d'autres termes, nous définissons la taille du filtre comme étant exactement la taille du volume d'entrée, et donc la sortie sera simplement 1 × 1 × 4096 car une seule colonne de profondeur "s'ajuste" sur le volume d'entrée, donnant un résultat identique à la couche entièrement connectée initiale.

Parmi ces deux conversions, la possibilité de convertir une couche entièrement connectée en une couche de convolution est particulièrement utile dans la pratique. Considérons une architecture ConvNet qui prend une image 224x224x3, puis utilise une série de couches de convolutions et des couches de mise en commun pour réduire l'image à un volume d'activations de taille 7x7x512 (dans une architecture AlexNet que nous verrons plus tard, cela se fait à l'aide de 5 couches de regroupement qui sous-échantillonnent spatialement l'entrée par un facteur de deux à chaque fois, ce qui donne la taille spatiale finale 224/2/2/2/2/2 = 7). A partir de là, un AlexNet utilise deux couches entièrement connectées de taille 4096 et enfin les dernières couches entièrement connectées avec 1000 neurones qui calculent les scores de classe. Nous pouvons convertir chacune de ces trois couches entièrement connectées en couches de convolution comme indiqué ci-dessus:

- Remplacer la première couche entièrement connectée qui pointe sur le volume [7x7x512] par une couche de convolution qui utilise un filtre de la taille F = 7, conduisant à un volume de sortie [1x1x4096].

- Remplacer la deuxième couche entièrement connectée par une couche de convolution qui utilise un filtre de taille F = 1, donnant un volume de sortie [1x1x4096].
- Remplacer de la même manière la dernière couche entièrement connectée, par une couche de convolution ayant F = 1, donnant une sortie finale [1x1x1000].

Chacune de ces conversions pourrait en pratique impliquer la modification (par exemple, le remodelage) de la matrice de poids W de chaque couche entièrement connectée en des

filtres de couche de convolution. Il s'avère que cette conversion permet au ConvNet de balayer de nombreuses positions spatiales dans des images plus grande très efficacement en un seul passage vers l'avant (forward pass).

Par exemple, si l'image 224 x 224 donne un volume de taille [7x7x512] - soit une réduction de 32, le transfert d'une image de taille 384 x 384 dans l'architecture convertie donnerait le volume équivalent en taille [12x12x512], puisque 384/32 = 12. Traverser avec les 3 prochaines couches de convolutions que nous venons juste de convertir à partir de couches entièrement connectées donnerait maintenant le volume final de taille [6x6x1000], puisque (12 - 7) / 1 + 1 = 6. Notez qu'au lieu d'un seul vecteur de scores de classe de taille [1x1x1000], nous obtenons maintenant un tableau entier 6x6 de scores de classe à travers l'image 384 x 384.

Naturellement, transmettre une seule fois le réseau converti est beaucoup plus efficace que d'itérer le réseau ConvNet original sur l'ensemble de ces 36 emplacements, puisque les 36 évaluations partagent le calcul. Cette astuce est souvent utilisée en pratique pour obtenir de meilleures performances, où par exemple, il est courant de redimensionner une image pour l'agrandir, utiliser un ConvNet converti pour évaluer les scores de classe à plusieurs positions spatiales et ensuite faire la moyenne des scores de classe.

3 Architectures ConvNet

Nous avons vu que généralement les réseaux de convolutions sont seulement constitués de trois types de couche: CONV, POOL (le pool Max est pris par défaut sauf mention contraire) et FC. En outre, la fonction d'activation RELU est aussi considérée comme une couche où une non-linéarité élémentaire est appliquée. Dans cette section, nous discutons de la façon dont ces couches sont généralement empilées ensemble pour former des ConvNets entiers.

3.1 Modèles de couche

La forme la plus courante d'une architecture ConvNet empile quelques couches CONV-RELU et les fait suivre par des couches POOL puis ce motif est répété jusqu'à ce que l'image soit fusionnée spatialement en une petite taille. À un certain point, il est courant de passer à des couches entièrement connectées. La dernière couche entièrement connectée

contient les résultats, tels que les scores de classe. En d'autres termes, l'architecture ConvNet la plus commune suit le modèle suivant :

INPUT -> [[CONV -> RELU] * N ->POOL ?]* M -> [FC -> RELU] * K -> FC

Où le symbole * indique la répétition, et la couche POOL ? signifie une couche POOL facultative. De plus, les nombre entiers N, M et K vérifient les conditions suivantes :

$N \geq 0$ (habituellement $N \leq 3$), $M \geq 0$, $K \geq 0$ (habituellement $K < 3$).

En exemple, voici quelques architectures ConvNet communes qui respectent ce modèle:

- INPUT -> FC, implémente un classificateur linéaire. Ici N = M = K = 0.
- INPUT -> CONV -> RELU -> FC
- INPUT -> [CONV -> RELU -> POOL] * 2 -> FC -> RELU -> FC. Ici, nous voyons qu'il y a une seule couche CONV entre chaque couche POOL.
- INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL] * 3 -> [FC -> RELU] * 2 -> FC Ici, nous voyons deux couches CONV empilées avant chaque couche POOL. C'est généralement une bonne idée pour les réseaux plus grands et plus profonds, car plusieurs couches CONV empilées peuvent développer des caractéristiques plus complexes du volume d'entrée avant l'opération de regroupement destructif.

Supposons que vous empiliez trois couches CONV 3 x 3 les unes sur les autres (avec bien sûr des non-linéarités entre les deux). Dans cet agencement, chaque neurone sur la première couche CONV a une vue de taille 3 x 3 du volume d'entrée. Un neurone sur la deuxième couche CONV a une vue de taille 3x3 de la première couche CONV, et donc par extension une vue de taille 5 x 5 du volume d'entrée d'origine. De même, un neurone sur la troisième couche CONV a une vue de taille 3 x 3 de la deuxième couche CONV, et donc une vue de taille 7 x 7 du volume d'entrée d'origine. Supposons qu'au lieu de ces trois couches de CONV 3 x 3, nous voulions seulement utiliser une seule couche CONV avec des champs réceptifs 7 x 7. Ces neurones auraient une taille du champ réceptif du volume d'entrée qui est identique dans l'étendue spatiale (7x7), mais avec plusieurs inconvénients. Premièrement, les neurones calculeraient une fonction linéaire sur l'entrée, tandis que les trois piles de couches CONV contiennent des non-linéarités qui rendent leurs caractéristiques plus expressives.

Deuxièmement, si nous supposons que tous les volumes ont des canaux C, alors on peut voir que la seule couche CONV 7 x 7 contiendrait des paramètres $C \times (7 \times 7 \times C) = 49C^2$, tandis que les trois couches CONV 3 x 3 ne contiendraient que $3 \times (C \times (3 \times 3 \times C)) = 27C^2$ paramètres. Intuitivement, empiler des couches CONV avec de minuscules filtres plutôt que d'avoir une couche CONV avec de gros filtres nous permet d'exprimer des caractéristiques plus puissantes de l'entrée, et avec moins de paramètres. En tant qu'inconvénient pratique, nous pourrions avoir besoin de plus de mémoire pour conserver tous les résultats des couches CONV intermédiaires si nous prévoyons de faire une rétropropagation.

3.2 Dimensionnement des couches

Jusqu'à présent, nous avons omis les mentions d'hyper-paramètres les plus communs utilisés dans des couches d'un réseau ConvNet. Nous allons d'abord énoncer les règles générales pour dimensionner les architectures et ensuite suivre les règles avec une discussion de la notation:

La couche d'entrée (contient l'image) doit être divisible par 2 plusieurs fois. Les nombres communs comprennent 32 (par exemple CIFAR-10), 64, 96 (par exemple STL-10), ou 224 (par exemple, ConvNets ImageNet communs), 384 et 512.

Les couches de convolution devraient utiliser de petits filtres (par exemple 3x3 ou au plus 5x5), en utilisant un pas S = 1, et, de manière cruciale, le remplissage du volume d'entrée avec des zéros tel que la couche de convolution ne modifie pas les dimensions spatiales de l'entrée. Autrement dit, lorsque F = 3, l'utilisation de P = 1 conservera la taille d'origine de l'entrée. Lorsque F = 5 on choisit P = 2. En général pour un F donné, on impose P tel que P = (F-1) / 2 pour préserver la taille d'entrée. Si vous devez utiliser des tailles de filtre plus grandes (comme 7x7 ou plus), il est courant de voir ceci sur la toute première couche de convolution qui regarde l'image d'entrée.

Les couches de mise en commun sont chargées de sous-échantillonner les dimensions spatiales de l'entrée. Le paramètre le plus courant consiste à utiliser la mise en commun maximale avec des champs réceptifs 2x2 (c'est-à-dire F = 2), et avec un pas de 2 (c'est-à-dire S = 2). Notez que cela supprime exactement 75% des activations dans un volume d'entrée (en raison du sous-échantillonnage de 2 en largeur et en hauteur). Un autre paramètre un peu moins commun est d'utiliser un champ réceptif de taille 3x3 avec un pas de 2, mais cela fait. Il est très rare de voir des tailles de champs réceptifs pour une mise en commun maximale

supérieure à 3, car la mise en commun est alors trop difficile et trop agressive. Cela conduit généralement à des performances moins bonnes.

Pourquoi utiliser un pas égale à 1 en convolution? Les petits pas fonctionnent mieux dans la pratique. De plus, comme déjà mentionné, le pas S = 1 nous permet de laisser tout l'échantillonnage spatial vers le bas aux couches de mise en commun, les couches de convolution ne faisant que transformer le volume d'entrée en profondeur.

Pourquoi utiliser le zero-padding? En plus de l'avantage susmentionné de maintenir les tailles spatiales constantes après la convolution, cela améliore réellement les performances. Si les couches de convolution ne devaient pas mettre à zéro les entrées et n'effectuaient que des convolutions valides, la taille des volumes diminuerait d'une petite quantité après chaque convolution et les informations sur les bordures seraient «balayées» trop rapidement.

Compromis basé sur des contraintes de mémoire. Dans certains cas (surtout au début des architectures ConvNet), la quantité de mémoire peut s'accumuler très rapidement avec les règles empiriques présentées ci-dessus. Par exemple, le filtrage d'une image 224x224x3 avec trois couches de convolution 3x3 avec 64 filtres chacune et un padding 1 créerait trois volumes d'activation de taille [224x224x64]. Cela représente un total d'environ 10 millions d'activations, soit 72 Mo de mémoire (par image, pour les activations et les dégradés). Étant donné que les GPU sont souvent goulotés par la mémoire, il peut être nécessaire de faire des compromis. En pratique, les concepteurs préfèrent faire le compromis uniquement à la première couche de convolution du réseau. Par exemple, un compromis serait d'utiliser une première couche de convolution avec des tailles de filtre de 7x7 et un pas de 2. Un autre exemple, un AlexNet utilise des tailles de filtre de 11x11 et de pas de 4.

3.3 Modèles d'architecture les plus utilisées

Il existe plusieurs architectures dans le domaine des réseaux de convolution qui ont un nom. Les plus communs sont:

- **LeNet.** Les premières applications réussies des réseaux de convolution ont été développées par Yann LeCun en 1990. Parmi ceux-ci, le plus connu est l'architecture LeNet utilisée pour lire les codes postaux, les chiffres, etc.

- **AlexNet.** Le premier travail qui a popularisé le Convolutional Networks dans Computer Vision a été un AlexNet, développé par Alex Krizhevsky, Ilya Sutskever et Geoff Hinton. L'AlexNet a été soumis au défi ILSVRC d'ImageNet en 2012 et a nettement surpassé le deuxième finaliste (avec une erreur de 16% par rapport au second avec 26% d'erreur). Le réseau avait une architecture très similaire à celle du réseau LeNet, mais était plus profond, plus grand et présentait des couches de convolution empilées les unes sur les autres (auparavant, il était courant de n'avoir qu'une seule couche de convolution suivie immédiatement d'une couche de mise en commun).

- **GoogLeNet.** Le gagnant de l'ILSVRC 2014 était un réseau de convolution de Szegedy et al. de Google. Sa principale contribution a été le développement d'un module de lancement qui a considérablement réduit le nombre de paramètres dans le réseau (4M, comparé à AlexNet avec 60M). En outre, ce papier utilise des pools de moyenne au lieu de couches entièrement connectées en haut du ConvNet, ce qui élimine une grande quantité de paramètres qui ne semblent pas avoir beaucoup d'importance. Il existe également plusieurs versions de suivi pour le GoogLeNet, plus récemment Inception-v4.

- **VGGNet.** Le second de l'ILSVRC 2014 était le réseau de Karen Simonyan et Andrew Zisserman, connu sous le nom de VGGNet. Sa principale contribution a été de montrer que la profondeur du réseau est un élément essentiel pour une bonne performance. Leur meilleur réseau final contient 16 couches CONV/FC et, de manière intéressante, dispose d'une architecture extrêmement homogène qui ne fait que 3x3 convolutions et 2x2 pooling du début à la fin. Leur modèle pré-entraîné est disponible pour une utilisation plug and play dans Caffe. Un inconvénient du VGGNet est qu'il est plus coûteux à évaluer et utilise beaucoup plus de mémoire et de paramètres (140M). La plupart de ces paramètres sont dans la première couche entièrement connectée, et il a été constaté que ces couches FC peuvent être retirées sans dégradation de la performance, ce qui réduit considérablement le nombre de paramètres nécessaires.

- **PilotNet.** Dans le cadre du développement des softwares pour la conduite autonome, NVIDIA a créé un système basé sur les réseaux de neurones de convolutions, connu sous le nom de PilotNet, son rôle est de générer les angles de braquage à partir des images de la route

à venir. PilotNet est entrainé en utilisant des images de route associées à des angles de braquage générés par un humain conduisant une voiture de collecte de données.

L'architecture du réseau est représentée à la Figure 3.6. Le réseau se compose de 9 couches, y compris une couche de normalisation, 5 couches de convolution et 3 couches entièrement connectées. L'image d'entrée est divisée en plans Y'UV (Le plan **Y'UV** définit un espace colorimétrique en trois composantes. Le signal Y'UV est créé depuis une source RGB, Y' Les valeurs de R, G et B sont additionnées selon leur poids relatif pour obtenir le signal Y'. Ce dernier représente la luminance Le signal U est obtenu en soustrayant le Y' du signal bleu d'origine ; de façon similaire le V est obtenu en soustrayant Y' du signal rouge, les deux représentent la chrominance) [Citation Wikipédia] et transmise au réseau.

La première couche du réseau effectue la normalisation de l'image. Le normalisateur est programmé en dur et ne peut pas être ajusté dans le processus d'apprentissage. L'exécution de la normalisation dans le réseau permet d'alterner le schéma de normalisation avec le réseau et d'accélérer le traitement par GPU.

Les couches de convolution ont été conçues pour effectuer l'extraction de caractéristiques et ont été choisies empiriquement grâce à une série d'expériences qui ont varié les configurations des couches. Nous utilisons pour les trois premières couches de convolution des filtres de taille 5 x 5 pixels et un pas de 2 x 2 et pour les deux dernières couches de convolution des filtres 3 x 3 sans pas de glissement (S=0). Il est a noté qu'aucun zero-padding n'a été utilisé pour les cinq couches de convolution. Les cinq couches de convolution sont suivies de trois couches entièrement connectées conduisant à une valeur de sortie qui est l'inverse du rayon de braquage. Les couches entièrement connectées sont conçues pour fonctionner comme un contrôleur pour l'angle de braquage du véhicule.

Calcul de la taille du volume de sortie:

La règle, établie dans les parties précédente, où on a :

$$W_2 = (W_1 - F + 2P) / S + 1 \quad \text{et} \quad H_2 = (H_1 - F + 2P) / S + 1$$

conduira à des valeurs relatives aux hyper-paramètres non valides, ainsi pour la première couche de convolution on aura :

$$W_2 = (200 - 5) / 2 + 1 = 98.5 \quad \text{et} \quad H_2 = (66-5) / 2 + 1 = 31.5$$

Ces valeurs ne sont pas appliquées plutôt, la taille du volume de sortie est calculée comme suit :

$$W_2 = (W_1 - F + 1) / S \quad \text{et} \quad H_2 = (H_1 - F + 1) / S$$

Et donc, pour la première couche de convolution on aura :

$$W_2 = (200 - 5 + 1)/2 = 98 \quad H_2 = (66 - 5 + 1) = 31$$

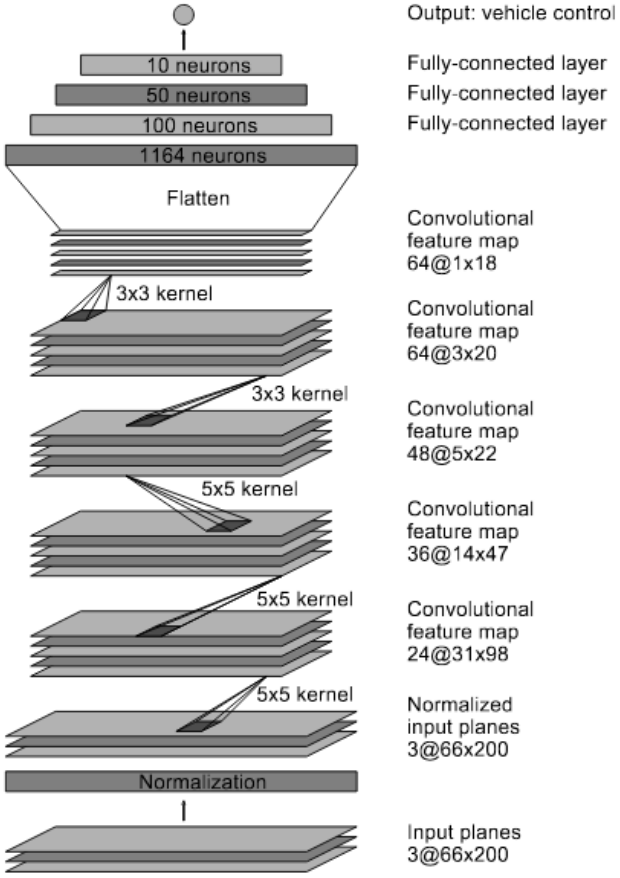


Fig.3.6 Architecture du PilotNet

Le réseau compte environ 27 millions de connexions et 250000 paramètres.

3.4 Considérations de calcul

Le plus gros problème à prendre en charge lors de la construction des architectures ConvNet est relatif à la taille nécessaire de la mémoire. La majorité des GPU modernes ont une limite de 3/4/6 Go de mémoire où les meilleurs GPU sont dotés d'une mémoire d'environ 12 Go. Il y a trois principales grandeurs qui influencent la taille de la mémoire

- Le nombre brut d'activations à chaque couche du ConvNet ainsi que leurs gradients (de taille égale) qui peuvent être déterminés à partir des tailles du volume intermédiaire. Habituellement, la plupart des activations sont sur les premières couches d'un réseau ConvNet (c'est-à-dire les premières couches de convolution). Celles-ci sont conservées parce qu'elles sont nécessaires pour la backpropagation, mais une mise en œuvre intelligente qui exécute un ConvNet seulement au moment du test pourrait en réduire considérablement le nombre, en stockant seulement les activations actuelles sur n'importe quelle couche et en rejetant les activations précédentes.

- Le nombre de paramètres du réseau et leurs gradients pendant la backpropagation. Par conséquent, la mémoire pour stocker le vecteur des paramètres seul doit être généralement multipliée par un facteur d'au moins 3 environ.

- La miscellaneous memory nécessaire pour la mise en œuvre du ConNet telle que les lots de données d'image (the image data batches) et peut-être même leurs versions augmentées, etc.

Une fois une estimation grossière du nombre total de valeurs (les activations, les dégradés et les misc) est effectuée, ce nombre doit être converti en taille en Go. Prendre le nombre de valeurs, multiplier par 4 pour obtenir le nombre brut d'octets (puisque chaque point flottant est de 4 octets, ou peut-être par 8 pour une double précision), puis diviser par 1024 plusieurs fois pour obtenir la quantité de mémoire en Ko, MB, et enfin GB. Si le réseau ne convient pas, une heuristique commune pour "l'adapter" est de diminuer la taille du lot, car la majeure partie de la mémoire est consommée par les activations.

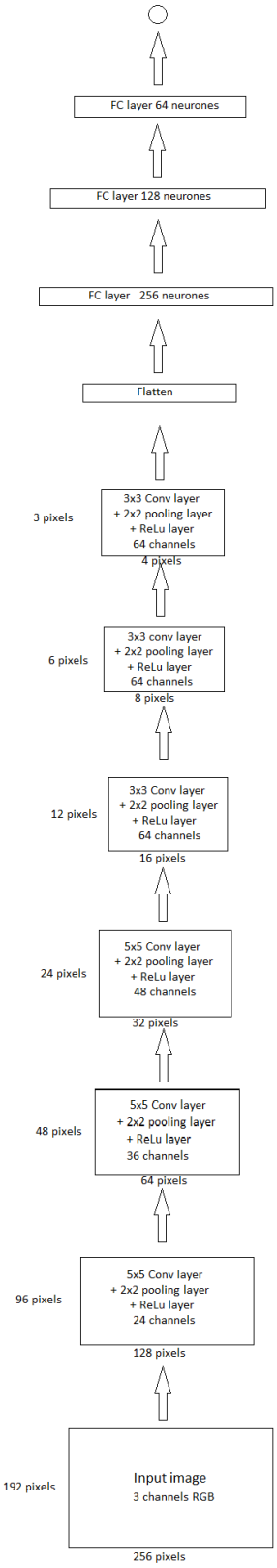


Fig 3.7 Architecture du réseau de neurones proposée pour la conduite d'un véhicule autonome

4 Création de notre architecture de réseau de neurones de convolution pour la conduite d'un véhicule autonome

Notre réseau est constitué de 6 couches (de convolution + mise en commun) chacune suivie d'une couche Relu et 4 couches complètement connectées (voir Fig.3.7).

Pour générer le modèle, on va utiliser les fonctions de la bibliothèque Keras.

Tout d'abord, commençons par charger nos dépendances :

```
1 from keras.models import Sequential
2 from keras.layers import Dense, Dropout, Flatten, Lambda, ELU, Activation
3 from keras.layers.convolutional import Convolution2D
```

Le script ci-dessous représente notre modèle qu'on va détailler ligne par ligne :

```
1
2     model = Sequential()
3
4     model.add(Convolution2D(24, 5, 5, subsample=(2, 2), border_mode="same", input_shape=(row, col, ch)))
5     model.add(Activation('relu'))
6     model.add(Convolution2D(36, 5, 5, subsample=(2, 2), border_mode="same"))
7     model.add(Activation('relu'))
8     model.add(Convolution2D(48, 5, 5, subsample=(2, 2), border_mode="same"))
9     model.add(Activation('relu'))
10    model.add(Convolution2D(64, 3, 3, subsample=(2, 2), border_mode="same"))
11    model.add(Activation('relu'))
12    model.add(Convolution2D(64, 3, 3, subsample=(2, 2), border_mode="same"))
13    model.add(Activation('relu'))
14    model.add(Convolution2D(64, 3, 3, subsample=(2, 2), border_mode="same"))
15    model.add(Flatten())
16    model.add(Dropout(0.5))
17    model.add(Activation('relu'))
18    model.add(Dense(256))
19    model.add(Activation('relu'))
20    model.add(Dense(128))
21    model.add(Activation('relu'))
22    model.add(Dense(64))
23    model.add(Activation('relu'))
24    model.add(Dense(1))
```

Les modèles issus de la bibliothèque Keras se présentent sous deux formes :Sequential et via l'API fonctionnelle. Pour la plupart des réseaux d'apprentissage profonds, on utilise généralement le modèle Sequential, car il permet d'empiler facilement des couches séquentielles (et même des couches récurrentes) du réseau dans l'ordre de l'entrée vers la sortie. L'API fonctionnelle permet de construire des architectures plus compliquées.

```
2   model = Sequential()
```

La première ligne déclare le type de modèle en tant que Sequential ()

```
4     model.add(Convolution2D(24, 5, 5, subsample=(2, 2), border_mode="same", input_shape=(row, col, ch)))
```

Ensuite, nous ajoutons une couche de convolution 2D pour traiter les images d'entrée. Le premier argument est la déclaration d'une couche Conv2D () ayant en sortie 24 canaux (nombre des filtres) de taille (kernel_size) 5 × 5 pixels ; le troisième paramètre (2, 2) fixe les

pas dans les directions x et y. Il est à noter que le paramètre subsample effectue une opération de sous-échantillonnage (2, 2) dans les directions x et y, il représente la couche de mise en commun de plus, pour cette version de Keras 1.1, la couche de mise en commun est introduite comme un argument pour chaque couche de convolution cela signifie qu'après chaque opération de convolution, nous avons une opération de sous-échantillonnage. Il est également possible d'exploiter le zero-padding (border_mode) et dans Keras, il existe deux options pour le padding: same ou valide où, Same signifie que le remplissage se fait sur le bord et valide signifie une absence de remplissage. Enfin, nous devons fournir au modèle la taille de l'entrée de la couche (qui est déclarée dans une autre partie du code). La déclaration de la forme d'entrée est seulement requise pour la première couche, Keras est assez suffisamment puissant pour calculer la taille des tenseurs qui traversent le modèle à partir de la deuxième couche.

```
5     model.add(Activation('relu'))
```

Cette ligne 5 du code déclare une couche d'activation (RELU) où est utilisée la fonction d'activation du type seuillage à zéro : *max(0,x)*.

Puis, on poursuit la construction du model de réseau de neurones en alternant dans la pile logicielle des couches CONV et des couches RELU.

```
16    model.add(Flatten())
```

Maintenant que nous avons construit les couches de convolutions, il faudrait aplatir leurs sorties pour qu'elles soient bien assorties avec les couches entièrement connectées. Cette opération est obtenue par la ligne de code 16.

```
17    model.add(Dropout(.5))
18    model.add(Activation('relu'))
```

Par la suite, nous ajoutons une couche Dropout (ligne de code 17) suivie d'une couche RELU (ligne de code 18).

Au fait, la fonction dropout est utilisée pour éviter le sur-apprentissage. L'idée est simple en soi en effet, pendant l'entraînement, à chaque itération, un neurone est temporairement "abandonné" ou désactivé avec une probabilité p. Cela signifie que toutes les entrées et sorties de ce neurone seront désactivées à l'itération en cours. Les neurones abandonnés sont ré-

échantillonnés avec la probabilité p à chaque étape d'entraînement, de sorte qu'un neurone abandonné à une étape peut être actif à la suivante. L'hyper-paramètre p est appelé taux de décrochage et il est généralement fixé à une valeur de 0,5, correspondant à 50% de neurones abandonnés.

```
19    model.add(Dense(256))
20    model.add(Activation('relu'))
21    model.add(Dense(128))
22    model.add(Activation('relu'))
23    model.add(Dense(64))
24    model.add(Activation('relu'))
25    model.add(Dense(1))
```

Dans Keras, les couches FC sont déclarées en utilisant Dense (). Nous spécifions d'abord la taille - conformément à notre architecture, nous avons spécifié 256 neurones pour la première couche FC, 128 neurones dans la 2ième couche FC, 64 neurones dans la 3ième couche FC et un neurone dans la 4ième couche FC (la dernière couche). Chaque couche FC est suivie d'une couche RELU.

L'exécution du programme nous permet de visualiser le nombre de paramètres et la taille des volumes de sorties.

Layer (type)	Output Shape	Param #	Connected to
convolution2d_1 (Convolution2D)	(None, 96, 128, 24)	2424	convolution2d_input_1[0][0]
activation_1 (Activation)	(None, 96, 128, 24)	0	convolution2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 48, 64, 36)	21636	activation_1[0][0]
activation_2 (Activation)	(None, 48, 64, 36)	0	convolution2d_2[0][0]
convolution2d_3 (Convolution2D)	(None, 24, 32, 48)	43248	activation_2[0][0]
activation_3 (Activation)	(None, 24, 32, 48)	0	convolution2d_3[0][0]
convolution2d_4 (Convolution2D)	(None, 12, 16, 64)	27712	activation_3[0][0]
activation_4 (Activation)	(None, 12, 16, 64)	0	convolution2d_4[0][0]
convolution2d_5 (Convolution2D)	(None, 6, 8, 64)	36928	activation_4[0][0]
activation_5 (Activation)	(None, 6, 8, 64)	0	convolution2d_5[0][0]
convolution2d_6 (Convolution2D)	(None, 3, 4, 64)	36928	activation_5[0][0]
flatten_1 (Flatten)	(None, 768)	0	convolution2d_6[0][0]
activation_6 (Activation)	(None, 768)	0	flatten_1[0][0]
dense_1 (Dense)	(None, 256)	196864	activation_6[0][0]
activation_7 (Activation)	(None, 256)	0	dense_1[0][0]
dense_2 (Dense)	(None, 128)	32896	activation_7[0][0]
activation_8 (Activation)	(None, 128)	0	dense_2[0][0]
dense_3 (Dense)	(None, 64)	8256	activation_8[0][0]
activation_9 (Activation)	(None, 64)	0	dense_3[0][0]
dense_4 (Dense)	(None, 1)	65	activation_9[0][0]

```
Total params: 406,957
Trainable params: 406,957
Non-trainable params: 0
```

Le modèle de notre réseau de neurones nécessite l'estimation de 406957 paramètres (poids). Cette opération s'effectue lors de la phase d'apprentissage.

5 Conclusion

Dans ce chapitre, nous avons exposés les éléments essentiels relatifs à un réseau de neurones de convolution. Nous avons entamé le chapitre par un bref rappel historique de l'évolution des réseaux de neurones jusqu'à l'émergence récente des réseaux de neurones de convolution. Puis, nous avons présenté la structure d'un ConvNet où nous avons montré qu'elle est composée principalement d'un nombre réduit de couches spécialisées : INPUT, CONV, RELU, POOL et FC où nous avons exposé l'architecture standard d'un ConvNet à partir de ces couches spécialisées. Cependant, la couche la plus fondamentale est la couche CONV. Ainsi, nous avons détaillé sa fonction et ses caractéristiques dépendant de quatre hyper-paramètres: le champ réceptif F, le pas S, le nombre de zero-padding P et le nombre de filtres K. Nous avons également discuté de la question relative au dimensionnement de ces différentes couches. En fin de chapitre nous avons présenté quelques réseaux ConvNets bien célèbres. De plus, nous avons exposés le modèle logiciel, via la bibliothèque Keras, du réseau de neurones de convolution que nous comptons exploiter pour la conduite d'un véhicule autonome.

Chapitre IV
Commande d’un véhicule autonome basée sur le « End to End Deep learning »

Le premier véhicule autonome est apparu en 1989 grâce aux efforts des chercheurs à l'université CMU aux états unis [1]. En effet, le projet ALVINN a démontré qu'un véhicule entrainé par la méthode « end to end learning » peut rouler, d'une façon autonome, sur les routes publiques. En 2015, une équipe de chercheurs de la firme NVIDIA ont pu développer la méthode en ajoutant la puissance de calcul des GPU [2]. Ainsi, il a été démontré, qu'avec une seule caméra frontale, l'ordinateur peut générer la commande du véhicule. Ce résultat est très intéressant pour les raisons suivantes :

- élimination de la complexité due à l'usage habituel de plusieurs capteurs ;
- centralisation de l'organe d'apprentissage évitant par là le recours à la solution habituelle d'un ordinateur par tache (perception, cartographie, détection des objets…etc.) ;
- apprentissage de la conduite, par le véhicule, en recopiant le comportement humain.

Dans ce chapitre, la méthode « end to end learning » (ou apprentissage E2E) est présentée de plus, elle est comparée aux méthodes classiques. Ensuite, on va présenter l'utilisation de l'apprentissage E2E dans le domaine des véhicules autonomes qui est fondée sur le travail de l'équipe de NVIDIA et on expliquera comment la commande peut être générée par cette approche en introduisant la méthode « Visual Backpropagation ». Enfin, on discutera des avantages et des limites de cette méthode.

1 Méthode « end to end » pour l'apprentissage

1.1 Définition

L'apprentissage « end to end » est une méthode d'entrainement des réseaux de neurones complexes en utilisant la descente du gradient déjà présentée au chapitre 2. En effet, dans l'approche « end to end », l'apprentissage ne se fait pas pour chaque module séparément mais il se fait pour tout le système qu'on peut qualifier d'une machine d'apprentissage centrale. Par exemple, pour la conduite d'un véhicule autonome la solution classique exploite plusieurs modules (étapes) nécessitant un traitement séparé : la perception, la cartographie, la détection des objets et enfin, la génération de la commande des actionneurs (Fig.4.1). Par contre, dans l'approche « end to end », ce processus est optimisé afin que la commande soit directement déterminée des images capturées par une caméra frontale sans passer explicitement par ces étapes élémentaires (Fig.4.1).

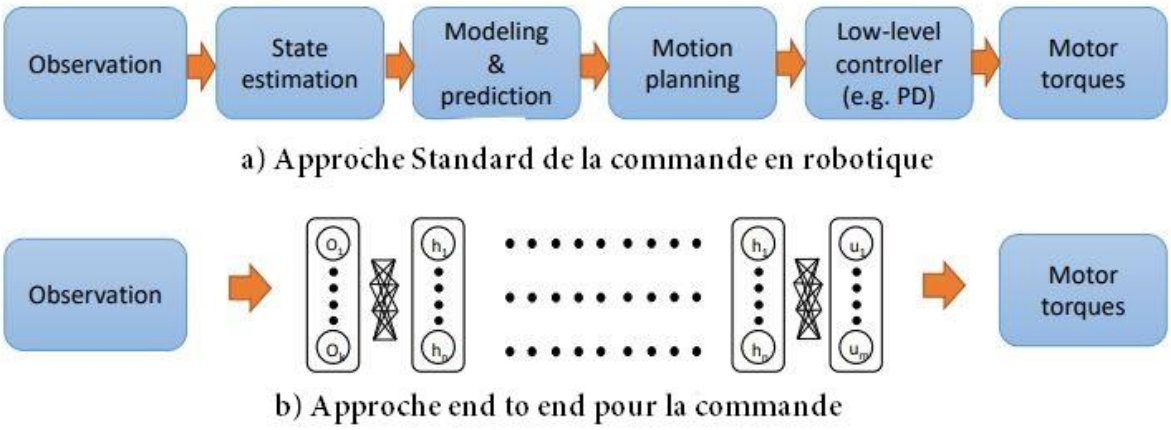


Fig.4.1 Illustration de l'approche standard utilisée dans la robotique autonome et de l'approche end to end basée sur l'apprentissage profond

L'utilisation de cette approche avantageuse à plus d'un titre. En effet, elle permet de réduire la complexité du schéma de commande et le nombre des capteurs utilisé (car on n'utilisera qu'une seule caméra frontale) ce qui réduit le coût.

1.2 Le développement de l'apprentissage E2E

La technique de l'apprentissage E2E exploite judicieusement le fait qu'il existe une différence entre les modules d'un système d'apprentissage. Cette approche peut être appliquée dans le cas où si on décompose le système complexe en modules simples indépendants, on pourrait ajuster séparément les paramètres de chaque sous-système.

L'apprentissage E2E est de plus en plus utilisé pour entrainer les réseaux de neurones profonds et complexes. L'utilisation de cette approche est plus importante dans le contexte de l'apprentissage profond car elle fait appel à l'apprentissage supervisé (l'utilisation des étiquettes dans un jeu de données) et aussi à l'apprentissage par renforcement (l'utilisation de la notion de récompense). L'application de l'apprentissage E2E a montré son efficacité dans plusieurs domaines notamment dans la conduite des véhicules autonomes [2] et la machine de Turing neuronale [3]. Cette efficacité est le résultat des développements dans plusieurs domaines :

- Le développement dans le domaine du calcul parallèle dû à l'essor des GPU qui a permis l'implémentation de réseaux de neurones de convolution qui à leur tour ont conduit au progrès dans le domaine de la vision par ordinateur.

- L'utilisation des GPU, a permis de traiter des problèmes d'apprentissage plus importants et plus complexes. De plus, la disponibilité d'une puissance de calcul suffisante pour traiter des jeux de données plus larges, a permis d'exploiter les techniques de l'apprentissage profond pour résoudre ce type de problèmes.
- Le progrès remarquable dans la recherche concernant les méthodes d'optimisation notamment la descente du gradient stochastique et le développement de nouvelles méthodes d'optimisation linéairement convergentes et faciles à implémenter sur ordinateur comme l'optimisateur d'ADAM [4]. Ceci a conduit à une amélioration des performances des réseaux profonds.
- L'introduction de bibliothèques de programmation pour faciliter développement de programmes dédiés aux problèmes à traiter: les réseaux de neurones de convolution pour MATLAB [5] et KERAS. Ces bibliothèques sont basées sur des packages très puissants comme TENSORFLOW et THEANO.
- La compétition ImageNet, dédiée à la classification des images, a permis d'introduire de nouvelles architectures de réseaux de neurones plus profondes et plus innovantes [6].

1.3 Apprentissage E2E et Apprentissage par module

Les systèmes d'apprentissage profonds actuels procèdent par un apprentissage par module, où chaque module représente une tache laquelle est caractérisée par des couches ou un groupement de couches des réseaux de neurones [7]. Le but étant de composer un système, à partir de plusieurs modules, afin d'effectuer une opération désirée. Cette façon de procédé est analogue à la structure du cerveau humain où on relève l'existence de plusieurs cortex : visuel, auditif, olfactif, commande des muscles…etc. Par exemple, pour une tache de conduite autonome d'un véhicule qui fait l'objet de ce mémoire, l'application de l'apprentissage par module nécessitera les modules suivants : perception, détection des objets, cartographie et commande des moteurs. Cette décomposition du problème complexe, basée sur le principe de « diviser pour régner » fait partie des méthodes d'ingénierie. Dans l'approche E2E, on construit un seul réseau de neurones pour tout le système, ce réseau de neurones profond couvre toutes les taches de conduite sans décomposer le système explicitement.

L'entrainement d'un système d'apprentissage est basé sur l'hypothèse que le pré-conditionnement est valable et assez puissant pour assurer la convergence de la descente du gradient stochastique d'un état initial aléatoire à une solution optimale non triviale. Ceci est

risqué vu la convexité du problème d'optimisation donc, cette approche est recommandée pour les situations où la tâche à effectuer est « presque triviale » dans l'environnement où elle opère.

Le deuxième problème rencontré dans l'utilisation de l'approche E2E par rapport à l'apprentissage par module est l'efficacité des données car, en absence d'interaction entre les modules, il serait nécessaire de disposer de jeux de données plus importants dont la taille augmente exponentiellement avec le nombre de modules [7].

Dans la procédure d'apprentissage par module, une interaction entre les modules peut avoir lieu, rien n'empêche qu'un module puisse effectuer une partie d'une tache d'un autre module. En plus, le caractère complexe des réseaux rend la vérification des taches plus difficile. La solution proposée au problème d'interaction entre les processus d'apprentissage des modules vise à concrétiser, en premier lieu, les taches simples afin de générer des composants pouvant être ensuite utilisés pour réaliser des taches plus complexes. Cette dernière solution peut être ignorée dans le contexte de l'apprentissage E2E car dans cette approche, l'organisation de la procédure d'apprentissage, tout au long de la structure, est tout à fait différente.

2 Apprentissage E2E pour la conduite d'un véhicule autonome

En 2016, une équipe de chercheur de NVIDIA ont développé la conduite d'un véhicule autonome en utilisant la technologie des réseaux neuronaux de convolution. Ce projet consistait à collecter les données et à appliquer l'approche E2E pour permettre au réseau neuronal de convolution, dédié à la conduite du véhicule, d'apprendre les données collectées. Ensuite, ils ont utilisé une simulation pour tester le véhicule avant de le tester sur routes réelles. Ce système basé sur les réseaux neuronaux de convolution fut baptisé « PilotNet » [2]. Motivé par le succès d'ALVINN qui a démontré l'efficacité de l'apprentissage E2E dans la conduite autonome [8], la firme NVIDIA lança un premier projet DAVE (DARPA Autonomous Vehicle) [9] suivi d'un projet complémentaire DAVE-2 pour s'introniser à son tour dans la course au développement de la conduite des véhicules autonomes. En fait, l'équipe NVIDIA a profité du développement des moyens de calcul pour implémenter un réseau neuronal complexe.

2.1. L'architecture de DAVE-2

Le système de collection des données est composé de trois caméras frontales et une unité de traitement qui associe simultanément aux trois images la commande de direction que doit appliquer le conducteur.

Afin de construire un système indépendant de la géométrie du véhicule, la commande de direction est représentée par $\frac{1}{r}$ au lieu de r (r étant l'angle de braquage) pour éviter la singularité représentée par $r = \infty$ lorsque le véhicule se ligne déplace en droite.

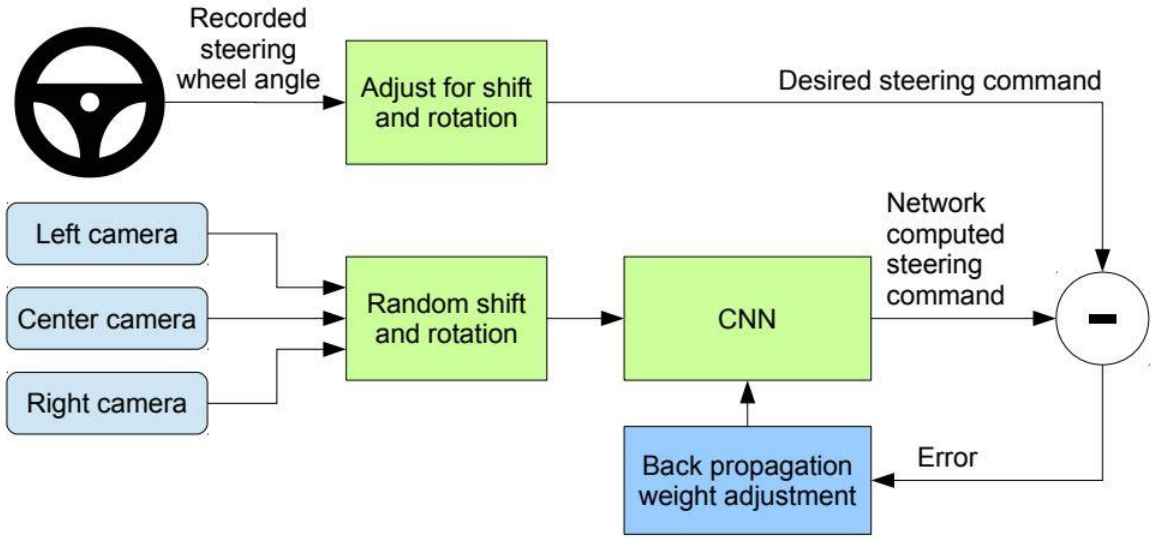


Fig.4.2 illustration de système d'acquisition de données de DAVE-2, et le processus d'entrainement de réseau de neurones de convolution [2]

La valeur de $\frac{1}{r}$ varie entre des valeurs négatives pour les virages à gauche et des valeurs positives pour les virages à droite.

Pour pouvoir disposer de plus de données pour l'entrainement, on procède en général à une augmentation des données acquises. Cette opération consiste à effectuer un décalage aux images capturées par les caméras droite et gauche. D'autres transformations peuvent être appliquées comme la rotation, la réduction ou l'augmentation de la luminosité.

L'entrainement du modèle basé sur les données provenant de l'exemple d'un conducteur n'est pas totalement suffisante, c'est pourquoi le système doit apprendre de ses erreurs, d'où la nécessité d'un apprentissage renforcé. Enfin, pour le test, les commandes de direction sont générées dans le cas où l'image provient d'une seule caméra frontale.

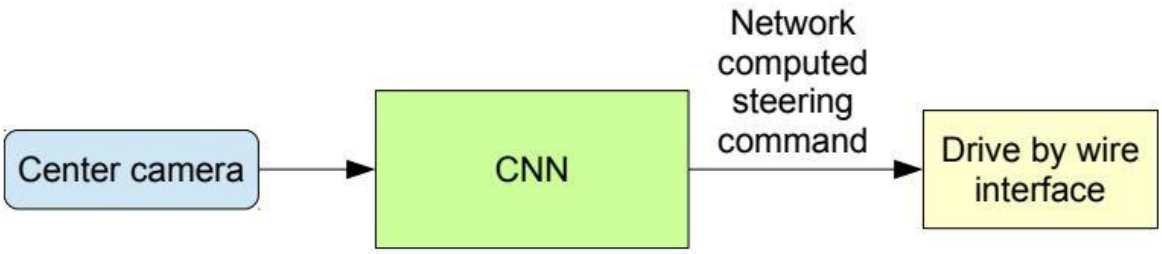


Fig.4.3 Réseau entrainé génère les commande en utilisant qu'une seule caméra frontale

2.2 Architecture du PilotNet

Le PilotNet est un réseau de neurones utilisé dans la phase d'apprentissage. Il est illustré à la Figure 3.4, ce réseau comprend 9 couches : une couche de normalisation, 5 couches de convolution et 3 couches totalement liées.

La première couche de ce réseau effectue la tache de normalisation qui consiste à alterner le schéma de normalisation avec le réseau et à accélérer le traitement via le traitement par le GPU.

Les couches de convolution consistent à extraire des formes des images, après des séries d'expériences, la configuration adoptée est telle que :

- Pour les 3 premières couches : stride 2x2 ; kernel 5x5
- Pour les 2 dernières couches : kernel 3x3

Les 3 dernières couches du réseau sont complètement liées et qui mènent à une seule valeur de sortie, donc ces 3 dernières couches jouent le rôle d'un contrôleur pour la direction du véhicule. Il est à noter qu'à cause de l'application pour l'apprentissage de l'approche E2E on ne peut pas préciser, en un temps donné, quelle partie du réseau fonctionne [2].

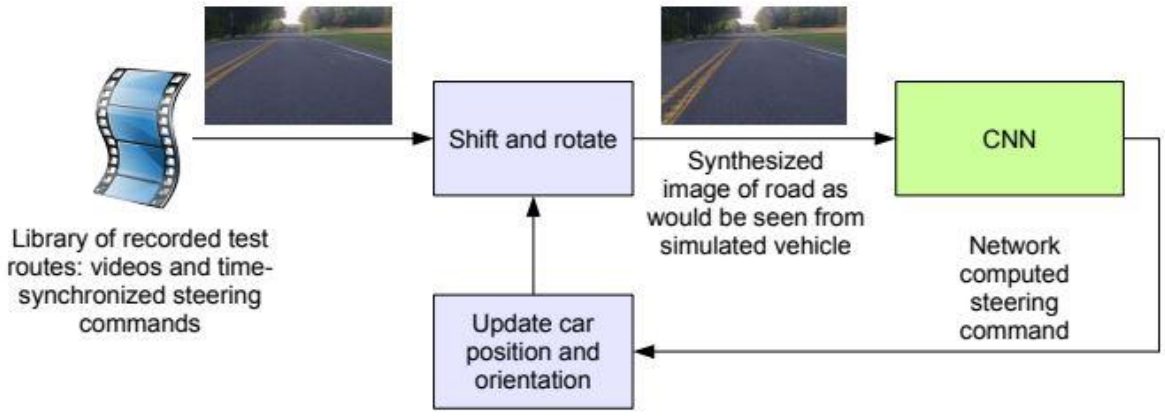


Fig.4. 4 Schéma bloc du simulateur de conducteur

2.3 Simulation et Résultats

Avant de tester le véhicule sur une route réelle, il a été testé sur un simulateur. La simulation a utilisé les images déjà capturées comme entrée principale et les commandes générées par le modèle entrainé pour produire des images qui illustrent ce que le véhicule aurait su en se basant sur ses propres décisions. La simulation permettait l'intervention du conducteur comme option dans le cas où le véhicule diverge de plus d'un mètre de la trajectoire désirée.

Un schéma de simulation est illustré à la Figure 4.5. Cette simulation a donné un aperçu réaliste sur le comportement du véhicule sur des routes réelles.

Après la phase de simulation, le véhicule a été testé sur une route réelle. Un conducteur humain était présent en cas de divergence du véhicule. Le réseau PilotNet n'a jamais été entrainé explicitement pour détecter le marquage du sol ni les objets mais durant la phase d'essai, le système a pu développer sa capacité de chercher les objets saillants dans une image, donc le modèle a pu, en se basant sur les données précédentes, à évoluer dans des nouveaux environnements présentant des caractéristiques qui n'ont pas été prédéfinies explicitement.

Les Figures 4.6&4.7 illustrent comment le PilotNet analyse une situation caractérisée soit par une richesse en informations utiles soit par une absence d'informations utiles.

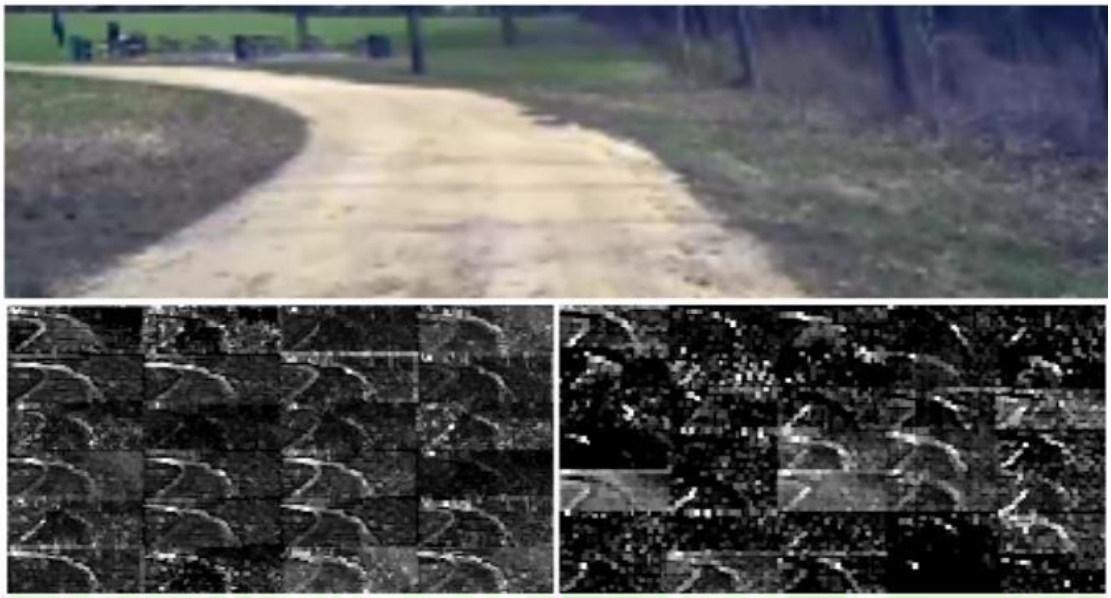


Fig.4.5 En haut : l'image en entrée. En bas : le PilotNet apprend à détecter des objets saillants sans qu'il ait appris explicitement à le faire.

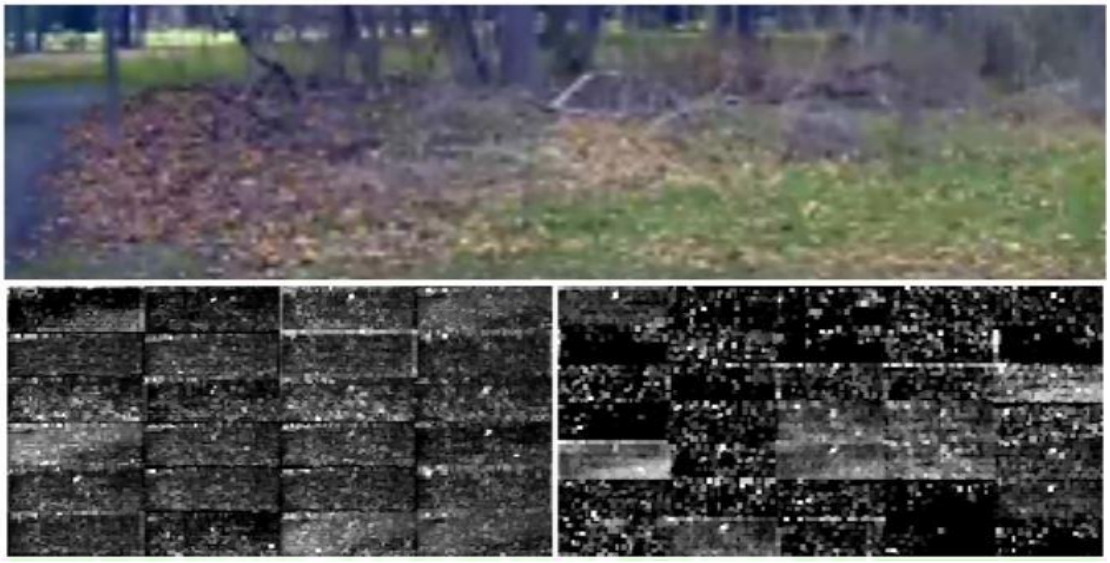


Fig.4.6 En haut : une image qui ne contient pas d'informations utiles. En bas : le PilotNet génère des bruits comme une indication de l'absence d'objets saillants.

NVIDIA a pu construire un véhicule autonome basé sur les réseaux de neurones de convolution et l'apprentissage E2E. Ce véhicule a pu évoluer d'une façon totalement autonome dans 98% des trajets donc l'intervention du conducteur n'était que de 2% de la durée des itinéraires [2].

3. La commande du véhicule par l'apprentissage E2E

Dans la section précédente, on a présenté le PilotNet de NVIDIA un système de conduite autonome qui peut générer les commandes à partir des images d'une seule caméra frontale comme entrée et en observant le comportement d'un conducteur humain pour élargir son domaine de connaissances, ce qui est connu sous le nom de « Behavioural Cloning » ou bien la recopie du comportement. En fait, la particularité de ce système est sa capacité à détecter des objets saillants dans une scène malgré qu'il n'ait pas été entrainé pour le faire explicitement, une caractéristique qui élimine le besoin d'anticiper les scénarios rencontrés dans la réalité. Dans cette partie, on analysera le réseau PilotNet pour découvrir comment peut-il détecter les objets saillants dans une image. La méthode d'analyse pour visualiser le comportement du réseau pour détecter les objets saillants est développée dans [10].

3.1 Les cartes d'extraction des formes (Feature maps) dans les réseaux neuronaux

Avant de présenter la méthode VisualBackProp, il est nécessaire de définir en quoi consiste la technique « feature maps » dans les réseaux neuronaux de convolution. La « Feature map », ou bien une carte d'extraction des formes, est une fonction, qui associe à chaque vecteur de données, un élément dans l'espace des formes. Le but étant de rendre l'apprentissage plus efficace en injectant des images qui contiennent le plus d'informations utiles après les avoir passées par des « feature maps ». Cette fonction aide à accélérer le traitement des images en réduisant les calculs comme indiqué à la Figure 4.8 où une comparaison est donnée entre un réseau de convolution sans feature maps et un autre réseau de convolution incorporant cette fonction.

On constate que, pour un réseau sans feature maps, on devait utiliser plusieurs réseaux de convolution appliqués sur plusieurs régions de l'image, ce qui augmente le temps de calcule. Par contre, en utilisant les feature maps, on n'aurait besoin que d'un seul réseau de convolution qui traite l'image entière ce qui réduit considérablement le temps de calcule.

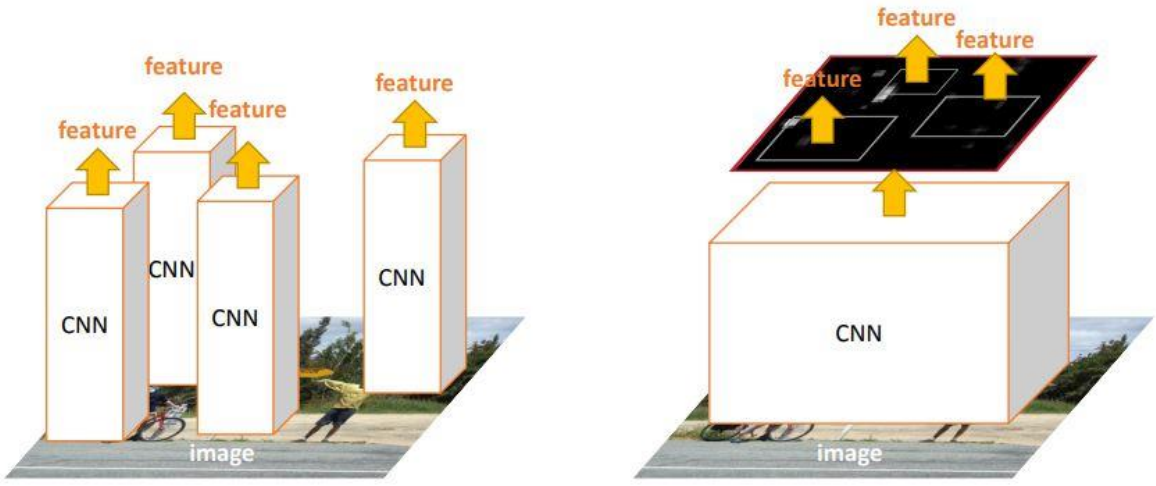


Fig.4.4 Utilisation des feature maps avec les réseaux de convolution

3.2. Visualisation des réseaux de neurones de convolution par la méthode de VisualBackProp

La méthode de la VisualBackProp ou bien la retro-propagation visuelle est utilisée pour visualiser le comportement des réseaux de neurones de convolution durant l'apprentissage. Cette méthode analyse le réseau et permet ainsi de déterminer les régions de l'image qui contribuent le plus à la prédiction de la commande. Cette méthode est basée sur l'analyse par les fonctions « feature maps » lesquelles sont utilisées pour l'extraction des formes incluses dans une image. La méthode est exécutée en temps réel donc elle exige moins de temps de calcul que la «forward propagation».

On sait d'une part qu'en avançant dans le réseau, les « feature maps » contiennent de moins en moins d'informations utiles pour la génération de la commande. D'autre part, la « feature map » de la dernière couche de convolution contient avec certitude les informations les plus utiles pour la génération de la commande. De plus, la résolution (taille) des « feature maps » est réduite dans les couches profondes. Donc, l'idée de cette approche consiste à combiner les « feature maps » profondes qui contiennent les informations utiles avec ceux à résolution élevée.

Pour appliquer cette méthode, on commence par rétro-propager (Back-propagate) les informations utiles présentes dans les « feature maps » de la dernière couche de convolution tout en augmentant simultanément la résolution. Contrairement à la méthode de

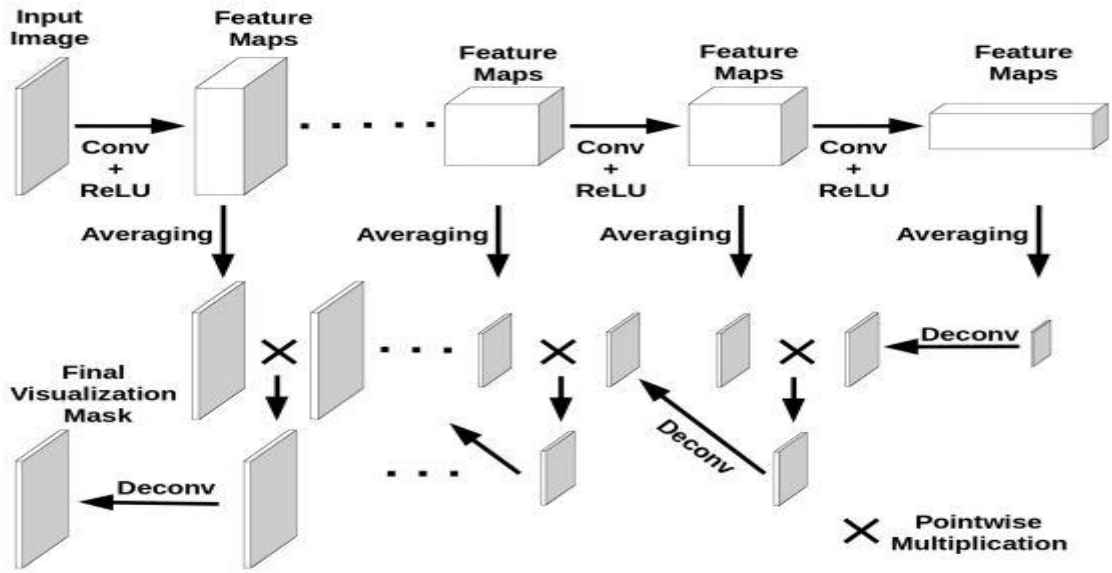


Fig.4.5 Schéma bloc de la méthode de visualisation

Backpropagation présentée dans le chapitre 2 qui est basée sur la descente du gradient, la méthode VisualBackProp est basée sur les valeurs des pixels d'où son appellation « VisualBackProp ».

Pour avoir, en premier lieu, une seule « feature map par couche», les « feature maps » de chaque couche sont moyennés (averaged) après chaque activation (ReLU). Ensuite, la « feature map » de la dernière couche de convolution est modifiée pour avoir la même taille que la « feature map » de la couche précédente par une opération de déconvolution. Cette dernière « feature map » modifiée est multiplié par la « feature map » moyennée et modifiée de la couche précédente, l'image qui en résultante est mise à l'échelle ensuite elle a son tour multipliée par la « feature map » modifié de la couche précédente.

Ce processus est répété tout au long du réseau pour obtenir à la fin un masque de la même taille que l'image d'entrée. Ce masque ne contient alors que les objets saillants dont le véhicule a besoin pour générer la commande [10].

La Figure 4.9 ci-dessous illustre les étapes de génération des masques, la colonne de gauche donne les « feature maps » moyennées de toutes les couches de convolution de l'entrée (en haut) jusqu'à la sortie (en bas) et la colonne de droite donne les masques intermédiaires correspondants [10].

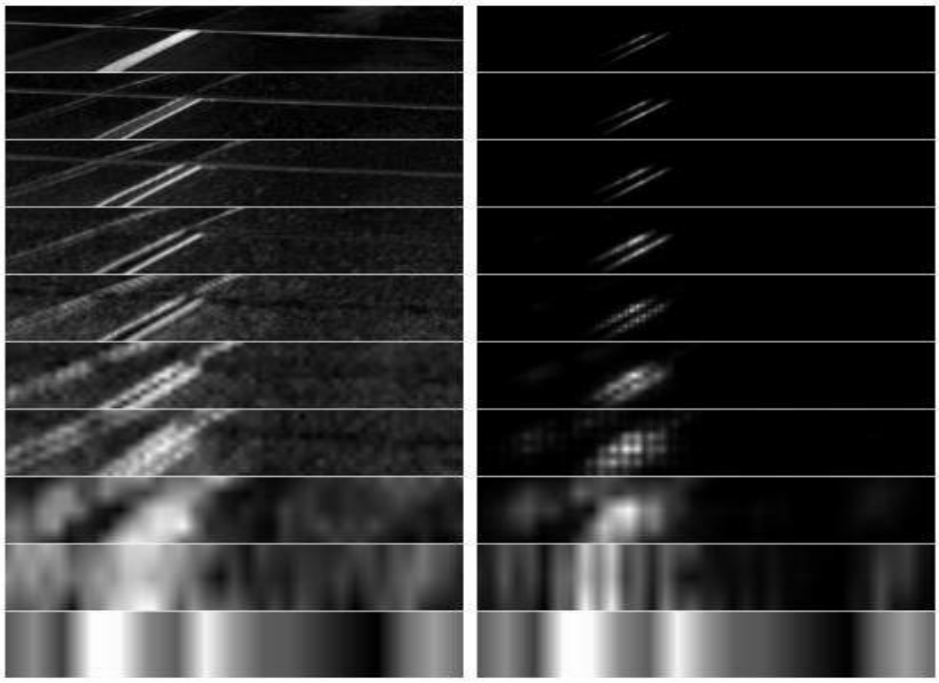


Fig.4.6 FM moyennés (colonne de gauche) et les masques correspondants (colonne de droite)

3.3. Utilisation de la méthode VisualBackProp dans le projet DAVE-2

La Figure 4.10 illustre l'algorithme utilisé par l'équipe de NVIDIA pour détecter les objets saillants durant la conduite du véhicule.

L'algorithme est basé sur l'approche discutée au paragraphe 3.2, l'idée étant de séparer les objets saillants en cherchant les régions de l'image où l'activation des « features » est importante. L'activation des FM profondes devient un masque pour les activations des FM de niveau bas en utilisant cet algorithme [11] :

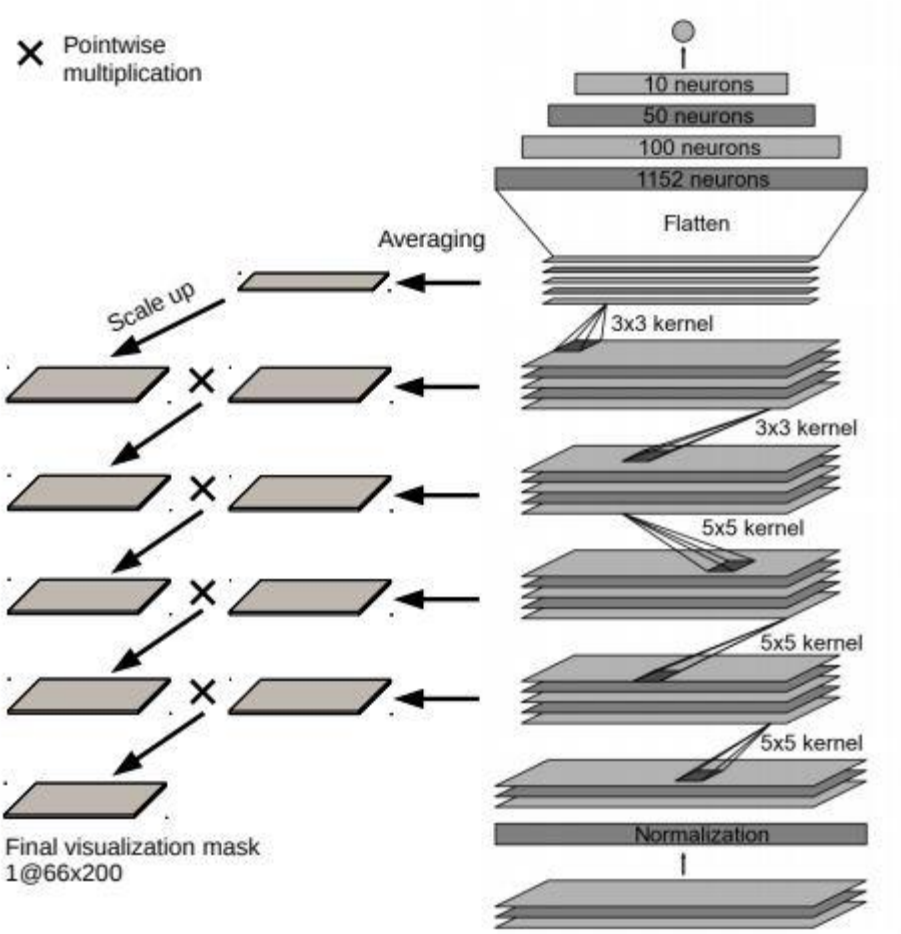


Fig.4.7 Schéma bloc de l'algorithme de détection des objets saillants

1. Dans chaque couche, les activations des FM sont moyennées
2. La plus profonde FM moyennée est mise à l'échelle (même taille) de la FM de la couche précédente.
3. Une fois la plus profonde FM est moyennée et mise à l'échelle, elle est ensuite multipliée par la FM moyennée de la couche précédente pour obtenir un masque intermédiaire.
4. La FM intermédiaire moyennée est mis à l'échelle (la même taille) que la FM de la couche précédente.
5. Une fois la FM intermédiaire est moyennée puis mise à l'échelle, elle est ensuite multiplié par la FM moyennée de la couche précédente, ce qui conduit à l'obtention d'un masque intermédiaire.
6. Étapes 4 et 5 sont répétées jusqu'à l'entrée du réseau, et le dernier masque obtenu est normalisé entre 0 et 1, et devient le masque de visualisation final qui ne contient que les objets saillants suite aux opérations de multiplication
7. Ce masque final est injecté dans le réseau des couches complètement liées, cette partie du réseau a comme mission la prédiction de l'angle de commande.

Le résultat de l'exploitation cet algorithme sur différentes images est montré à la Figure 4.11.

Fig.4.8 Exemples de détection d'objets saillants sur différentes images

On remarque que les objets saillants détectés par cet algorithme sont corrects et sont ceux qui doivent influencer la génération de la commande pour la navigation autonome.

3.4 La prédiction de l'angle de braquage pour la commande

Le système est censé effectuer une tache de régression, i.e. : générer une valeur réelle à la sortie du réseau de neurones. Après l'extraction des formes et la détection des objets saillants effectuées au niveau des couches de convolution, les couches complètement liées appliquent l'algorithme du perceptron pour ajuster les paramètres du réseau avec un optimisateur tel que ADAM que nous allons utiliser dans le chapitre 5 lors de la phase de simulation.

L'optimisateur ADAM est une extension de la méthode de la descente du gradient stochastique. La différence majeure réside dans le taux d'apprentissage, pour la SGD le taux d'apprentissage est fixe pendant toute la phase d'apprentissage. Par contre, pour la méthode

ADAM le taux d'apprentissage est adaptatif et il est propre à chaque paramètre du réseau il est calculé à partir des estimations du premier et du second moment du gradient.

Cet algorithme d'optimisation calcule un décalage moyen du gradient qui évolue exponentiellement, les paramètres qui contrôlent ce décalage sont spécifiés dans la configuration.

La configuration de l'optimisateur ADAM comporte 4 paramètres :

- **Alpha :** c'est la valeur du pas, la proportion avec laquelle les poids sont mises à jour. Une valeur typique pour alpha est de 0.001 car si on prend par exemple 0.3 (une grande valeur) il en résulte un apprentissage rapide avant que le taux d'apprentissage soit mis à jour. De même, si on prend pour alpha la valeur 10^{-5} (une valeur très petite), il peut en résulter freinage de l'apprentissage durant l'entrainement du réseau.
- **beta1 :** le taux de décalage exponentiel du premier moment estimé
- **beta2 :** le taux de décalage exponentiel du second moment estimé.
- **Epsilon :** c'est une faible valeur petite utilisée afin d'éviter la division par zéro lors de l'implémentation de l'algorithme.

Dans notre projet, et en utilisant cet algorithme sur KERAS, la configuration par défaut est :

Alpha=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08.

L'algorithme ADAM:

1. Entrer : α : le pas
2. Entrer : $\beta_1, \beta_2 \in$: les taux de décalage pour les moments estimés
3. Entrer : $f(\theta)$: fonction d'objectif stochastique de paramètres θ
4. Entrer : θ_0: le vecteur des paramètres initial
5. $m_0 \leftarrow 0$: Initialisation du premier vecteur de moment
6. $v_0 \leftarrow 0$: Initialisation du second vecteur de moment
7. $t \leftarrow 0$: Initialisation du pas temporaire
8. tant que : $\theta_t$ne converge pas, faire :
9. $t \leftarrow t + 1$
10. $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$: calcule de gradient en temps t
11. $m_t \leftarrow \beta_1 . m_{t-1} + (1 - \beta_1) . g_t$: mise à jour du premier moment biaisé et estimé
12. $v_t \leftarrow \beta_2 . v_{t-1} + (1 - \beta_2) . {g_t}^2$: mise à jour du second moment biaisé et estimé
13. $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$: calcule du premier moment estimé et corrigé
14. $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$: calcule du second moment estimé et corrigé
15. $\theta_t \leftarrow \theta_{t-1} - \alpha . \frac{\hat{m}_t}{(\sqrt{\hat{v}_t} + \epsilon)}$: mise à jour des paramètres
16. Stop
17. Return θ_t : les paramètres résultants

4 Avantages et limites de l'apprentissage E2E

Après avoir étudié les bases de l'apprentissage E2E et un exemple d'application pour la conduite autonome d'un véhicule, on va citer dans cette partie les avantages et limites de l'apprentissage E2E.

4.1 Avantages de l'apprentissage E2E

- En fait, le système est entrainé d'une manière générale basée sur un seul principe.
- Chaque étape d'apprentissage est dirigée vers le but final, caractérisé par une fonction objective globale.
- Dans cette approche, on se dispense d'entrainer des modules sur des objectifs auxiliaires sans rapport avec la tâche à effectuer.
- La puissance de l'apprentissage E2E est démontrée par plusieurs exemples, comme celui de la conduite d'un véhicule autonome.
- L'apprentissage E2E est consistent avec l'approche générale de l'apprentissage automatique dans le sens où d'une part on élimine l'influence de l'expérience humaine et d'autre part on résout les tâches à base d'une solution orientée données.

4.2 Limites de l'apprentissage E2E

- La limitation principale, due à l'application de la descente du gradient stochastique, réside dans la convergence lente ou vers un optimum local dans le cas des problèmes mal conditionnés.
- Dans certains cas, les signaux générés par l'apprentissage E2E peuvent être inappropriés. Par exemple, un module responsable pour la représentation visuelle peut être entrainé avec un module très différent en termes de récompense (dans le cas d'un apprentissage par renforcement). Donc, il est préférable d'entrainer un module de vision d'une manière indépendante, soit avec des méthodes d'apprentissage non supervisées ou bien en démarrant d'un réseau déjà entrainé.
- La décomposition du problème peut conduire à des modules qui contiennent des informations de valeur, ces informations sont souvent ignorées dans l'apprentissage E2E ce qui est dangereux dans le cas où ces modules interagissent d'une façon non négligeable car ils peuvent affecter le processus d'apprentissage de chacun d'autre.

5 Conclusion

Dans ce chapitre, on a présenté l'approche d'apprentissage end to end, son développement, ses caractéristiques et un exemple d'application dans le domaine des véhicules autonomes.

On a présenté le projet DAVE-2 de NVIDIA, dans celui-ci, l'approche E2E a été utilisée pour générer la commande. En fait, les résultats obtenus confirment l'efficacité de cette approche pour l'apprentissage des systèmes complexes.

Le projet de NVIDIA est basé sur le PilotNet, un réseau de neurones qu'on entraine avec la méthode VisualBackProp. Le principe de cette dernière méthode consiste à rétro-propager des images au lieu des gradients dans le cas de la méthode de Backpropagation. En effet, le système peut détecter les objets saillants efficacement et sans une intervention humaine explicite.

Enfin, on a cité les avantages et les limitations de l'apprentissage E2E et on peut conclure que cette approche est inefficace pour l'entrainement des réseaux complexes composés de plusieurs modules non triviaux d'où la nécessité d'entrainer le réseau complexe d'une manière structurée, en commençant par les modules simples et indépendants du reste.

Chapitre V

Simulations et Résultats

Le but de cette partie est de créer un réseau de neurones profond capable de reproduire le comportement humain lors de la conduite d'un véhicule. Ce réseau spécialisé est testé, sur un simulateur de la firme Udacity, pour la conduite autonome d'un véhicule. A cet effet, le réseau prend en entrée l'image de la caméra frontale et prédit le braquage.

L'objectif de ce mémoire est de développer un algorithme basé sur l'apprentissage profond pour imiter la conduite d'un conducteur humain. Les données ont été recueillies en conduisant manuellement le véhicule virtuel autour de la première piste via le simulateur Beta Simulator d'Udacity. Par la suite, on a procédé à une augmentation des données pour générer plusieurs échantillons d'entraînement représentant la conduite dans des conditions différentes. Pour valider le réseau neurones déjà entraîné on a testé son aptitude à la conduite autonome d'un véhicule virtuel sur la deuxième piste du simulateur.

1 Implémentation et test d'un algorithme d'apprentissage profond via un simulateur

1.1 Présentation du Beta simulator

Le beta simulator est un simulateur de jeux de Udacity, il comporte deux modes, un mode manuel pour conduire manuellement une voiture virtuelle afin de collecter des données d'entraînement, et un mode autonome pour tester si la voiture peut être conduite de manière autonome pour valider le modèle de réseaux déjà entrainé.

Fig. 5.1Ecran principal du simulateur

D'abord, on choisit une piste en cliquant sur l'une des images de la scène. Pour l'étape d'entraînement, on a sélectionné la première piste (celle de gauche).

Ensuite, on choisit un mode: Mode Entraînement ou Mode Autonome. Dès qu'un mode est sélectionné, une voiture virtuelle apparaît à la position de départ.

Fig. 5.2 Mode entraînement

1.2 Aperçu Code d'UDACITY

- config.py : permet la configuration des hyper-paramètres tels que la taille des images d'entrée, le batchsize, le biais et le nombre des canaux de l'entrée.
- model.py : héberge l'architecture du modèle et le programme de l'apprentissage.
- drive.py : permet la conduite de la voiture en mode autonome.
- model.h5 : héberge le modèle du réseau de neurones de convolution déjà entraîné.

1.3 Collecte de données

Les données sont rassemblées en exploitant le simulateur d'Udacity lui-même. Lorsque le simulateur est en mode d'entraînement, la voiture est contrôlée par un conducteur humain via le clavier, et les données relatives à la scène et la direction de braquage sont stockées sur un disque. Pour ce projet, nous avons constitué un ensemble d'entraînement contenant 41316 échantillons (images). Pour chaque échantillon, les informations recueillies correspondent à l'angle de braquage et aux images issues des trois caméras : frontale, gauche et droite (voir Fig.5.2).

Fig. 5.3 Visualisation d'une scène par les trois caméras installées sur le véhicule

En arrêtant la collecte des données, le fichier contenant les images et leurs commandes correspondantes est généré automatiquement. Une visualisation de ces données montre que malheureusement, il y a un énorme biais dans la distribution des données: comme nous pouvons le voir, la distribution de l'angle de braquage est fortement biaisée vers le zéro ceci est dû à la nature de la trajectoire.

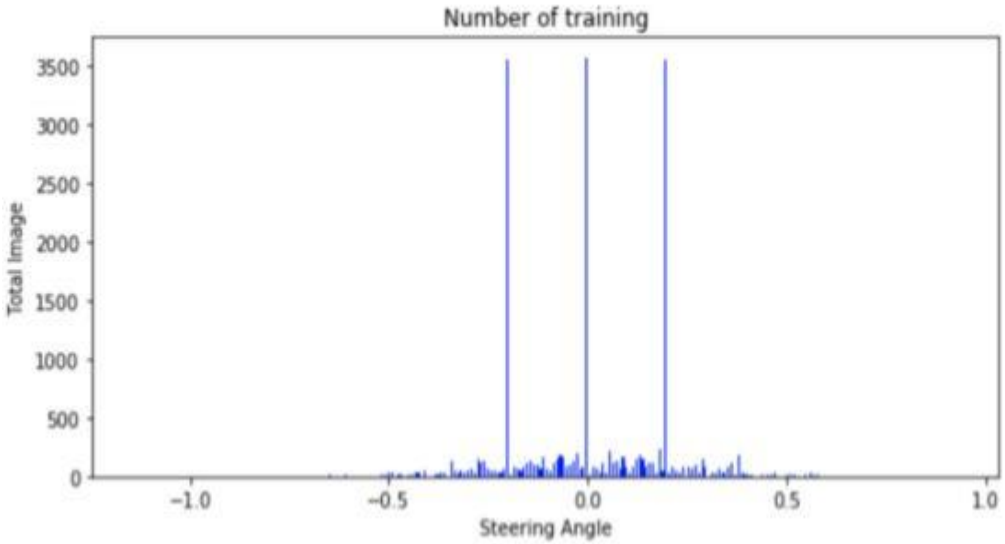


Fig. 5.4 Distribution des angles de braquage relatifs aux données d'entraînement

Une solution à ce problème consiste en une augmentation des données pour rendre gaussienne (normale) la distribution des angles de braquage gaussienne (normale).

1.4 Augmentation des données

En raison du déséquilibre des données susmentionné, il est facile d'imaginer que chaque algorithme d'apprentissage que nous formerions sur les données brutes apprendrait simplement à prédire la valeur de direction 0. En outre, nous pouvons voir que la piste de test « circuit de validation» est complètement différente du circuit d'entraînement d'un point de

vue visuel. Ainsi, un réseau naïvement entraîné sur la première piste ne généraliserait guère la seconde. En outre, diverses formes d'augmentation des données peuvent nous aider à faire face à ces problèmes.

- **Exploiter les caméras gauche et droite.** Pour chaque angle de braquage, nous disposons de trois images capturées respectivement à partir de la caméra frontale, gauche et droite. Les images des caméras latérales peuvent ainsi être utilisées pour augmenter les données d'entraînement, en corrigeant de manière appropriée l'angle de braquage.

  Pour la Figure 5.3, nous avons essayé d'exploiter l'angle de la caméra gauche et droite pour les faire correspondre aux données du panneau central. Pour cela, nous avons inséré un décalage de 0.25 correspondant à un angle de 6,25°. La caméra de droite devrait se déplacer vers la gauche pour venir au centre, et la caméra de gauche devrait se déplacer vers la droite pour arriver au centre. On a donc ajouté 0.25° pour obtenir l'angle de braquage pour l'image de la caméra de gauche et soustrait 0.25° pour obtenir l'angle de braquage relatif à l'image issue de la caméra de droite.

- **Changements de luminosité.** Ensuite, nous avons changé la luminosité des images de la caméra centrale afin que la voiture puisse apprendre à naviguer dans les conditions de jour et de nuit. Pour ce faire, nous avons converti les images RGB en HSV ou TSL en français (teinte, saturation et couleur se sont les trois paramètres de description d'une couleur utilisés dans en graphisme informatique et en infographie) et la valeur du canal est multipliée par une valeur aléatoire entre 0.25 et 1.25 puis on a reconverti les images en RGB. Plus la gamme est large, plus les images augmentées seront différentes de celles d'origine.

```
def augment_brightness_camera_images(image):
    image1 = cv2.cvtColor(image,cv2.COLOR_RGB2HSV)
    random_bright = .25+np.random.uniform()
    image1[:,:,2] = image1[:,:,2]*random_bright
    image1 = cv2.cvtColor(image1,cv2.COLOR_HSV2RGB)
    return image1
```

```
    plt.figure(figsize=(16,8))
for i in range(9):
    image1 = augment_brightness_camera_images(image2)
    plt.subplot(3,3,i+1)
    plt.imshow(image1)
    plt.axis('off')
```

Fig. 5.5 Augmentation des données par modification de la luminosité

Nous avons inversé les images de la ligne médiane verticale pour simuler la conduite dans la direction opposée. Les données collectées relatives à la piste d'entraînement se caractérisent par un nombre important de virages à gauche, donc pour équilibrer le nombre de virages à gauche et avec celui à droite, les images ont été retournées au hasard autour de l'axe vertical. Les anges correspondant aux images retournées ont été multipliés par -1 pour corriger la direction dans la direction opposée.

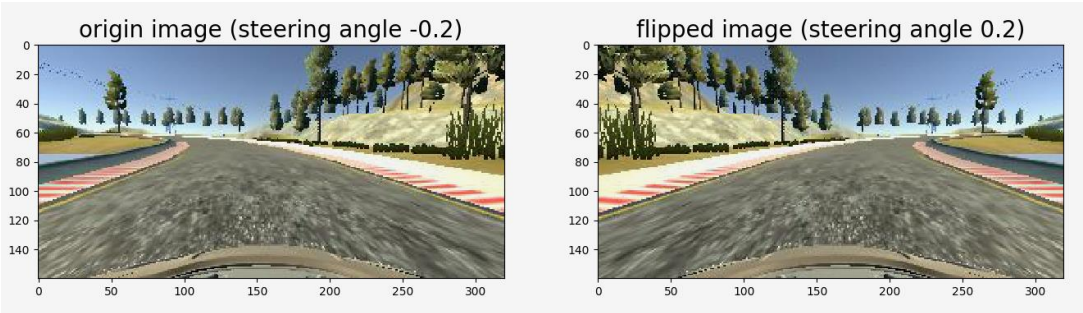


Fig. 5.6 A gauche nous avons l'image d'origine collectée lors de la phase d'entraînement
A droite l'image et l'angle de braquage inversés

Fig. 5.7 Recadrage de l'image

- **Changer la taille des images.** Dans l'image, la partie haute (ciel) et la partie basse (partie avant de la voiture) ne sont pas très utiles pour l'entraînement, en plus, cela peut conduire à un sur-apprentissage. Alors que nous avons décidé de recadrer seulement la partie la plus utile (Fig.5.7), et cela se fait en ajoutant la ligne suivante à notre programme, cette dernière ligne va créer une couche:

$$model.add(Cropping2D(cropping = ((70,25),(0,0))))$$

Après application des procédures précédentes, nous avons abouti à une distribution gaussienne des angles de braquage comme le montre la Figure 5.8

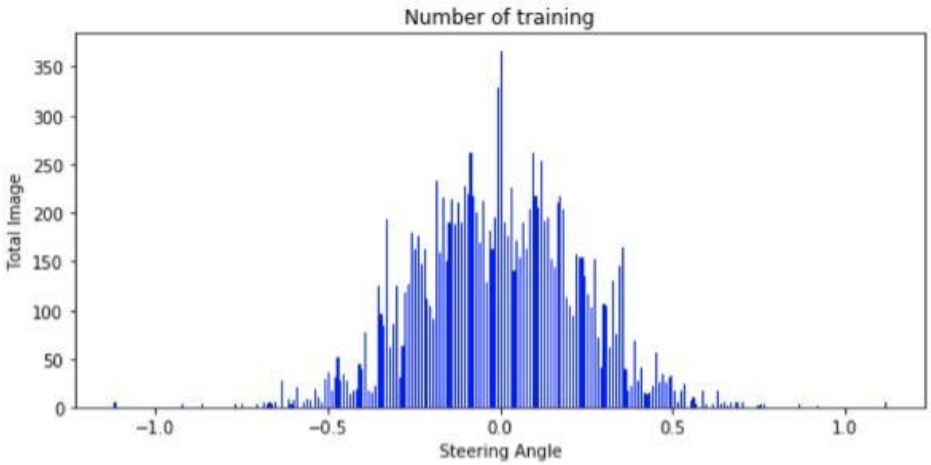


Fig. 5.8 Nouvelle distribution des angles de braquage après augmentation des données

1.5 Architecture du ConvNet

Notre réseau se compose de 6 couches de convolution, 6 couches de mise en commun, des couches ReLu et 3 couches complètement connectée (Voir chapitre 3).

```
model = Sequential()

model.add(Convolution2D(24, 5, 5, subsample=(2, 2), border_mode="same", input_shape=(row, col, ch)))
model.add(Activation('relu'))
model.add(Cropping2D(cropping=((70,25),(0,0))))
model.add(Convolution2D(36, 5, 5, subsample=(2, 2), border_mode="same"))
model.add(Activation('relu'))
model.add(Convolution2D(48, 5, 5, subsample=(2, 2), border_mode="same"))
model.add(Activation('relu'))
model.add(Convolution2D(64, 3, 3, subsample=(2, 2), border_mode="same"))
model.add(Activation('relu'))
model.add(Convolution2D(64, 3, 3, subsample=(2, 2), border_mode="same"))
model.add(Activation('relu'))
model.add(Convolution2D(64, 3, 3, subsample=(2, 2), border_mode="same"))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Activation('relu'))
model.add(Dense(256))
model.add(Activation('relu'))
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dense(1))

model.compile(optimizer="adam", loss="mse")

print('Model is created and compiled..')
return model
```

1.6 Prévenir l'overfitting

Malgré la forte augmentation des données mentionnée ci-dessus, on n'est pas à l'abri d'un overfitting. Afin d'éviter ce problème, nous avons ajouté une couche dropout avec une probabilité p de 0.5.

1.7 Détails de l'entraînement

Avant le lancer l'entraînement du réseau proposé, nous avons réparti nos données en deux parties, la première partie pour entraîner le modèle de réseaux de neurones à effectuer la conduite d'un véhicule et la deuxième partie pour la validation de ce modèle.

Nous avons entraîné le modèle en utilisant la procédure KERAS model.fit() avec des batch size de 32 pour 10 epochs, l'entraînement a nécessité une durée de 1H 30min sur le GPU de mon ordinateur portable (équipé de la carte NVIDIA GeForce GTX 950M).

Pour chaque epoch, sont effectuées les opérations suivantes :

- les paramètres (poids) sont calculés en utilisant l'optimisateur d'Adam et un modèle généré ;

-l'erreur d'entraînement (erreur quadratique moyenne entre l'angle de braquage réel et l'angle prédit) est calculée pour le jeu de données utilisé lors de l'entraînement ;

- l'erreur de validation (erreur quadratique moyenne entre l'angle de braquage réel et l'angle prédit) est calculée pour le jeu de données utilisé lors de la validation.

A chaque fois qu'on obtient une erreur de validation plus faible que l'erreur précédente, les poids de l'ancien modèle sont écrasés et le nouveau modèle est sauvegardé à sa place.

1.8 Test du modèle

A la fin de l'entraînement, nous avons obtenu un modèle avec une erreur d'entraînement faible de 2,12% mais une erreur de validation de 3,01% qui semble élevée par rapport à l'erreur d'entraînement, ce qui suggère que le modèle a été sur-ajusté (overfitting). Donc pour y remédier, nous avons ajouté des couches dropout dans le modèle et nous avons réduit le nombre de neurones dans les couches complètement connectées, le nouveau réseau obtenu est donné ci-dessous :

```
model = Sequential()

model.add(Convolution2D(24, 5, 5, subsample=(2, 2), border_mode="same", input_shape=(row, col, ch)))
model.add(Activation('relu'))
model.add(Cropping2D(cropping=((70,25),(0,0))))
model.add(Convolution2D(36, 5, 5, subsample=(2, 2), border_mode="same"))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Convolution2D(48, 5, 5, subsample=(2, 2), border_mode="same"))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Convolution2D(64, 3, 3, subsample=(2, 2), border_mode="same"))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Convolution2D(64, 3, 3, subsample=(2, 2), border_mode="same"))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Convolution2D(64, 3, 3, subsample=(2, 2), border_mode="same"))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Activation('relu'))
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dense(50))
model.add(Activation('relu'))
model.add(Dense(10))
model.add(Activation('relu'))
model.add(Dense(1))

model.compile(optimizer="adam", loss="mse")

print('Model is created and compiled..')
return model
```

Après, on relance l'entraînement, une fois terminé, nous avons obtenu des erreurs d'entrainement et de validation faibles et proches (une erreur de validation = 2,46% et une erreur d'entraînement = 2,5%) Le réseau peut conduire avec succès sur les deux pistes. De manière assez surprenante, il conduit mieux et plus en douceur sur la piste d'entraînement par rapport à la piste d'essai (moins d'oscillations).

Les deux principaux défis de ce projet reviennent à surmonter les deux difficultés suivantes :

- une répartition asymétrique des données d'entraînement (forte polarisation vers 0) ;
- une faible taille des données d'entraînement obtenues via la conduite d'un véhicule sur une seule piste (risque d'overfitting)

Ces deux défis ont été surmontés, ou du moins atténués, en utilisant une augmentation des données et les dropout. Le principal inconvénient relevé réside maintenant dans le fait que le réseau a quelques difficultés à maintenir le véhicule à filer tout droit: le véhicule tend à osciller un peu. À côté de cet aspect, le réseau est capable d'assurer la conduite du véhicule en toute sécurité sur les deux voies, ne quittant jamais la piste.

2 Implémentation et test de l'algorithme sur les images réelles

2.1 Aperçu du code

- config.py : configuration des hyper-paramètres.
- preprocess_train_data : fait le prétraitement des données d'apprentissage.
- preprocess_test_data : fait le prétraitement des données de test.
- train.py : héberge l'architecture du modèle et le programme de l'apprentissage.
- predict.py : Prédit les angles de braquage.

2.2 La collecte de données

Les données utilisées pour cette partie étaient fournies par Udacity, ces données ont été recueilli pendant une heure de conduite sur les routes américaines (33808 images donc 1.2GB de données).

Ces images ont été réparties en deux phases :

Phase 1 : La première phase était basée sur les données de conduite d'El Camino Real, Californie, (le trajet comporte de faibles virages et principalement des lignes droites), les données de la phase 1 ont été seulement recueillies de la caméra frontale et vont être utilisées dans la phase de test.

Phase 2 : la deuxième phase était basée sur les données de conduite de San Mateo à Half Moon Bay, Californie, (une partie du trajet comporte des virages et l'autre partie du trajet est sur autoroute), ces données vont être utilisées pour entraîner notre réseau.

Les données de la phase 1 sont sous le format image PNG et les données de la phase 2 sont sous le format rosbag.

2.3 Extraction des images des rosbag

Nous avons utilisé Udacity Reader docker tool pour convertir les images des rosbag en image PNG.

Il est à noter que « Docker » est un outil qui peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur. Ceci permet d'étendre la flexibilité et la portabilité d'exécution d'une application, que ce soit sur la machine locale, un cloud privé ou public, etc.

Les scripts ont été configurés pour s'exécuter dans un conteneur Docker afin de pouvoir extraire les données du format rosbag sans le recours à l'installation du ROS. Le conteneur docker est construit à partir du ROS kinetic-perception image.

Les données de la phase 2 sont stockées dans 6 rosbags, qui sont tous mis dans un dossier Data puis il faut extraire les images d'un rosbag à la fois.

Les étapes pour extraire les images des rosbags sont les suivantes :

-Installer le docker.

-Lancer le conteneur docker en utilisant ./build.sh

-Exécuter la commande suivante dans un terminal pour extraire les images d'un rosbag et les mettre dans un dossier qu'on a nommé Output :

$$./run - bagdump.sh \ -i \ /Data \ -o \ /Output$$

Le dossier Output contient lui-même trois dossiers et chaqu'un contient les images provenant d'une seule caméra (gauche, centre ou droite) et un fichier .csv contenant les adresses des images et les angles de braquage correspondants. Même si les images proviennent des trois différentes caméras, nous n'utiliserons que les images provenant de la caméra du centre (33808 images).

2.4 Augmentation de données

Les données utilisées sont relatives à une heure de conduite sur des routes américaines, ces données fournies par UDACITY ne sont pas suffisantes pour établir un algorithme end to end capable de conduire le véhicule ainsi, un prétraitement était nécessaire pour élargir la taille des données.

La première opération consiste à faire redimensionner les images brutes de 640 x 480 pixels à 256 x 192 pixels

La seconde étape de prétraitement consiste à convertir les images de leur format RGB (red, green, blue) en format greyscale (niveaux de gris). Cette opération sert à remplacer les trois valeurs représentant les niveaux de rouge, de vert et de bleu de chaque pixel dans une image par une représentation de la quantité (intensité) de lumière.

La dernière étape du prétraitement consiste à convertir les données stockées dans le fichier steering.csv en un tableau en utilisant la bibliothèque numpy.

Le tableau contient: l'horodatage, l'angle de braquage, la vitesse de la voiture et le chemin complet de l'image correspondante.

```
1 timestamp,angle,speed,fullpath
2 1477429515920298496,-0.0314159281552,4.52121921864,data/train/center/1477429515920298589.jpg
3 1477429515970173696,-0.0314159281552,4.60574369549,data/train/center/1477429515970173809.jpg
4 1477429516020495872,-0.0314159281552,4.69766315084,data/train/center/1477429516020495761.jpg
5 1477429516070363648,-0.0314159281552,4.74745715765,data/train/center/1477429516070363624.jpg
6 1477429516120457216,-0.0330262329939,4.83288763141,data/train/center/1477429516120457269.jpg
```

Pour lancer le prétraitement des données de la phase 2 (données utilisées pour l'entraînement), on exécute la commande suivante:

$$python\ preprocess\_train\_data.py$$

Pour lancer le prétraitement des données de la phase 1 (données utilisées pour le test), on exécute la commande suivante:

$$python\ preprocess\_test\_data.py$$

Maintenant que nos données sont exploitables, on peut les utiliser pour entraîner notre réseau ConvNet.

2.5 Architecture du ConvNet

On va utiliser le deuxième modèle établi dans ci-dessus

Layer (type)	Output Shape	Param #	Connected to
convolution2d_1 (Convolution2D)	(None, 96, 128, 24)	2424	convolution2d_input_1[0][0]
activation_1 (Activation)	(None, 96, 128, 24)	0	convolution2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 48,64, 36)	21636	activation_1[0][0]
activation_2 (Activation)	(None, 48, 64, 36)	0	convolution2d_2[0][0]
convolution2d_3 (Convolution2D)	(None, 24, 32, 48)	43248	activation_2[0][0]
activation_3 (Activation)	(None, 24, 32, 48)	0	convolution2d_3[0][0]
convolution2d_4 (Convolution2D)	(None, 12, 16, 64)	27712	activation_3[0][0]
activation_4 (Activation)	(None, 12, 16, 64)	0	convolution2d_4[0][0]
convolution2d_5 (Convolution2D)	(None, 6, 8, 64)	36928	activation_4[0][0]
flatten_1 (Flatten)	(None, 3072)	0	convolution2d_5[0][0]
activation_5 (Activation)	(None, 3072)	0	flatten_1[0][0]
dense_1 (Dense)	(None, 100)	307300	activation_5[0][0]
activation_6 (Activation)	(None, 100)	0	dense_1[0][0]
dense_2 (Dense)	(None, 50)	5050	activation_6[0][0]
activation_7 (Activation)	(None, 50)	0	dense_2[0][0]
dense_3 (Dense)	(None, 10)	510	activation_7[0][0]
activation_8 (Activation)	(None, 10)	0	dense_3[0][0]
dense_4 (Dense)	(None, 1)	11	activation_8[0][0]

Total params: 444,819
Trainable params: 444,819
Non-trainable params: 0

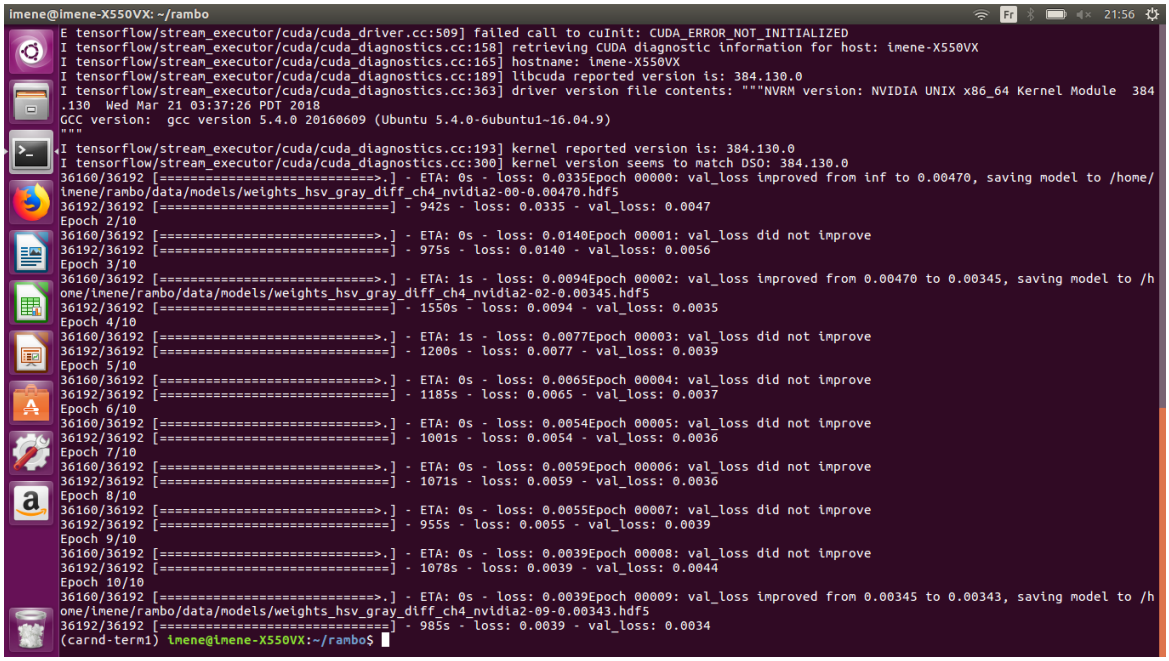
Notre nouveau modèle comporte 444,819 paramètres à estimer contrairement au premier modèle qui ne comportait que 406,957 paramètres.

2.6 Entraînement

On ouvre un terminal et on lance l'entraînement par la commande:

$$python\ train.py$$

L'entraînement a duré 2h49min, on relève que le meilleur modèle est obtenu au dixième epoch où l'une erreur de validation était de 0. 34% et l'erreur d'entraînement était de 0.39%



2.7 Test du dernier modèle

Contrairement à l'implémentation sur simulateur, ici, il n'était pas possible de visualiser la voiture en mode autonome. Donc pour le test, on utilise les images de la phase 1 (5 614 images), les faire passer par le modèle et on enregistrer les angles de braquage prédits par le réseau dans un fichier txt puis on les compare à ceux enregistrés lors de la collecte des données.

Pour prédire les angles de braquage à partir des données de test, on exécute:

$$python\ predict.py$$

En exploitant MATLAB, on trace les angles de braquages réels et prédits puis l'erreur de prédiction.

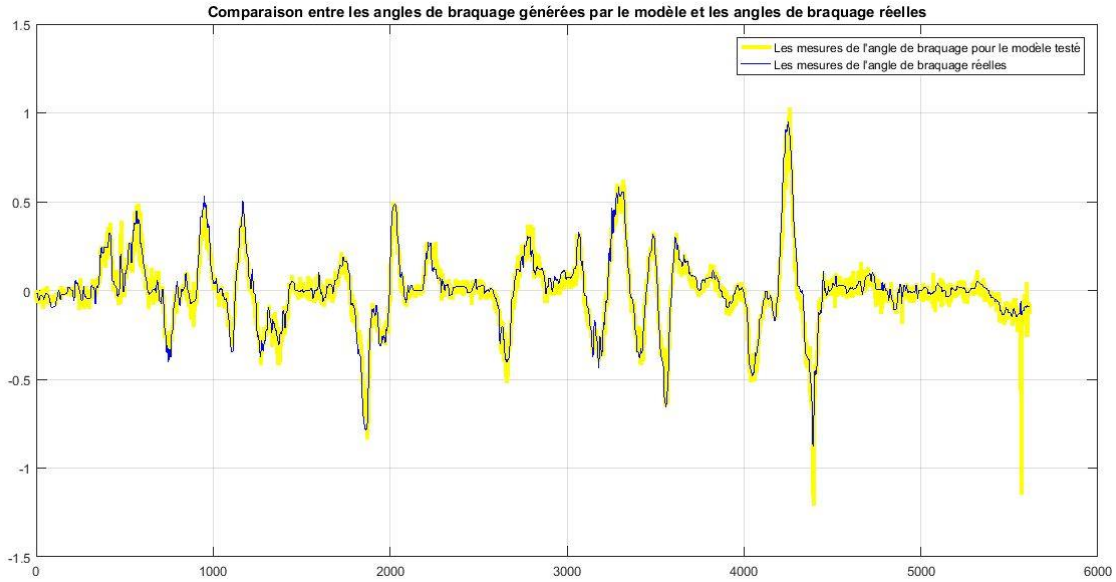


Fig. 5.8 Visualisation des angles de braquage réels et prédits

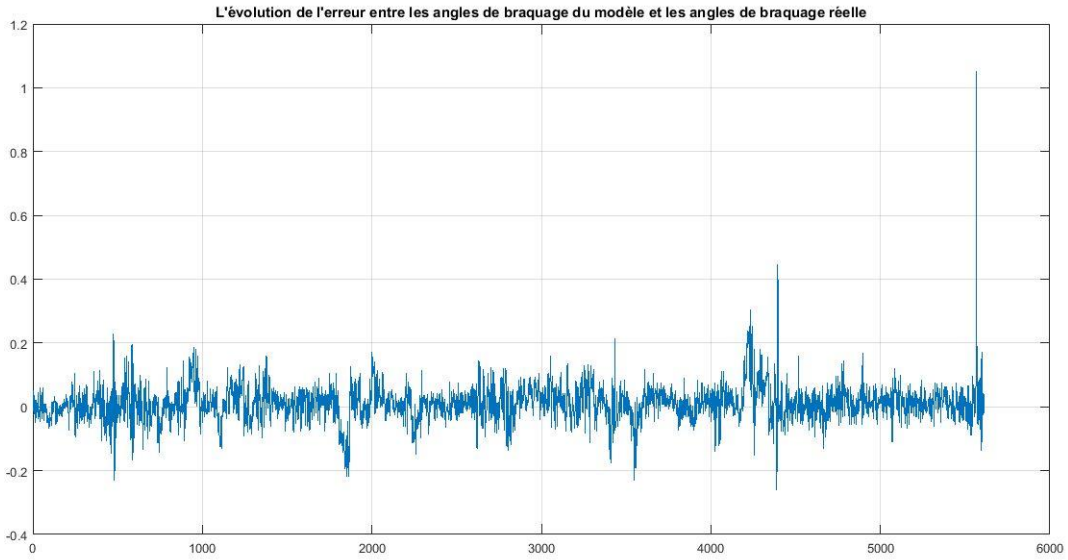


Fig. 5.9 Evolution de l'erreur de prédiction

On relève que notre modèle ConvNet arrive à prédire les angles de braquage avec une erreur au environ de 0.1 quoique les scènes sont plus complexes.

3 Conclusion

Nous avons montré, à travers notre projet, que les ConvNets sont capables d'apprendre à reconnaître et à suivre une voie automatiquement sans avoir recours à décomposer la conduite du véhicule en plusieurs tâches telles que la détection de la route ou de son marquage, la planification du trajet et la commande. Nous avons seulement exploité une faible quantité de données d'entraînement provenant de quelques heures de conduite cependant, le développement du projet a montré qu'il était suffisant pour entraîner le ConvNet à assurer la conduite d'une voiture dans diverses conditions : sur autoroute, sur les routes par temps ensoleillé, nuageux ou pluvieux. Ceci constitue l'avantage principal de l'apprentissage E2E. Ainsi, le ConvNet est capable d'apprendre par lui-même des caractéristiques de route significatives à partir d'un seul signal d'entraînement (l'angle de braquage). Le système apprend par exemple à détecter le contour d'une route sans avoir besoin d'étiquettes explicites pendant l'entraînement.

Conclusion générale

La structure entraînable et multicouche des ConvNet est ce qui les distingue des autres réseaux de neurones. Leurs couches peuvent être personnalisées en ajustant leurs paramètres pour s'adapter au mieux à leur fonction. Les neurones améliorent constamment la précision de leurs sorties en apprenant à partir de chaque donnée d'entrée. Ceci est particulièrement utile pour les applications dans les véhicules autonomes. La capacité des réseaux de neurones de convolution à distinguer à la fois la présence et la profondeur des obstacles fait d'eux l'avenir promoteur du transport autonome.

Ce mémoire s'inscrit dans le cadre du projet du Centre de Développement des Technologie Avancée (CDTA) visant à entraîner un réseau de neurone de convolution à commander le véhicule directement à partir d'une caméra frontale d'où l'appellation de l'approche « End to End Learning». Cette approche End to End s'est révélée étonnamment puissant. Avec peu de données d'entraînement provenant des humains, le réseau apprend à prédire les angles de braquage correctement sur différentes scènes telle que des routes avec ou sans marquage de voie et les autoroutes.

Les résultats des tests effectués, nous ont permis de montrer que les ConvNets sont capables d'apprendre à reconnaître et à suivre une voie automatiquement sans avoir recours à décomposer le processus de navigation du véhicules. En outre, une faible taille de données d'entraînement (quelques heures de conduite) était suffisante pour entraîner le CNN à conduire la voiture dans diverses conditions : autoroutes, routes, par temps ensoleillé ou nuageux. Ceci constitue l'avantage principal de l'apprentissage E2E. Ainsi, le ConvNet est capable d'apprendre des caractéristiques de route significatives à partir d'un seul signal d'entraînement (l'angle de braquage). Le système apprend, par exemple, à détecter le contour d'une route sans avoir besoin d'étiquettes explicites pendant l'entraînement.

Bien que l'apprentissage d'une seule tâche basée sur l'angle de braquage ait donné de bonnes performances, l'angle de braquage seul n'est pas suffisant pour contrôler la voiture en toute situation donc, nous proposons en perspectives de considérer un apprentissage multitâche relatif à la prédiction de l'angle de braquage et au contrôle de vitesse.

Bibliographie

[1]**A.Nielsen Michael** Neural Networks and Deep Learning [En ligne]. - Determination Press, 2015. - 29 03 2018.
Disponible sur http://neuralnetworksanddeeplearning.com.

[2] **Alex Krizhevsky Ilya Sutskever and Geoffrey E Hinton** Imagenet classification with deep convolutional neural networks [Revue] // Proceedings of advances in Neural Information. - 2012.

[3] **David E Rumelhart Geoffrey E Hinton and Ronald J Williams.** Learning internal representations [Livre]. - [s.l.] : Tech. rep., 1985.

[4] **Denis François** [En ligne]. - 25 03 2018.
Disponible sur http:/www.grappa.univ-lille3.fr/polys/intro/index.htm.

[5] **Ian Goodfellow Yoshua Bengio,Aaron Courville** Deep Learning [Book]. - [s.l.] : MIT Press, 2016.
Disponible sur http://www.deeplearningbook.org.

[6] **Kriesel D.** A Brief Introduction to Neural Networks [Livre], 2005.

[7] **Rosenblatt Frank** The perceptron: A probabilistic model for information storage and [Revue] // Psychological review. - 1958. - 65. - pp. 386-408.

[8] **Rougier Nicolas** Biological neuron schema [Représentation], 2007.

[9] **A. Graves G. Wayne, and I. Danihelka** Neural turing machines [Revue] // Technical Report, 2014.

[10] **al. J. Deng and** Imagenet: A large-scale hierarchical image database [Conférence] // IEEE Conference on Computer Vision and Pattern Recognition, 2015.

[11] **al. M. Bojarsk and** End to End Learning for Self-Driving Cars [Livre], 2016.

[12] **al. Mariusz Bojarski and** Explaining How a Deep Neural Network Trained with End to end learning steers a car [Livre], 2016.

[13] **al. Mariusz Bojarski and** VisualBackProp: efficient visualization of CNNs [Livre],2016.

[14] **Diederik Kingma Jimmy Lei Ba** ADAM: a method for stochastic optimization [Conférence] // ICLR, 2015.

[15] **Glasmachers Tobias** Limits of End-to-End Learning [Livre], 2017.

[16] **Lenc A. Vedaldi and K.** Convolutional Neural Networks for MATLAB [Conférence] Proceeding of the ACM Int, 2015.

[17] **Net-Scale Technologies Inc** Autonomous off-road vehicle control using end-to-end learning [Conférence], 2004.

[19] **Pomerleau Dean A.** ALVINN, an autonomous land vehicle in a neural network [Conférence], 1989.

[20] **Pomerleau Dean A.** EfficientTraining of Artificial Neural Networks for Autonomous vehicules [Revue] // Technical report,Carnegie Mellon University, 1989.

[21] **M. Nielsen.** "Neural Networks and Deep Learning." Creative Commons, 2017. Disponible sur http://neuralnetworksanddeeplearning.com/index.html

[21] **E. Weisstein.** "Convolution." Wolfram Math World. n.d, 2017

[22] **Wu.** "Introduction to Convolutional Neural Networks."National Key Lab for Novel Software Technology, Nanjing University. 2016.

[23] **RSIP Vision.** “Deep Learning and Convolutional Neural Networks: RSIP Vision Blogs.” n.d, 2017

[24] **Arun Manglick** “Artificial Inteligence & Deep Learning” , 2017