République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université YahyiaFares de Médéa

Faculté de Technologie
Département de Génie Electrique
Mémoire de Fin d'Etudes pour l'obtention du diplôme de Master
En Génie Electrique

Option : Systèmes de Télécommunications

THEME

Un concept d'apprentissage incremental pour le traitement des données dans des réseaux neuronaux profonds

Etudié par:

-Mr.DEBIEB Mohamed Redha

-Mr.DALIBRAHAM Nouraddine

Promoteur :

-Mr. SAID Bilel

Co-Promoteur :

-FERGANI Nesrine

Promotion:2016-2017

Remerciement

Nous tenons tout d'abord à remercier ALLAH pour nous avoir guidés vers le bon chemin de la lumière et du savoir et pour nous avoir donné du courage, de la volonté afin de pouvoir réaliser ce modeste travail.

ET après nous remercions nos parents pour leurs aides.

Nous tenons à exprimer notre profonde gratitude à notre promotrice

Melle FERGANI pour l'encadrement de ce mémoire pour son aide, sa patience et la confiance qu'il nous a accordée.

Nos sincères remerciements vont également à tous les enseignants du département de Génie électrique de l'université de Médéa, et à tous les enseignants qui ont participé à notre formation.

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

Sommaire …….. I

Liste des figures ……..... IV

Liste des tableaux ……..... V

Abréviations et symboles ……...…. VI

Chapitre I : Généralité

I.1. Introduction : ..... **Erreur ! Signet non défini.**

I.2. Classification des images : ..... **Erreur ! Signet non défini.**

I.3.1. Classification supervise: ..... **Erreur ! Signet non défini.**

1.3.2. Classification non supervisé :..... **Erreur ! Signet non défini.**

I.4. Jeux de donnée (Dataset) :..... **Erreur ! Signet non défini.**

1.4.1. Les animaux avec les attributs de données : ..... **Erreur ! Signet non défini.**

I.4.2 Classification à grains fins :..... **Erreur ! Signet non défini.**

I.4.3. CIFAR-10 and CIFAR-100 Datasets:..... **Erreur ! Signet non défini.**

I.4.4. Places205 Datasets:..... **Erreur ! Signet non défini.**

I.4.5. Image Net Dataset(ILSVRC) :..... **Erreur ! Signet non défini.**

I.5. Réseau neuronal:..... **Erreur ! Signet non défini.**

I.5.1. Perceptron multicouche (MLP) : ..... **Erreur ! Signet non défini.**

I.5.2. Retro-Propagation (back-propagation BP) : ..... **Erreur ! Signet non défini.**

I.5.3. Réseaux de neurones récurrents:..... **Erreur ! Signet non défini.**

I.6. Réseau neuronal profond :..... **Erreur ! Signet non défini.**

I.6.1. Réseau neuronal convolutif : ..... **Erreur ! Signet non défini.**

a. Couche convolutionnelle : ..... **Erreur ! Signet non défini.**

b. Couche de regroupement :..... **Erreur ! Signet non défini.**

c. Couche complétement connectée : ..... **Erreur ! Signet non défini.**

I.6.2. Réseaux de croyances profondes : ..... **Erreur ! Signet non défini.**

I.6.3. Réseaux neuronaux récurrents: ..... **Erreur ! Signet non défini.**

I.7. Infrastructure de développement à apprentissage profond :..... **Erreur ! Signet non défini.**

I.7.1. Caffe :..... **Erreur ! Signet non défini.**

I.7.2 . MatConvNet :..... **Erreur ! Signet non défini.**

I.7.3. OverFeat :..... **Erreur ! Signet non défini.**

Conclusion : ..... **Erreur ! Signet non défini.**

Chapitre II : Une formation d'apprentissage progressif dans DNN_s

II.1. Introduction : ..... **Erreur ! Signet non défini.**

II.2. Notre concept d'apprentissage progressif en profondeur: ..... **Erreur ! Signet non défini.**

II.3. Génération de jeux de données ImageNet-Minuscule: **Erreur ! Signet non défini.**

II.3.1. Caffe : ..... **Erreur ! Signet non défini.**

a. Pourquoi Caffe : ..... **Erreur ! Signet non défini.**

b. Matériels :..... **Erreur ! Signet non défini.**

c. Les commandes de Caffe : ..... **Erreur ! Signet non défini.**

II.3.2. Génération des Datasets ImageNet-Tiny :..... **Erreur ! Signet non défini.**

II.4. Redimensionnement des réseaux profonds pour la formation incrémentielle : ..... **Erreur ! Signet non défini.**

II.5. Formation des réseaux profond:..... **Erreur ! Signet non défini.**

II.6. Les réseaux avec filtres à l'échelle : ..... **Erreur ! Signet non défini.**

II.6.1. Couches convolutionnelle: ..... **Erreur ! Signet non défini.**

II.6.2. Couche de produit interne : ..... **Erreur ! Signet non défini.**

II.7. Ajout des filtres convolutionnels : ..... **Erreur ! Signet non défini.**

II.7.1. Initialisation à Zero : ..... **Erreur ! Signet non défini.**

II.7.2. Initialisation du basculement des filtres existants : ..... **Erreur ! Signet non défini.**

II.8. Réglage fin : ..... **Erreur ! Signet non défini.**

II.9. Conclusion :..... **Erreur ! Signet non défini.**

Chapitre III : Expériences et résultats

III.1. Introduction :..... **Erreur ! Signet non défini.**

III.2. Évaluation de l'apprentissage progressif avec des filtres échelonnés :..... **Erreur ! Signet non défini.**

III.3. Expériences sur l'ajout et l'initialisation de nouveaux filtres convolutionnels : ..... **Erreur ! Signet non défini.**

III.7 Évaluation de l'apprentissage incrémentiel avec des filtres ajoutés : ..... **Erreur ! Signet non défini.**

III.8 conclusion : ..... **Erreur ! Signet non défini.**

Conclusion générale : 65

Annexe

Bibliographie

Chapitre I : Généralité sur les DNN

Fig I.1: Le pipeline de classification d'image 5

Fig I.2 : Comparaison entre l'apprentissage supervise et l'apprentissage non supervis. 6

Fig I.3 :Exemples du jeu de données Places205 10

Fig I.4 : Exemples de l'ensemble de données de classification d'image ILSVRC. 11

FigI.5 :Un MLP simple avec une couche cachée..... 14

Fig I.6 : un exemple de RNN avec une couche cachée 17

Fig I.7 :Comparaison entre les réseaux de neurones peu profonds et profonds. 18

Fig I.8 : couche convolutionnelle 19

Fig I.9:structure de couche de regroupement. 20

Fig I.10 :Structure d'une couche complètement connectée. 21

Fig I.11 :Une structure simple de DBN 23

Chapitre II : *Une formation d'apprentissage progressif dans DNN_s*

Fig II.1 La relation entre les images à basse et haute résolution 29

Fig II.3 : Gauche : calcul de couche et connexions; 32

Droite: un exemple de réseau Caffe..... 32

Fig II.4 : Un exemple de l'image pleine grandeur de classe du renard utilise pour générer de minuscules images 8*8 , 16*16 et 32*32 35

Fig II.4 :Résultats de la formation I_8 avec net-1, net-2, net-3 et net-4 pour l'ensemble de données ImageNet..... 37

FigII.5 : Architecture Net-3 39

Fig II.6 : Les détailles Caffe lors de l'initialisation du réseau 40

Fig II.7: Les détailles Caffe dans la sortie pendant l'entraînement..... 40

Fig II.8 :Les détailles Caffe pendant l'entraînement et les tests. 40

Fig II.9 : Les détailles Caffe pendant l'achèvement de l'entraînement. 41

Fig II.10 : Paramètres de transformation en couche convolutionnelle de ***DNNs*** 44

Fig II.11 : Paramètres de transformation dans les couches de produits internes des DNNs 47

Fig II.12 : Les premières couches convolutives avec filtres par Krizhevsky [5]. 50

Fig II.13 : Feuilletage des filtres. 51

Fig II.14 : Sortie du journal Caffe lors du réglage fin 52

Chapitre III : *Expériences et résultats*

Fig III.1 :Comparaison des courbes de formation de la méthode traditionnelle et de l'apprentissage progressif avec des filtres à l'échelle sur l'ensemble de données ImageNet. 57

Fig III.2 : Comparaison des courbes de formation de la méthode traditionnelle et de l'apprentissage progressif avec des filtres à l'échelle sur l'ensemble de données ImageNet. 58

Fig III.3 : Comparaison des courbes de formation de la méthode traditionnelle et de l'apprentissage progressif avec des filtres à l'échelle sur le jeu de données Places205. 60

Fig III.4 : Résultats lors de l'initialisation de nouveaux filtres ajoutés comme basculant horizontalement, verticalement, en diagonale et centralement sur l'ensemble de données ImageNet. 61

Fig III.5 : Résultats de la méthode traditionnelle et de l'apprentissage progressif avec des filtres ajoutés et à l'échelle pour l'ensemble de données ImageNet. 62

Fig III.6 :Résultats de la méthode traditionnelle et de l'apprentissage progressif avec des filtres ajoutés et à l'échelle pour l'ensemble de données Places205. 64

Chapitre II :Une formation d'apprentissage progressif dans DN

Tab II.1 : La procédure de mise à l'échelle des DNN pour l'apprentissage progressif. 36

Tab II.2 :Résumé des paramètres en couche convolutionnelle dans DNN_8, DNN_{16} et DNN_{32}. 37

Chapitre III : Experiences et résultats

Tab III.1:Précision finale de la méthode traditionnelle et apprentissage progressif avec des filtres à l'échelle sur l'ensemble de données ImageNet. 59

Tab III.2 : Précisions finales de la méthode traditionnelle et apprentissage progressif avec des filtres à l'échelle sur le jeu de données Places205..... 60

Tab III.3 : Précision finale de la méthode traditionnelle et apprentissage progressif avec des filtres ajoutés sur l'ensemble de données ImageNet. 63

Tab III.4 : Précision finale de la méthode traditionnelle et de l'apprentissage progressif avec des filtres ajoutés sur le jeu de données Places205. 64

Abréviations et Symboles

ANN **A**rtificial **N**eural **N**etwork (Réseau neuronal artificiel)

AwA **A**nimals**w**ith**A**ttributes (Animaux avec Attributs)

BP **B**ack-**P**ropagation (Propagation arrière)

BPTT **B**ack-**P**ropagation **T**hrough**T**ime (Propagation temporelle à travers le temps)

CNN **C**onvolutional**N**eural **N**etwork (Réseau neuronal convolutionelle)

DBF **D**eep **B**elief **N**etwork (Réseaucroyance profound)

DNN **D**eep**N**eural **N**etwork (Réseau neurnalprofound)

DRNN **D**eep**R**ecurrent**N**eural **N**etwork (Réseau neurnalRécurrent profond)

EKF **E**xtended **K**alman**F**iltering (Filtrage étendu de Kalman)

ILSVRC **I**mageNet**L**arge **S**cale**V**isual **R**ecognition **C**hallenge (Défi de reconnaissance visuelle à grande échelle ImageNet)

MLP **M**ultilayer**P**erceptron (Perceptron multicouches)

RBM **R**estricted**B**oltzmann **M**achines (Machines à boltzmann restreintes)

RNN **R**ecurrent**N**eural **N**etwork (Réseau neuronal récurrent)

RTRL **R**eal-**T**ime **R**ecurrent **L**earning (Apprentissage récurrent en temps réel)

α La constante de élan

$\delta_i(n)$ Le gradient local du j^{eme} dans l'itération n

Δw_{ij} La correction du poids

H Le taux d'apprentissage

φLa fonction d'activation

$b_i^{(h)}$La polarisation du j^{eme}node en couche h

b Le veteur de biais

DNN_8DNN utilisé en I_8

hL'ensemble des noeuds cachés

h_iLe j^{eme} nœud caché

I_8L'ensemble de données de 8x8 images

I_{16S}L'ensemble de données de 16x16 échantillons d'images

K_2Une matrice de filtre 2x2

mLe nombre de signaux d'entrée

M_8Le modèle formé pour I_8 utilisant DNN_8

M_{16t}Le modèle transformé pour I_{16} au réglage fin

M_i^{n-1}La J^{eme} carte des éléments d'entrée

m_i^nLa J^{eme} carte de sortie

nLe nombre d'itérations

pLe nombre de nœuds de sortie

qLe nombre de noeuds cachés

tTemps

vLe champ local induit du neurone

$\mathbf{w_{ij}^{(h)}}$Le poids du J^{eme}nœud connecté au i^{eme}noeud dans la couche h

$\mathbf{W}$La matrice des poids

$\mathbf{x_i}$La i^{eme} entrée

$\mathbf{x}$Le vecteur d'entrée

$\mathbf{y_i}$La i^{eme} sortie

$\mathbf{y}$Le vecteur de sortie

$\mathbf{Y}$La matrice de sortie

$\mathbf{z^{-1}}$Retard d'unité

Introduction générale

La vision par ordinateur tente d'imiter le système visuel humain en général et extrait des informations utiles de l'entrée telles que des images ou des séquences vidéo. Cela s'est révélé une tâche étonnamment difficile, elle a occupé des milliers d'esprits intelligents et créatifs au cours des dernières décennies, et malgré cela, nous sommes encore loin d'être en mesure de construire une "machine à voyants" à usage général [1]. Sur la base de cela, la classification de l'image est l'une des tâches essentielles de la vision par ordinateur. Il traite des images dont le contenu est prédit en utilisantun classificateur représenté par les fonctions.

Grâce aux nouvelles technologies, il y a une croissance rapide des données dans le monde réel. Cependant, de nombreux algorithmes d'apprentissage en machine nécessitent plus de ressources qui augmentent plus rapidement que les données. En d'autres termes, la majorité des méthodes d'apprentissage par machine, en particulier pour gérer les ensembles de données à grande échelle, nécessitent toujours plus de temps pour former un modèle. Par conséquent, il est nécessaire d'élaborer de nouvelles approches d'apprentissage pour gérer les données de formation à grande échelle afin d'améliorer l'efficacité.

Dans ce mémoire , nous explorons une méthode pour l'amélioration de la classification de l'image. Dans ce qui suit, il y a plusieurs objectifs principaux du travail de mémoire , ainsi que la structure de le mémoire .

L'objectif principal est de proposer une nouvelle méthode basée sur des réseaux neuronaux profonds (DNN), l'apprentissage progressif, afin d'améliorer la performance et l'efficacité de la classification de l'image. L'apprentissage incrémental en utilisant

DNNs est une direction de recherche difficile, qui n'a pas été étudiée dans la littérature existante.

Le premier but de cette recherche est d'apprendre les réseaux de neurones avec les DNN et de comprendre leurs principes de travail. Ceci est le fondement théorique, qui comprend l'apprentissage de la façon dont chaque couche fonctionne. Et comment les paramètres ont un effet sur l'entrée et la sortie.

En outre, le deuxième but est de mettre en œuvre DNN par Caffe, qui est l'une des boîtes à outils populaires. Apprendre à l'installer, l'utiliser et extraire les poids des neurones des DNN peut être utile pour les expériences

Enfin. Basé sur la théorie et avec l'utilisation de la boîte à outils. Nous devons les employer pour trouver la relation entre les poids grossiers et finis des filtres, les transformer. Et les mettre en œuvre avec Caffe pour atteindre notre objectif principal.

Le chapitre I décrit les revues de la littérature sur la classification de l'image, plusieurs ensembles de données populaires, réseaux de neurones, DNN et boîtes à outils connexes. Le chapitre II présente les travaux préparatoires, qui consistent en l'apprentissage progressif, les commandes et la composition du réseau Caffe, et le prétraitement des données et la définition de l'architecture réseau avant de se transformer. Le chapitre III présent les détails des expériences. Ceux-ci consistent en des résultats, des analyses et des comparaisons entre différentes méthodes. Enfin, la conclusion du travail de mémoire.

Chapitre I :
Généralités sur les DNN

I.1. Introduction :

Dans ce chapitre on va commencer par la classification des images dans l'apprentissage profond. Ensuite, nous présentons quelques jeux de données populaires utilisés dans les travaux récents. En plus, les théories de base des méthodes utilisées sont présentées, qui incluent différents types de réseaux de neurones. Enfin, certaines Framworks « infrastructure de développement » tendances pour la formation des réseaux neuronaux sont présentées. afin de mieux comprendre le travail qui va suivre dans les chapitres suivant.

I.2. Classification des images :

Dans l'intelligence artificielle, l'apprentissage par machine est l'un des sujets de recherche important, et la classification est l'un des problèmes les plus fondamentaux dans ce domaine. La classification de l'image consiste à affecter une image d'entrée à une étiquette d'un ensemble fixe de catégories en fonction du contenu de l'image, comme la reconnaissance du visage dans la biométrie [2] et le système d'inspection dans l'usine de produits [3]. Un algorithme qui implémente le classement est connu comme un classificateur, qui mappe les données d'entrée dans une catégorie [4].

Il existe de nombreuses techniques procéder pour complétez la tâche de classification d'image. Le pipeline le plus populaire de la classification des images

repose sur l'hypothèse que l'image représente une ou plusieurs fonctionnalités et que chacune de ces fonctionnalités appartient à l'une des nombreuses classes distinctes et exclusives [5]. Montré à la figure I.1 Les classes peuvent être spécifiées a priori par un analyste (comme dans une classification surveillée ou regroupées automatiquement) dans des ensembles de classes de prototypes, où l'analyste spécifie simplement le nombre de catégories souhaitées [6].

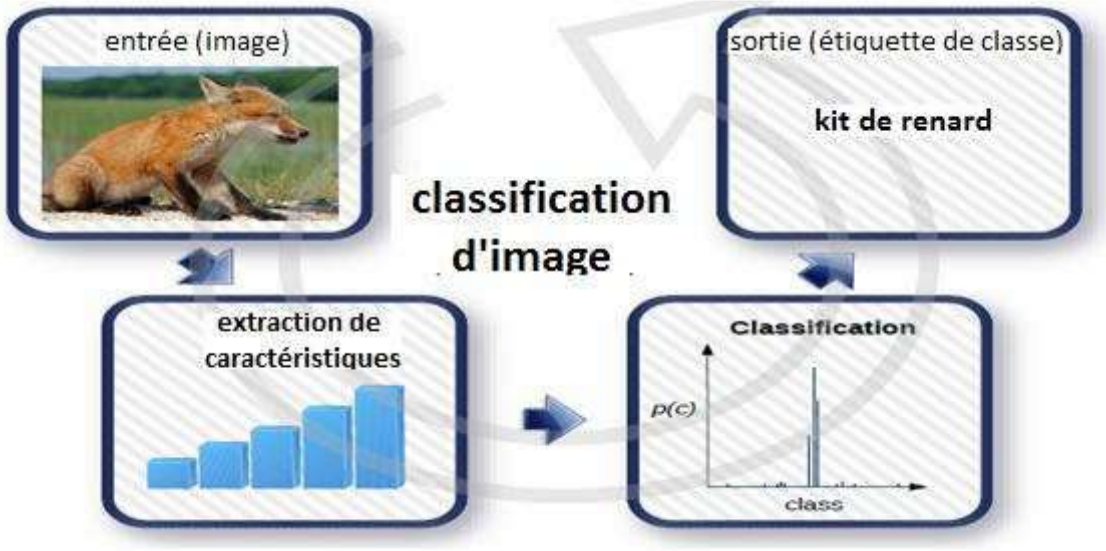


Fig I.1: Le pipeline de classification d'image

La classification de l'image est une tâche essentielle pour les humains et les ordinateurs. C'est l'une des formes d'analyse de données qui peuvent être utilisées pour extraire un modèle ou un classificateur construit pour prédire les étiquettes catégoriques [7]. Un ensemble de formation, un ensemble de tests et un ensemble de validation sont exploités pour découvrir et évaluer les relations du modèle. Par exemple, si le classificateur le plus approprié est exploré, l'ensemble de formation est utilisé pour former les algorithmes candidats, le jeu de validation est utilisé pour comparer leurs performances et décider lequel prendre, et l'ensemble de test est utilisé pour obtenir la performance caractéristique telle que la précision [8]. Généralement, les techniques de classification des images sont constituées d'apprentissage supervisé et d'apprentissage non supervisé, illustré à la figure I.2.

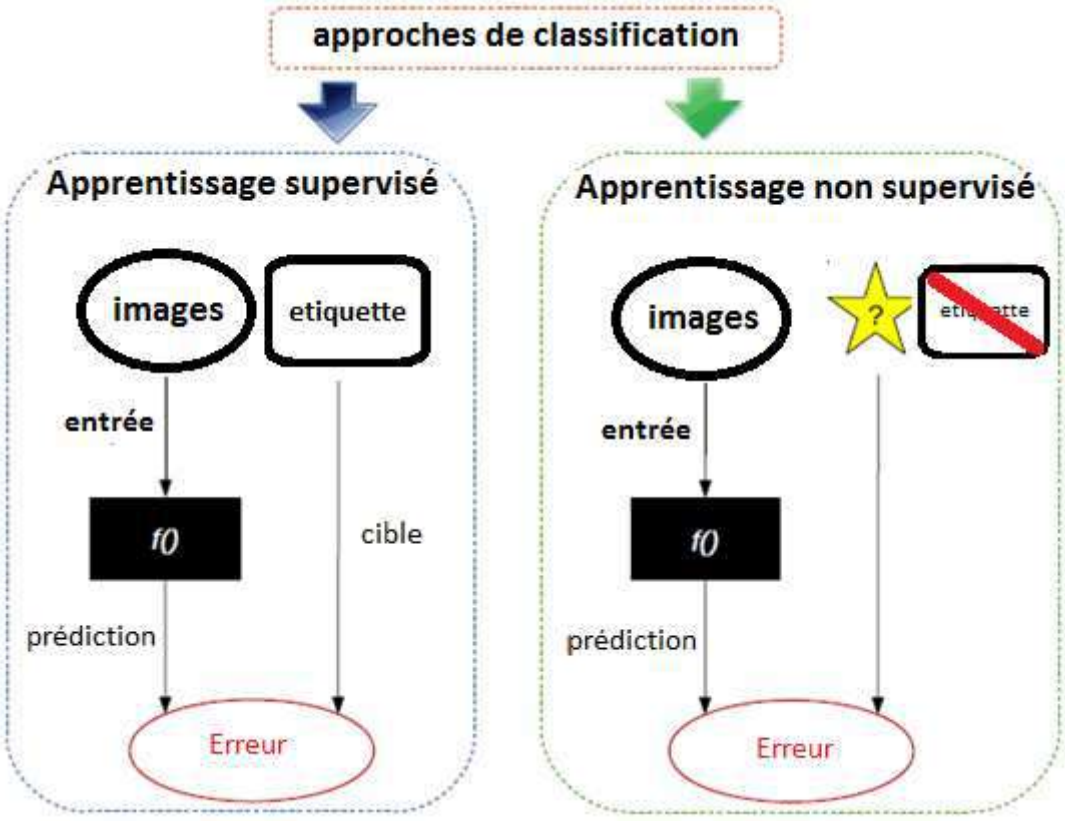


Fig I.2 : Comparaison entre l'apprentissage supervise et l'apprentissage non supervisé .

I.3.1. Classification supervise:

Sur le plan conceptuel, l'apprentissage supervisé est considéré comme la connaissance de l'environnement où la connaissance est désignée comme un ensemble d'exemples d'entrées-sorties. En d'autres termes, chaque donnée de formation est une paire d'entrée et sa sortie cible correspondante. Au cours de la phase de formation, un algorithme d'apprentissage construit le classificateur en analysant ou en «apprenant» d'un ensemble de formation composé de bases de données et de leurs étiquettes de classes associées. La sortie souhaitée représente l'action optimale qui est effectuée par le classificateur. Les paramètres d'un classificateur sont appliqués pour minimiser l'erreur entre les étiquettes prédites et la vérité au sol. La connaissance de l'environnement disponible dans l'ensemble est transférée au classificateur à travers la formation aussi complètement que possible.

Lorsque cette condition est atteinte. Nous pouvons alors nous dispenser de l'entraînement et laisser le classificateur traité complètement l'environnement par lui-même [2]. Les expériences de le mémoire sont basées sur l'apprentissage supervisé pour faire face à la classification de l'image.

1.3.2. Classification non supervisé :

Au contraire, une autre solution consiste à utiliser les données sans étiquette de classe de chaque ensemble de formation dans un apprentissage non supervisé. Dans la phase de formation, il essaie d'associer automatiquement l'information de l'entrée à une réduction intrinsèque de la dimension des données ou de la quantité totale de données d'entrée [16] avec la similitude entre les échantillons de formation. Une fois que le réseau a été réglé sur les régularités statistiques des données d'entrée, il développe la capacité de former des représentations internes pour encoder les fonctions de l'entrée et ainsi créer de nouvelles classes automatiquement [2].

I.4. Jeux de donnée (Dataset) :

Avec le développement de la vision par ordinateur, le nombre croissant d'ensembles de données est collecté pour résoudre la tâche de la classification de l'image. L'ensemble de données peut être utilisé pour résoudre différents problèmes de classification, tels que la classification basée sur l'attribut, les catégories d'objets, la taille et la quantité d'images. Il existe plusieurs jeux d'images largement utilisés. Et ces deux derniers sont utilisés dans cette mémoire.

1.4.1. Les animaux avec les attributs de données :

Le jeu de données Animaux avec attributs (AwA) [17, 18] est un ensemble de données pour la classification basée sur les attributs avec des cours de formation et de test disjoints. La répartition des cours n'est pas faite au hasard, mais une grande partie de la diversité des animaux dans l'ensemble de données (eau / terrestre, domestique sauvage, etc.) se reflète dans la formation ainsi que dans l'ensemble de tests [18]. Il est aligné avec les 50 classes d'animaux Osherson [19] et Kemp [20] en interrogeant les

moteurs de recherche de Google, Microsoft, Yahoo et Flickr, fournissant ainsi 85 valeurs d'attributs numériques pour chaque classe. La collection comprend 24 295 images de 40 classes pour la formation et 6 180 images des 10 autres classes pour les tests. Chaque image comporte six représentations de fonctionnalités pré-extraites. Il se forme en combinant les images collectées avec la matrice d'attributs. L'ensemble de données peut être servi comme apprentissage de données zéro en vision par ordinateur. Et il est identifié sur la base d'une description de haut niveau qui est exprimée en termes d'attributs ; Tels que la couleur et la forme de l'objet [18].

I.4.2 Classification à grains fins :

Les jeux de données à grain fin sont utilisés pour la catégorisation des objets à grain fin, ce qui signifie distinguer la race des objets d'une image, comme Chihuahua, chien maltais et épagneul Blenheim.

Les ensembles de données comprennent de nombreuses catégories d'avions, d'oiseaux, de voitures, de chiens et de chaussures. Il existe de nombreux ensembles de données utilisés pour la catégorisation visuelle fine, y compris Stanford Dogs Dataset [21], Oxford-IIIT-Pet Dataset [22], Caltech-UCSD Birds200-2011 [23] et Birdsnap [24]. Nous présentons deux exemples de jeux de données à grain fin ci-dessous :

- L'ensemble de données de Stanford Dogs est l'un des ensembles de données à grain fin avec 120 classes. Il y a 20 580 images de chiens annotées avec des étiquettes de bordereaux et d'objets. Les images et les boîtes de délimitation appartiennent à ImageNet [25]. Que nous allons présenter plus tard. La classification des chiens est un problème difficile. Premièrement, il y a peu de variations entre classes. Telles que les différentes catégories de chiens peuvent partager des caractéristiques faciales très semblables, mais diffèrent considérablement dans leur couleur. De plus, les chiens de la même classe pourraient avoir différents âges et même des couleurs. En outre. Une grande proportion d'images entraîne une plus grande variation [21].

- Le Caltech-UCSD Birds-200-2011 Dataset est une autre catégorisation fine utilisée pour distinguer les espèces d'oiseaux. Il contient 11, 788 images provenant de 200 espèces. Chaque image annotée par 15 emplacements, 312 attributs binaires et une case délimitée. Le classement des espèces d'oiseaux est un problème difficile. Parce que différentes espèces d'oiseaux peuvent varier considérablement en forme et en apparence (par exemple, les pélicans et les moineaux). En même temps. Même pour les observateurs d'oiseaux experts, certaines paires d'espèces d'oiseaux sont presque indifféremment visibles, comme les espèces de moineaux. La variance intra-classe est élevée en raison de la variation de l'éclairage et du fond et de la variation extrême de la pose (par exemple, les oiseaux volants, les oiseaux de natation et les oiseaux perchés qui sont partiellement obstrués par des branches) [23].

I.4.3. CIFAR-10 and CIFAR-100 Datasets:

Le jeu de données CIFAR-10 [26] est composé de 60 000 images naturelles de 10 classes. Il existe 5 000 images sélectionnées au hasard de chaque classe en tant qu'entraînement, tandis que les 10 000 images restantes forment l'ensemble de tests. Chaque image est une image RVB 32 x 32. Les dix classes comprennent l'avion, l'automobile, l'oiseau, chat, cerf, chien, grenouille, cheval, bateau et camion.

Il existe une extension de l'ensemble de données CIFAR-10, nommé ensemble de données CIFA [26]. Il comporte 100 classes (étiquette fine) et 20 superclasses (étiquette grossière). Il y a 500 images d'entraînement et 100 images de test pour chaque classe. Chaque image est une image RVB 32 x 32.

I.4.4. Places205 Datasets:

Places205 Dataset [9] est un jeu de données à grande échelle fourni par MIT Laboratoire d'informatique et d'intelligence artificielle (Computer Science and Artificial Intelligence Laboratory). Il est utilisé comme référence des méthodes de reconnaissance de scène. La taille totale de l'ensemble de données est d'environ 2.5 millions d'images de 205 catégories de scènes il y a 2 448 873 images choisis au hasard comme ensemble de formation et les 100 images restantes par catégorie comme

ensemble de validation. Il existe plusieurs milliers d'images distinctes dans l'ensemble de test séparé. Le jeu de données Places205 est conçu pour représenter des lieux et des scènes trouvés dans le monde réel. Exemples illustrés à la figure I.3.

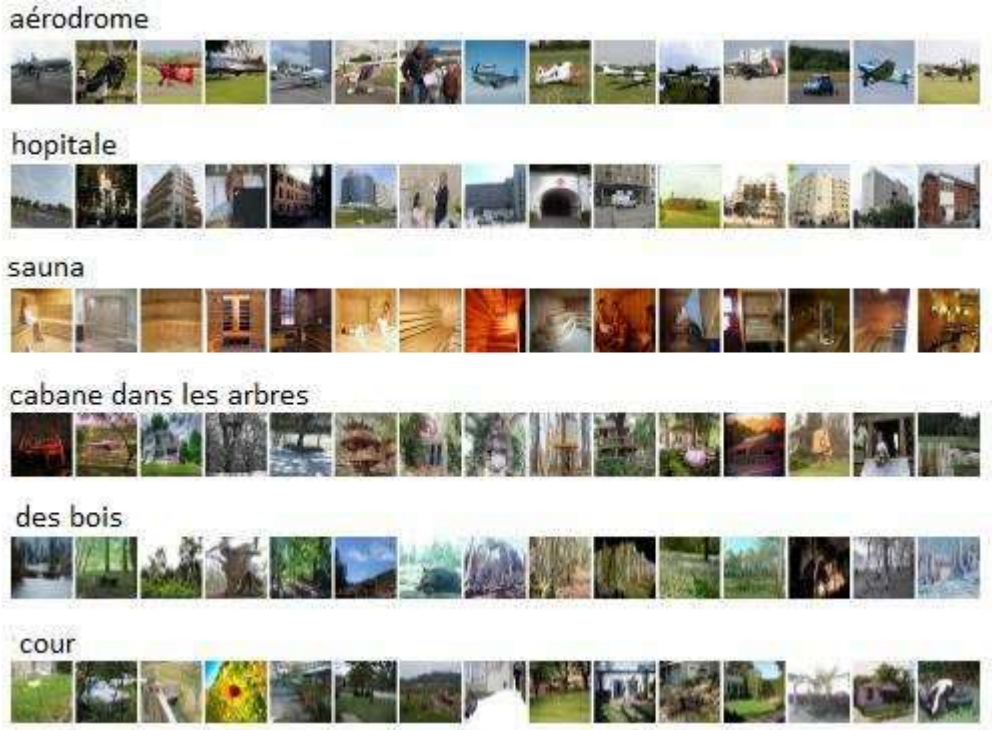


Fig I.3 :Exemples du jeu de données Places205

I.4.5. Image Net Dataset(ILSVRC) :

L'ensemble de données ImageNet [25] est une base de données à grande échelle avec des millions d'images provenant de plus de 20000 catégories. C'est une ressource utile pour la classification des images et la détection des objets.

ImageNet ILSVRC (ImageNet Large Scale Visual Recognition Challenge) [27] est devenu l'indice de référence dans la classification d'image à grande échelle ainsi que la détection d'objets. La principale tâche de ce mémoire est la classification d'image, et nous choisissons donc le jeu de données de classification d'image fourni par ILSVRC comme cible expérimentale. Il s'agit d'un sous-ensemble de ImageNet et collecté à partir d'Internet via plusieurs moteurs de recherche d'images. L'ensemble de données se compose de 1000 classes, et chaque image contient une étiquette de vérité .

Il y a 1,2 million d'images pour la formation, 50 000 images pour la validation et 100 000 images pour les tests. La figure 1.4 montre les exemples d'images provenant de 5 classes d'ensemble de données de classification d'image ILSVRC.



Fig I.4 : Exemples de l'ensemble de données de classification d'image ILSVRC.

I.5. Réseau neuronal:

Un réseau neuronal est une méthode d'apprentissage par machine distribuée massivement parallèle qui contient des unités de traitement simples, qui a une propension naturelle à stocker les connaissances et à les mettre à disposition [28], également appelé réseau neuronal artificiel (ANN). En d'autres termes, il a la puissance de calcul grâce à la structure répartie en parallèle et à l'aptitude à apprendre.

En gros, l'idée générale d'un réseau de neurones est inspirée par des réseaux de neurones biologiques et est présentée comme une combinaison de neurones qui peuvent calculer des valeurs à partir de l'entrée. Un neurone est défini comme une

unité de traitement de l'information qui est fondamentale pour le fonctionnement d'un réseau neuronal [2]. Cependant, les réseaux de neurones sont motivés par l'objectif d'obtenir des algorithmes d'apprentissage machine très efficaces, indépendamment du fait que ces algorithmes reflètent les processus biologiques [29].

Les réseaux de neurones ont une variété de propriétés et de capacités utiles, comme la non-linéarité, la cartographie des sorties-entrées, de manière adaptative et ainsi de suite [2]. Leur apprentissage peut être robuste aux erreurs dans les données de formation et a été appliqué avec succès à des problèmes tels que l'interprétation de scènes visuelles [30]. Reconnaissance de la parole [3l] et apprentissage des stratégies de contrôle du robot [32, 29].

Dans les sous-sections suivantes, on introduit un type d'architectures de réseau appelées multicouches (MLP), un algorithme commun pour la formation de réseaux neuronaux et le réseau neuronal récurrent (RNN).

I.5.1. Perceptron multicouche (MLP) :

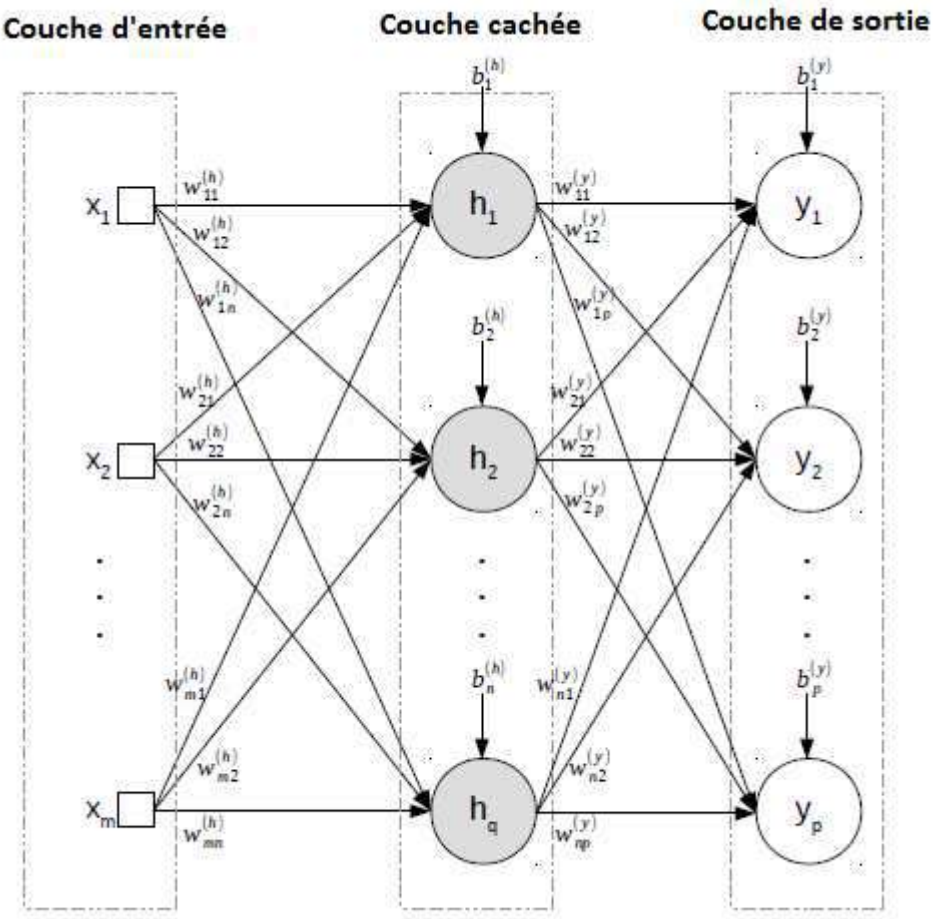
Le but de tout algorithme d'apprentissage supervisé est de trouver une fonction qui correspond adéquatement entre l'entrée et la sortie. Le mappage simple peut être appris par un seul neurone mais il ne peut pas apprendre des données non linéaires séparables. Le MLP surmonte cette limitation car il peut créer des représentations internes en tant que couches et apprendre différentes fonctionnalités dans chaque couche [33]. Un réseau neuronal MLP générique peut s'adapter aux données de formation difficiles présentées par Rumelhart et al. [34]. Généralement, un MLP dont les nœuds sont des neurones avec une activation logistique, est un graphique acyclique dirigé fini. Qui consiste en une couche d'entrée avec des unités sensorielles d'une ou plusieurs couches cachées et d'une couche de dépassement avec des nœuds de calcul. Le signal d'entrée se propage à travers le réseau pour transmettre la direction couche par couche. Il existe trois caractéristiques principales des MLP [2].

- Chaque neurone d'un MLP a une fonction d'activation non linéaire. La présence de non-linéarités est importante car sinon la relation entrée-sortie du réseau pourrait être réduite à celle d'un perceptron à une seule couche [2]. Généralement, le non linéarité sigmoïdale définie par la fonction logistique. Qui est biologiquement motivé est utilisé pour la non-linéarité. Comme :

$$\varphi(v) = \frac{1}{21 + \exp(-v)}$$

- Un MLP contient une ou plusieurs couches de neurones cachés, qui ne font pas partie de l'entrée ou de la sortie. Ces neurones cachés permettent au réseau d'apprendre des tâches complexes en extrayant des fonctionnalités progressivement plus significatives à partir des modèles d'entrée [2] En principe, chaque couche cachée réalise une transformation d'un produit scalaire de la sortie de la couche précédente et le vecteur de poids reliant la Couche précédente à un nœud donné [36].

- Le réseau révèle un haut degré de connectivité, déterminé par les synapses du réseau. Une modification de la connectivité d'un réseau nécessite une modification de la population des connexions synaptiques ou de leurs poids [2].



FigI.5 :Un MLP simple avec une couche cachée

Un MLP simple avec une couche cachée est illustré à la Figure I.5. Ici, nous désignons m signaux d'entrée en tant que vecteur x= $[x_1, x_2, \dots, x_m]^T$ q nœuds caché comme h=$[h_1, h_2, \dots, h_q]^T$, et p nœuds de sortie comme y= $[y_1, y_2, \dots, y_p]^T$ L'indice est configuré pour représenter la couche de paramètres, de sorte que la sortie des neurones cachés h peut être écrite comme :

$$h_j = \varphi \sum_{i=1}^{m} \left(\omega_{ij}^{h}\, x_i - b_j^{(h)}\right)$$

$$y_k = \varphi \sum_{j=1}^{q} \left(\omega_{jk}^{y} h_j - b_k^{(y)} \right) = \varphi \sum_{j=1}^{q} \left(\omega_{jk}^{y} \varphi \sum_{i=1}^{m} \left(\omega_{ij}^{h} x_i - b_j^{(h)} \right) - b_k^{y} \right)$$

De même, ω_{jk}^{y} et $b_k^{(y)}$ présente le poids et la polarisation situés dans la couche de sortie y.De même, la sortie générale de MLP avec des couches plus cachées peut être désignée comme :

$$y_j^{(l)} = \varphi \sum_{i=1}^{m} \left(\omega_{ij}^{l} y_i^{(l-1)} - b_j^{(l)} \right)$$

I.5.2. Retro-Propagation (back-propagation BP) :

L'algorithme BP est la méthode supervisée la plus populaire dans les MLP de formation. Il est décrit comme une méthode d'optimisation du système dynamique à plusieurs étages par E. Bryson et Yu-Chi Ho [37, 38] Depuis 1974, l'algorithme de BP a été développé par Paul Werbos [39] et David E. Rumelhart [33, 40]. L'intention et la motivation pour développer l'algorithme de BP est de trouver un moyen de former un réseau de neurones multicouches qui peut apprendre les représentations internes appropriées pour lui permettre d'apprendre toute cartographie arbitraire de l'entrée à la sortie. La dérivée d'une fonction d'erreur peut être calculée efficacement dans les MLP. En d'autres termes, elle est basée sur la règle d'apprentissage de correction d'erreur [2] pour minimiser les erreurs et ajuster les paramètres du réseau.

Pour une description plus détaillée de l'algorithme. Laisser un vecteur d'entraînement comme x = [x_1,x_2,…,x_m] et sa sortie cible est t = [t_1,t_2,…,t_p] Normalement, l'algorithme BP peut être divisé en deux passages, passer en avant et passer en arrière [2, 29 , 4l] .

I.5.3. Réseaux de neurones récurrents:

Nous avons introduit MLPs auparavant dont les connexions ne forment pas de cycles. Si les connexions cycliques sont autorisées, un RNN peut être acquis, ce qui est une classe d'ANN et contient au moins une boucle de retour. À la lumière de cela, il est connu que le type le plus simple de RNN est un MLP plus des boucles de retour supplémentaires. En général, un MLP ne peut que cartographier d'entrée en sortie, alors qu'un RNN peut en principe mapper de l'historique complet de l'entrée précédente à chaque sortie. Le point essentiel est que les connexions récurrentes permettent à une "mémoire" d'entrée précédente de persister dans l'état interne du réseau et ainsi d'influencer la sortie du réseau [44]. La capacité d'utiliser la mémoire interne pour traiter des séquences d'entrée arbitraires le rend puissant sur certaines tâches telles que la reconnaissance l'écriture manuscrite [45] et de nombreux autres problèmes séquentiels.

Les RNN sont des outils puissants qui permettent aux réseaux de neurones de gérer des données de longueur arbitraires. Ils peuvent apprendre de nombreux comportements , processus de traitement de séquences , programmes d'algorithmes qui ne sont pas reconnus par les méthodes traditionnelles d'apprentissage par machine [46]. Dans la Figure I.6, nous présentons un exemple de RNN avec une seule couche cachée, dont les commentaires se connectent aux neurones cachés et aux neurones de sortie. En outre, il existe un ensemble d'éléments de retard de l'unité notés (z^{-1}) utilisés dans les boucles de retour [2]. En d'autres termes, il utilise des commentaires au temps (t) comme entrée à l'instant $\mathbf{t_1}$, où les formes ont dirigé les cycles entre les neurones de la RNN.

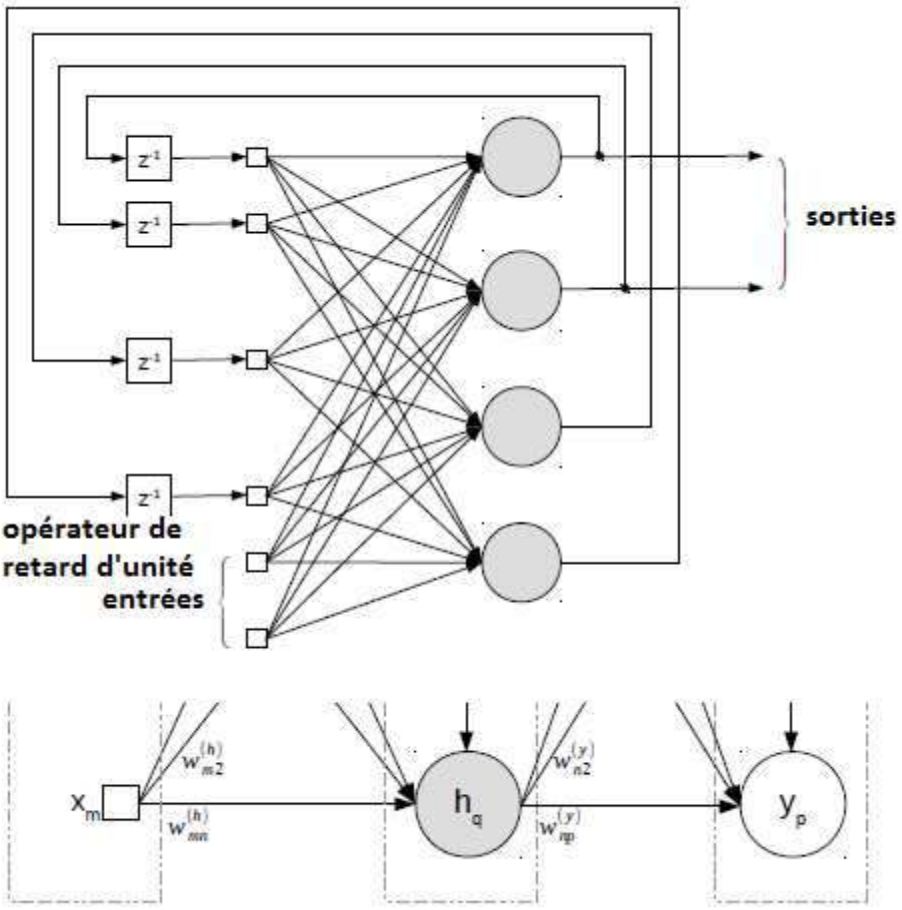


Fig I.6 : un exemple de RNN avec une couche cachée

Avec les réseaux feedforward, de nombreuses variétés de RNN ont été proposées, telles que les réseaux Elman et les réseaux Jordan [44]. Les RNN sont des systèmes dynamiques avec des représentations temporelles de l'état [16] avec plusieurs méthodes de formation supervisée, comme une rétro-propagation à travers le temps (BPTT), l'apprentissage récurrent en temps réel (RTRL) et le filtrage étendu de Kalman (EKF). En plus, les algorithmes BPTT et RTRL sont basés sur une descente en gradient pour minimiser l'erreur, tandis que EKF est plus efficace sur l'utilisation d'informations d'ordre supérieur. EKF peut converger plus rapidement que les algorithmes BPTT et RTRL, mais nécessite une complexité computationnelle plus élevée [47].

I.6. Réseau neuronal pronfond :

Les antécédents sur un réseau «plus profond» sont décrits dans la littérature rappelant les réseaux de neurones. La profondeur est définie dans le cas des réseaux neuronaux feedforward comme plusieurs couches non linéaires entre l'entrée et la sortie [48]. La différence générale entre réseaux neuronaux et DNN est illustrée à la figure I.7.

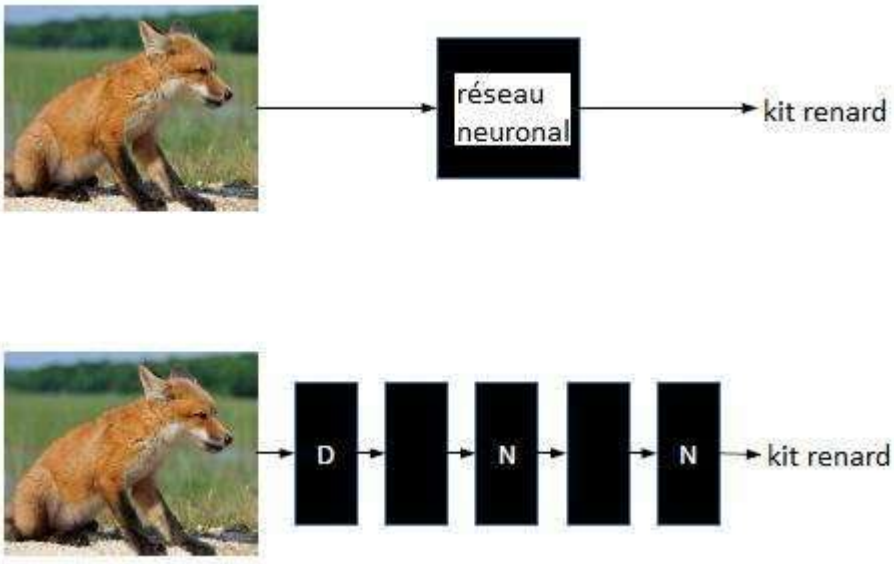


Fig I.7 :Comparaison entre les réseaux de neurones peu profonds et profonds.

Une architecture profonde est celle dans laquelle il existe plusieurs niveaux de représentation , avec des niveaux plus élevés basés sur des niveaux inférieurs (le plus bas étant l'entrée brute d'origine) et des niveaux plus élevés représentant des concepts plus abstraits définis en termes de moins abstraits des niveaux inférieurs [49].

I.6.1. Réseau neuronal convolutif :

Un réseau neuronal convolutif (CNN) est un type de réseau neuronal nourricier utilisant la convolution au lieu de la multiplication générale, inspirée par le traitement biologique et premièrement introduit par Kunihiko Fukushima en 1980 [50]. Un CNN se compose d'une ou plusieurs couches de convergence puis suivie d'une ou plusieurs couches complètement connectées comme dans une structure MLP standard. Avec

l'augmentation de l'informatique GPU efficace, il est possible de former de plus grands réseaux. Les CNN sont composés de plusieurs couches, et nous présentons les trois principales couches ci-dessous.

a. Couche convolutionnelle :

Dans la couche convolutionnelle, Une petite sous-région de l'image pour un filtre convolutif est appliquée pour calculer la sortie, également appelée la carte des caractéristiques, illustrée à la figure I.8.

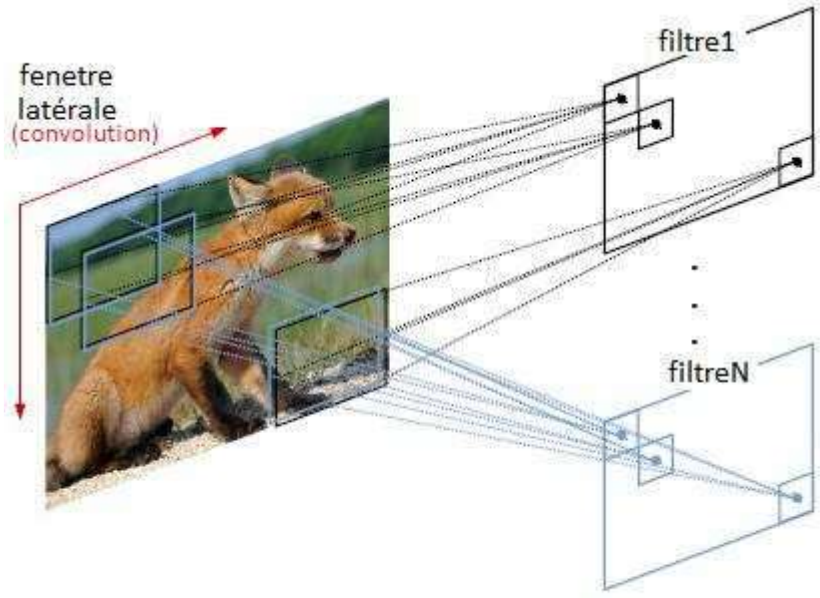


Fig I.8 : couche convolutionnelle

En d'autres termes, chaque neurone prend sa contribution synaptique à partir d'un champ réceptif local dans la couche précédente pour extraire différentes caractéristiques. Par conséquent, la fonction des couches convolutionnelles est d'extraire différentes caractéristiques de l'entrée, et elle peut être représentée comme [51]

$$m_j^{(n)} = \max(0, \sum_{i=1}^{k} m_i^{(n-1)} w_{ij}^{(n)})$$

Où l'exposant n-1 est la couche précédente de n, m_j^n est la j^{eme} carte des caractéristiques de sortie,m_i^{n-1} est la i^{eme}carte des caractéristiques d'entrée et w_{ij}^n est le poids dans le filtre. Normalement, il existe de nombreux filtres convolutifs dans chaque couche, et chaque filtre partage des poids à travers l'image entière. Pour la première couche convolutionnelle, elle acquiert typiquement des caractéristiques de bas niveau, telles que des bords, des lignes et etc. À la lumière de cela, plus le réseau de couches a, les caractéristiques de niveau supérieur sont obtenues. En raison du partage de poids, l'évaluation des activations de ces unités équivaut à une convolution des intensités de pixels d'image avec un filtre comprenant les paramètres de poids [35]

b. Couche de regroupement :

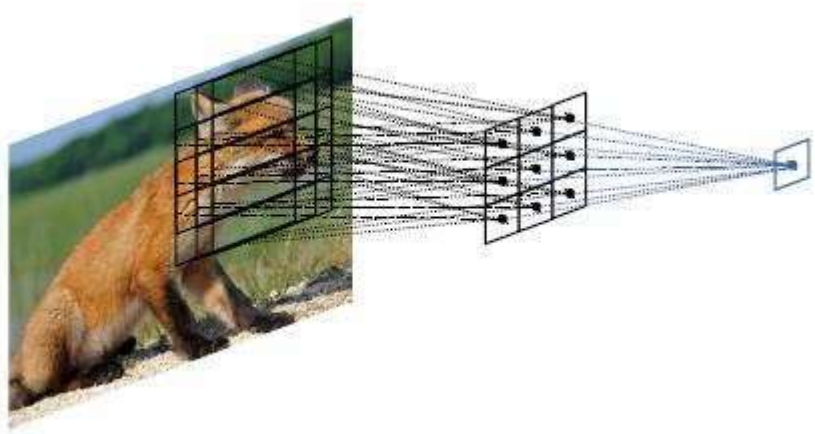


Fig I.9:structure de couche de regroupement.

Il peut y avoir une couche de mise en commun après une couche convolutionnelle, qui est une opération importante pour la classification et la détection des objets. Les couches de mise en commun dans CNN résument la sortie des groupes de neurones voisins dans la même carte de filtre [15]. Il est utilisé pour réduire la variabilité en calculant la valeur moyenne (ou max) d'une caractéristique particulière sur une région de l'image, visualisée dans la Figure I.9.

Ces statistiques récapitulatives ont une dimension beaucoup plus faible (par rapport à l'utilisation de toutes les fonctionnalités extraites) et peuvent également améliorer les résultats (moins sur-ajustement) [52]. L'opération d'agrégation est appelée mise en commun moyen ou groupage maximal, qui dépend de l'opération de mise en commun appliquée. En regroupant les réponses du filtre à différentes localisations, nous obtenons une robustesse à l'emplacement géographique exact des fonctionnalités [15].

c. Couche complétement connectée :

En fin de compte, les CNN ont une ou plusieurs couches complètement connectées après plusieurs couches de convolution et de mise en commun. Il prend toutes les sorties dans la couche précédente et se connecte à chaque neurone, illustré à la Figure I.10. Les couches entièrement connectées ne sont plus spatialement situées, donc il ne peut y avoir de couches convolutives après une couche complètement connectée

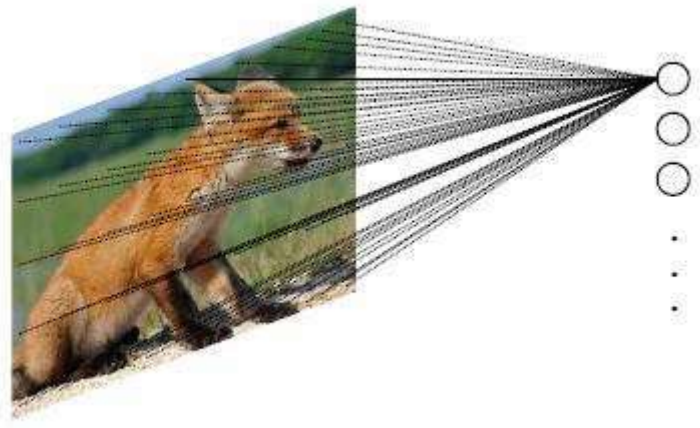


Fig I.10 :Structure d'une couche complètement connectée.

En plus des trois types de couches ci-dessus, les CNN ont également d'autres, comme la couche de décrochage, et la couche de perte [53]. En outre, tout le réseau peut être formé par la minimisation des erreurs En utilisant l'algorithme BP. L'architecture est conçue pour tirer parti de la structure 2D d'une image d'entrée, qui

est obtenue avec les connexions locales et les poids liés, suivis d'une forme de mise en commun qui se traduit par des fonctionnalités invariantes de traduction [52]. Un autre avantage majeur des CNN est l'utilisation de poids partagé en couches convolutionnelles, ce qui signifie que le même filtre est utilisé pour chaque pixel de la couche; Ceci réduit la taille de la mémoire requise et améliore les performances [54].

Sur la base des poids partagés, le nombre de poids dans le réseau est plus petit que si le réseau est entièrement connecté. Par conséquent, le nombre de paramètres indépendants à tirer des données est beaucoup plus petit, en raison du nombre important de contraintes sur les poids [35].

I.6.2. Réseaux de croyances profondes :

Un réseau de croyances profondes (DBN) est un modèle génératif probabiliste introduit par Geoffrey Hinton [55, 56], considéré comme une composition de modules d'apprentissage simples, dont chacun est une machine de Boltzmann restreinte (RBM).

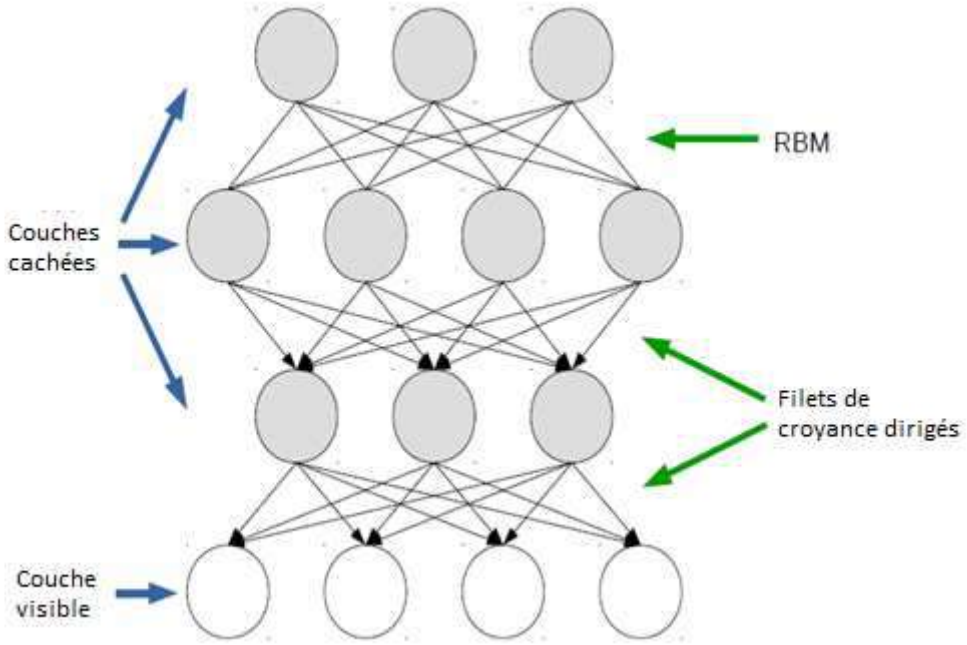


Fig I.11 :Une structure simple de DBN

Il a été utilisé pour générer et reconnaître les images [56] et séquences vidéo [57]. La structure d'un DBN se compose de plusieurs couches, qui peuvent être divisées en couches visibles et couches cachées. Par ailleurs, La couche d'unités visibles représente les données d'entrée, tandis que la couche d'unités cachées apprend à représenter des fonctionnalités qui capturent des corrélations d'ordre supérieur dans les données [58].

La structure simple du DBN est représentée dans la Figure I.11, où la couche inférieure représente les vecteurs de données et chaque unité de couche est un élément de données. Les connexions des deux premières couches ne sont pas dirigées et forment une mémoire associative tandis que d'autres couches inférieures ont des connexions dirigées vers le haut. Il existe une procédure couche par couche pour apprendre les poids, qui est déterminée par la façon dont les variables d'une couche dépendent des variables dans la couche ci-dessus. Après apprentissage, les valeurs des variables latentes dans chaque couche peuvent être déduites d'un seul passage ascendant qui commence avec un vecteur de données observé dans la couche inférieure et utilise les poids générateurs dans le sens inverse [58].

Pour l'efficacité des DBN, l'apprentissage gourmand peut être suivi, ou combiné avec, d'autres procédures d'apprentissage qui affinent tous les poids pour améliorer la performance générative ou critique de l'ensemble du réseau [58]. De plus, nous pouvons ajouter une couche finale pour représenter la sortie souhaitée. Et utiliser l'algorithme BP pour faire respecter l'apprentissage supervisé. Lorsque les réseaux avec de nombreuses couches cachées sont appliqués à des données d'entrée hautement structurées, la rétro-propagation fonctionne beaucoup mieux si les détecteurs de fonctionnalité dans les couches cachées sont initialisés en apprenant un DBN qui modélise la structure dans les données d'entrée [55].

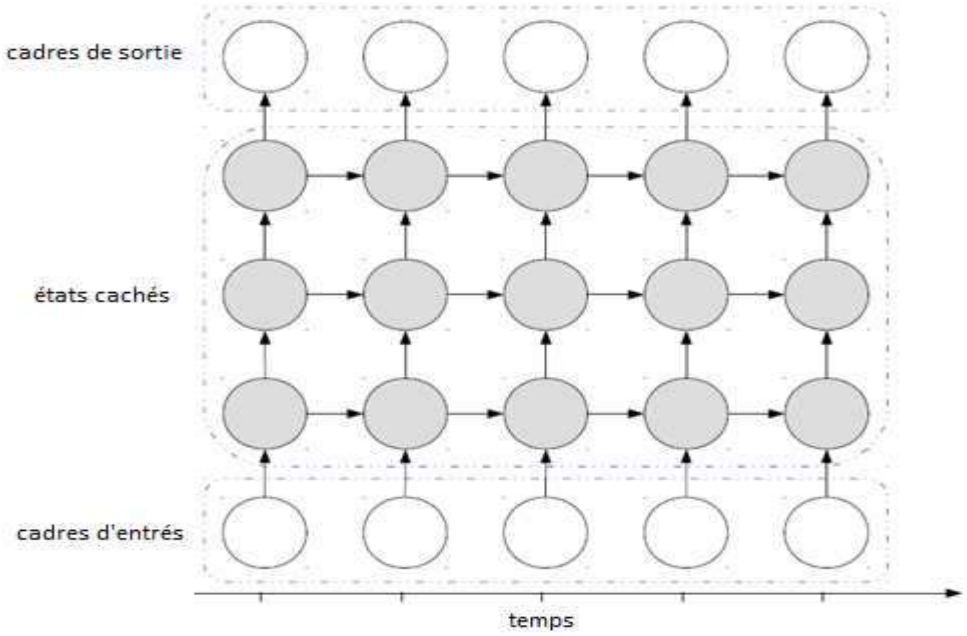
I.6.3. Réseaux neuronaux récurrents:

Un réseau neuronal récurrent (DRNN) est une combinaison de DNN et RNN en principe, et il peut être exploité dans, par exemple, la reconnaissance vocale [59]. Et la

modélisation du langage au niveau des personnages [3]. Il est mentionné que RNNs sont intrinsèquement profond dans le temps, puisque leur état caché est fonction de tous les états cachés précédents [59]. Chaque mise à jour du réseau une nouvelle information parcourt la hiérarchie et le contexte temporel est ajouté a chaque couche [3]. Il existe un DRNN simple visualisé dans la Figure I.12, qui montre les RNNs dans la hiérarchie, et la couche précédente envoie l'état caché à la couche suivante correspondante.

En plus, un DRNN peut également être établi d'une autre manière. Comme l'empilage multiples récurrents des états cachés l'un de l'autre. Il permet potentiellement aux états cachés à chaque niveau de fonctionner à différents horaires [3, 48]. De plus, sur la base du document de Razvan [48], il existe deux modèles pour étendre les RNN aux DRNN. Qui sont RNN de transition profonde et RNN de sortie profonde respectivement.

Par conséquent, les DRNN combinent les multiples niveaux de représentation qui se sont avérés efficaces dans des réseaux profonds avec une utilisation flexible du contexte à long terme qui permet aux RNN [59].



FigI.12 :Illustration schématique d'un DRNN.

I.7. Infrastructure de développement à apprentissage profond :

Des boîtes à outils approfondies sont utilisées pour implémenter l'expérience avec les jeux de données. Nous décrivons ici plusieurs cadres d'apprentissage profond largement utilisés.

I.7.1. Caffe :

Caffe [4] est une plat forme basé sur la bibliothèque C ++ avec des interfaces Python et MATLAB pour un apprentissage en profondeur. Il fournit un paquet complet pour la formation, le test, le réglage fin et le déploiement de modèles. Il existe plusieurs caractéristiques positives dans Caffe. Tout d'abord, il s'agit d'un cadre ouvert modulaire, facile à étendre aux nouveaux formats de données, et aux fonctions de perte.

En plus, les modèles sont définis dans les fichiers de configuration, ce qui entraîne une représentation et une implémentation distinctes. De plus, Caffe fournit également des modèles de référence préformés pour les chercheurs. Caffe définit un net couche par couche dans son propre schéma modèle. Le réseau définit l'intégralité du modèle de bas en haut des données d'entrée à la perte. Le réseau Caffe se composé de blobs, couches et réseaux.

I.7.2 . MatConvNet :

MatConvNet [60] est une boîte à outils MATLAB à conception simple pour la mise en œuvre de CNN. Il comprend des fonctions simples pour calculer les blocs de construction CNN, qui sont faciles à utiliser les fonctions MAT LAB et s'intégrent dans une CNN complète. Les blocs informatiques CNN, les enveloppes CNN, les exemples d'applications et les modèles pré-formés sont les éléments de MatConvNet. En plus, il est flexible de modifier, étendre ou combiner avec de nouveaux réseaux dans le code MAT LAB. La mise en œuvre est efficace pour exécuter les derniers

modèles tels que Krizhevsky [5], et prend en charge le calcul CPU et GPU. Les seules exigences sont de travailler avec le compilateur Matlab et C / C ++. MatConvNet emprunte ses algorithmes de convolution à Caffe, alors qu'il est légèrement plus lent que Caffe.

I.7.3. OverFeat :

OverFeat [61] est un cadre intégré pour l'utilisation des CNN pour la classification, la localisation et la détection. Il s'agit d'un extracteur de fonctionnalités, qui fournit des fonctions puissantes pour la recherche sur la vision par ordinateur. Les codes source C ++ sont fournis pour exécuter le réseau OverFeat pour la reconnaissance des images et l'extraction des fonctionnalités. L'idée principale est d'utiliser une approche multi-échelle et à fenêtre coulissante pour une mise en œuvre efficace dans une CNN. Ceci est censé avoir plusieurs emplacements et échelles pour prédire une boîte de délimitation par classe. Le cadre intégré est le gagnant de la tâche de localisation de ILSVRC 2013 et a obtenu des résultats très compétitifs pour les tâches de détection et de classification.

Conclusion :

En résumé, nous avons montré la tâche générale de la classification des images, des ensembles de données, des méthodes et des boîtes à outils liées à ce mémoire. Tout d'abord, la tâche de la vision par ordinateur, la classification de l'image est expliquée. De plus, CNNs sont la principale méthode utilisée dans le mémoire. Enfin, Caffe est utilisé pour mettre en œuvre et exécuter les expériences.

Chapitre II :

Une formation d'apprentissage progressif dans DNNs

II.1. Introduction :

Ce chapitre commence par expliquer en détail le concept d'apprentissage progressif. Deuxièmement, le prétraitement des jeux de données avec des images à faible résolution et les structures des DNN sont étudiés. Enfin, l'ensemble de données et sa mise en œuvre DNN correspondante dans Caffe pour former le modèle.

II.2. Notre concept d'apprentissage progressif en profondeur:

Avec le nombre croissant de bases de données dans le monde réel, il est nécessaire d'exploiter meilleurs algorithmes d'apprentissage pour gérer plus de données de formation.

Inspiré par le cerveau humain,L'apprentissage progressif est la manière la plus rapide et la plus complète d'apprendre à la disposition des personnes [62]. Par conséquent, c'est aussi une solution possible à adopter pour l'apprentissage par machine. L'apprentissage incrémental est un paradigme d'apprentissage par machine où le processus d'apprentissage se déroule chaque fois qu'un nouvel exemple(s) émerge et ajuste ce qui a été appris selon les exemples précédents [63].

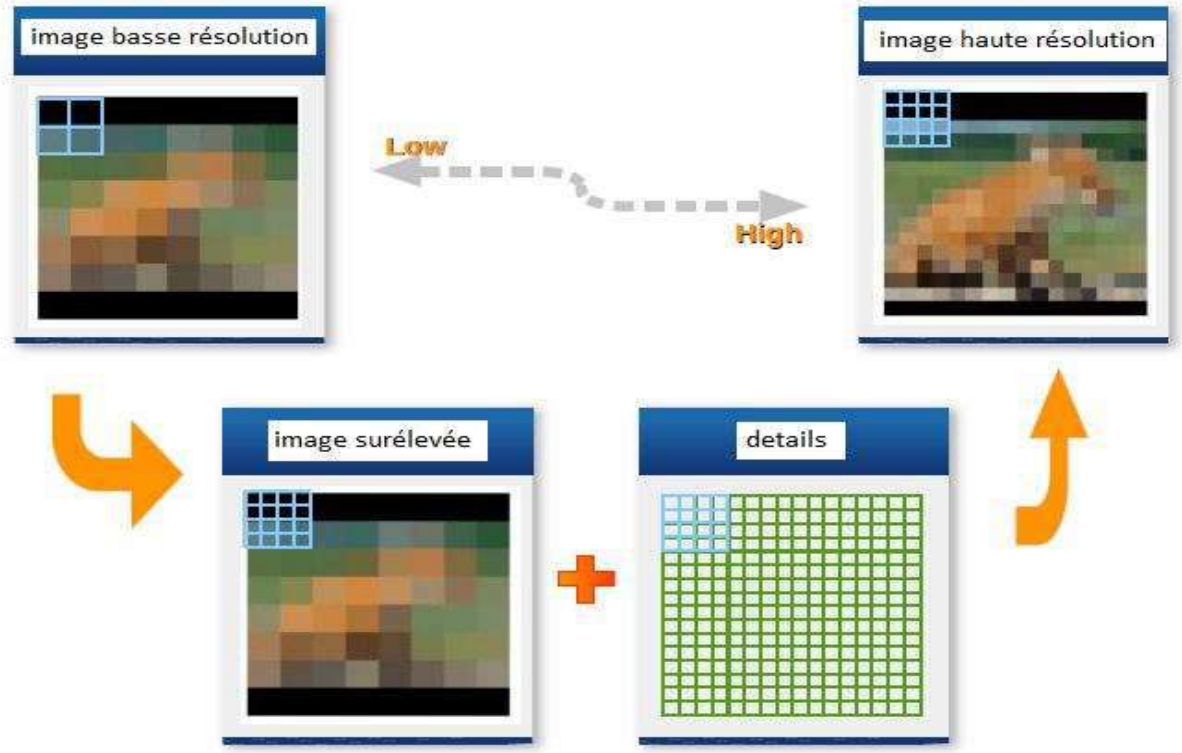


Fig II.1 La relation entre les images à basse et haute résolution

Généralement, le concept de notre idée est d'une manière grossière à fine. Nous utilisons les modèles «grossiers» formés à partir d'images basse résolution comme paramètres initiaux du réseau pour des images à haute résolution pour améliorer l'efficacité de l'ensemble de la procédure de formation.

Par rapport aux images à basse résolution, les images hautes résolution peuvent être divisées en deux parties, illustrées à la figure II.1. La première partie à la même information pour les images à faible résolution et a appris par une petite DNN,la deuxième partie à de nouvelles informations détaillées nécessaires pour être acquises de manière absolue par la fine DNN à partir d'images plus grandes. Si la fine DNN doit les apprendre dès le début, elle gaspille le temps et la mémoire pour former la pièce disponible en basse échelle. Par conséquent, le modèle préformé pour les images de solution peut être utilisé pour transformer l'information en DNN formé en haute résolution d'images.

La procédure complète d'un apprentissage progressif peut être séparée en plusieurs étapes. Tout d'abord, les jeux de données de minuscules images et leurs DNN

correspondants sont construits. Par exemple. Ensemble de données images 8x8 (dénommé I_8) avec ses DNN_8. Deuxièmement, le modèle «grossier» formé peut être obtenu en mettant en œuvre la méthode traditionnelle avec l'ensemble de données d'images à faible résolution et son DNN. Cela renvoie au modèle formé M_8 à I_8 en utilisant DNN_8. De même, la relation entre DNN_8 et DNN_{16} peut être étudiée. Et transformer le modèle de forme M_8 en M_{16}.

Finalement, un nouveau modèle formé M_{16} peut être perfectionné à l'aide des images à résolution plus élevée grâce au réglage fin DNN_{16} avec I_{16}. La procédure susmentionnée est notre processus d'apprentissage incrémental I_8 à I_{16}, illustré à la figure II.2. De même, l'apprentissage progressif peut être mis en œuvre une fois et encore jusqu'à l'achèvement du réglage final des images en taille réelle et d'obtenir la performance finale.

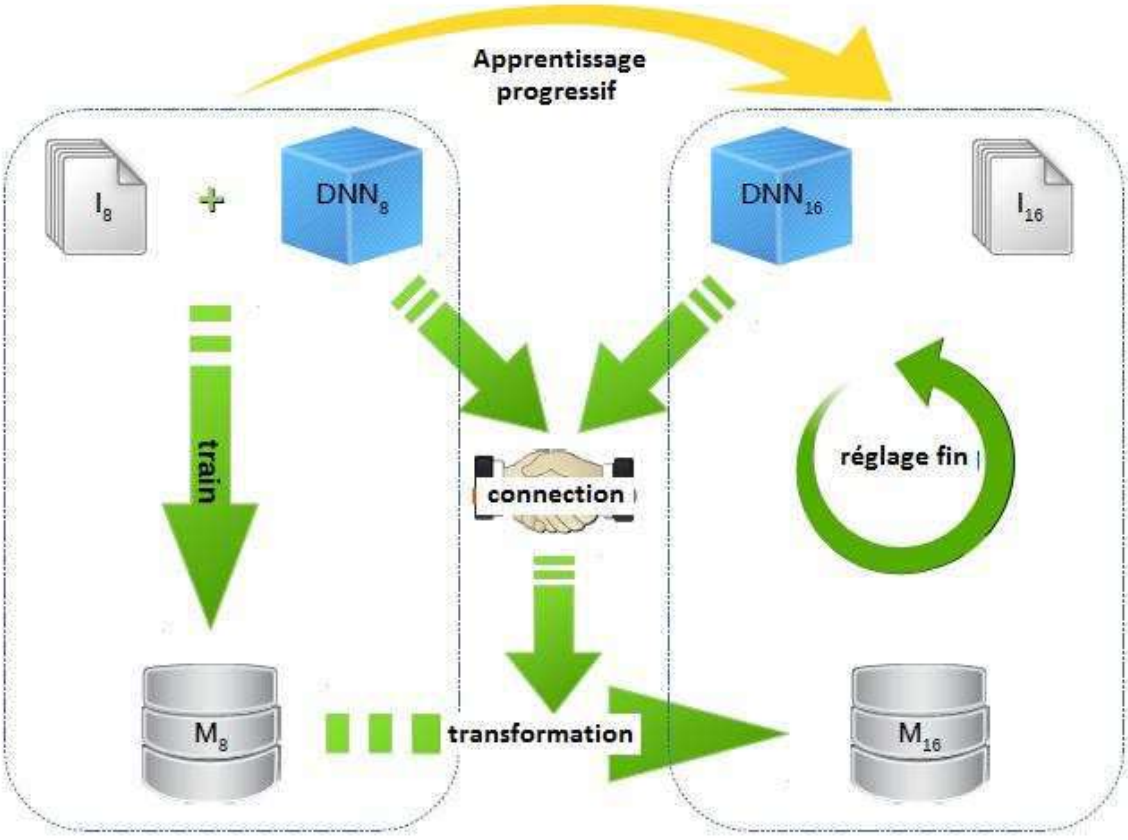


Fig II.2Un processus d'apprentissage progressif des CNNs.

Dans la section suivante, les étapes spécifiques de la méthode sont présentées. L'ensemble de données de classification d'images ILSVRC est utilisé dans les expériences. Pour l'ensemble de données Places205, les mêmes réseaux et méthodes peuvent être exécutés, et nous montrons ses résultats dans le chapitre III.

II.3. Génération de jeux de données ImageNet-Minuscule:

L'ensemble de données de classification d'image (ILSVRC)est un ensemble de données à grande échelle, qui contient 1,2 million d'images pour la formation et 50000 images pour le test. Ces images appartiennent à 1000 classes d'objets et chaque image correspond à une étiquette de classe de vérité. Pour la mise en œuvre de l'idée de l'apprentissage progressif, «les minuscules jeux de données d'images à basse résolution sont créés par downscaling des images en taille réelle et alimentés au DNN mis en œuvre dans Caffe.

II.3.1. Caffe :

C'est un plat forme d'apprentissage profond, crée par BVLC (Berkeley Vision and Learning Center),Il réduit le travail de l'utilisateur en permettant à l'utilisateur de développer des réseaux complexes de neurones profonds, il est basé sur la bibliothèque C ++ avec des interfaces Python et MATLAB pour un apprentissage en profondeur.

Caffe définit un net couche par couche dans son propre schéma modèle. Le réseau définit l'intégralité du modèle de bas en haut des données d'entrée à la perte. Le réseau Caffe se compose de blocs, de couches et de réseaux. Dans la figure II.3, il existe le calcul de la couche et les connexions à gauche, ainsi qu'un exemple de réseau Caffe à droite.

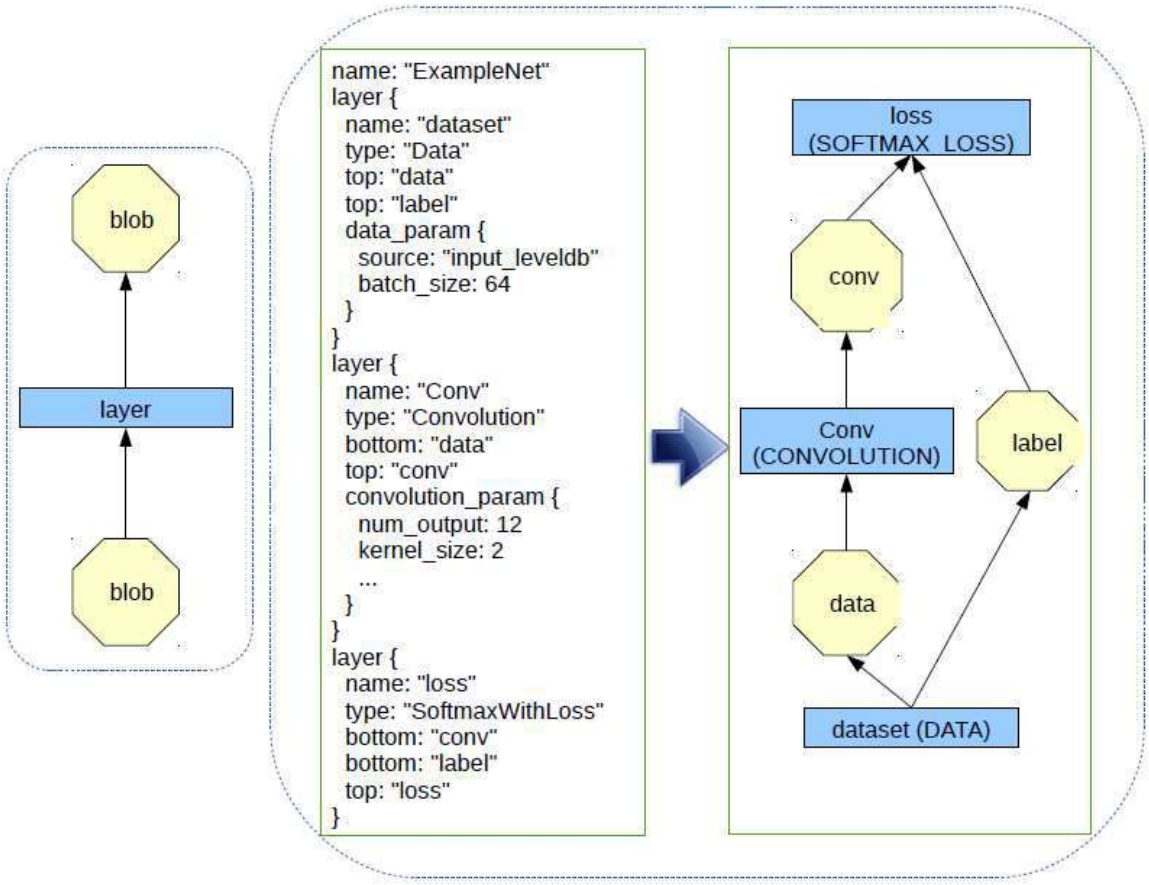


Fig II.3 : Gauche : calcul de couche et connexions;
Droite: un exemple de réseau Caffe

Le réseau Caffe se compose de blobs, Couches et réseaux :

Blob: une matrice de 4 dimension utilisé pour stocker des données et des paramètres, qui peuvent être utilisés pour vérifier les dimensions, lire, transformer et enregistrer les paramètres. L'octogone désigne une blob dans la figure II.3.

Couche: une couche de réseau neuronal en utilisant des blobs en entrée et en sortie, présentées par des rectangles. Pour les réseaux d'exploitation, les couches prennent des responsabilités à la fois en avant et en arrière. Caffe fournit un ensemble complet de couches dans différents types pour concevoir des réseaux dans les expériences.

Réseaux: un ensemble de couches connectées dans un graphique de calcul.

a. Pourquoi Caffe :

On a choisi Caffe pour ces nombreux avantages, on note parmi eux :

- **Expression:** Les modèles + les optimisations sont des schémas en texte clair, pas un code.
- **La vitesse:** Pour des modèles à la fine pointe de la technologie et des données massives.
- **Modularité:** Pour étendre à de nouvelles tâches et paramètres.
- **Ouverture:** Code commun et modèles de référence pour la reproductibilité.
- **Communauté:** Discussion conjointe et développement par licence BSD-2.

Par contre Caffe à une grande consommation de mémoire et faible support RNN.

b. Matériels :

L'installation du programme Caffe et l'exécution nécessite une machine plus fort qu'une simple machine, aussi pour qu'il peut faire ses calcule rapidement :

- Système d'exploitation (64 bits)
- Installation NVIDIA
- Installation (MATLAB/Python)

c. Les commandes de Caffe :

Drop:définit une partie aléatoire de 0, ajustant le reste de la grandeur vectorielle en conséquence.

Relu :Étant donné une valeur d'entrée x, la couche ReLU calcule la sortie comme x si x> 0 et negative_slope * x si x<= 0. Lorsque le paramètre de pente négative n'est pas défini, il équivaut à la fonction ReLU standard de prendre max (x , 0). Il supporte également le calcul en place, ce qui signifie que le bas et le bloc supérieur pourraient être les mêmes pour préserver la consommation de mémoire.

Norm : Normalise l'entrée pour avoir une variance de 0-moyenne et / ou unité (1) dans le lot.

Conv : La couche de convolution fonctionne sur le principe de la convolution, ce qui implique l'application d'une fonction mathématique à une gamme de valeurs. Ceci est la plus couramment utilisé pour extraire des images et d'autres entrées multidimensionnelles.

Data : Les données traversent Caffe comme Blobs. Les couches de données chargent l'entrée et enregistrent la sortie en convertissant vers et depuis Blob vers d'autres formats. Les transformations courantes comme la soustraction moyenne et la mise à l'échelle des fonctions sont effectuées par la configuration de la couche de données. Les nouveaux types d'entrée sont pris en charge en développant une nouvelle couche de données.

II.3.2. Génération des Datasets ImageNet-Tiny :

Tout d'abord, la conversion d'images est fournie par l'outil ImageMagick8. La commande « convert » peut être exécutée comme

```
$ convert<INPUT>.JPEG -resize(weight)x(height)<OUTPUT>.bmp
```

Qui redimensionne les images à la taille poids*hauteur . Les paramètres de poids et de hauteur sont réglés sur les valeurs souhaitées pour obtenir les images à basse résolution. En plus l'option de redimensionnement, il existe également d'autres options utiles, telles que l'extension, l'espace de couleurs et etc. Les images 8x8, 16x16 et 32x32 sont générées pour les expériences, et l'exemple est illustré à la figure II.3.

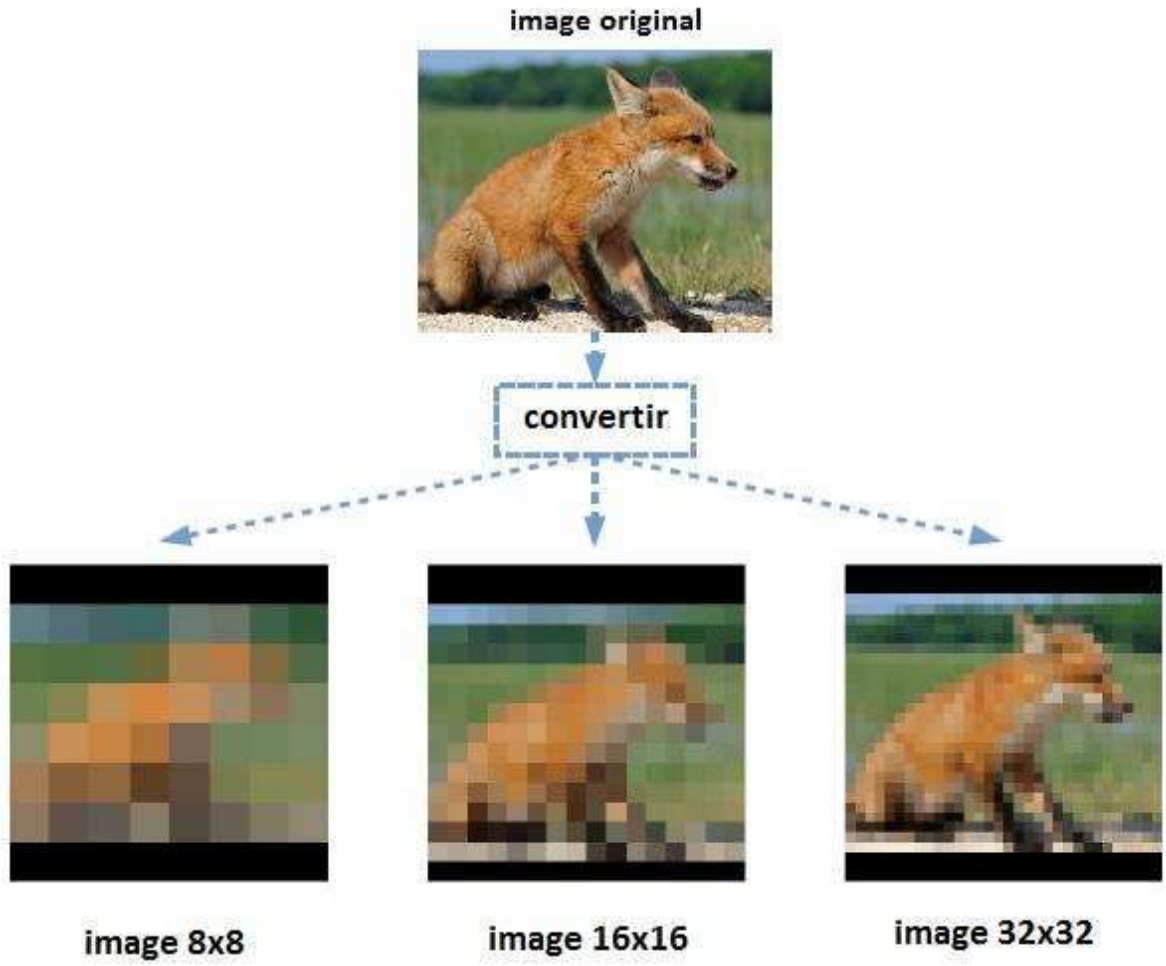


Fig II.4 : Un exemple de l'image pleine grandeur de classe du renard utilise pour générer de minuscules images 8*8 , 16*16 et 32*32

Ensuite, l'ensemble des images est nécessaire pour convertir en une base de données lmdb / leveldb, qui est un moyen efficace d'accéder aux réseaux Caffe. Caffe fournit un programme de conversion, exécuté comme [4] :

```
$ convert_imageset [FLAGS] ROOTFOLDER/ LISTFILE DB_NAME
```

En fin de compte, les jeux de données compatibles Caffe d'images à faible résolution sont obtenus, notés comme I_8, I_{16} et I_{32}, qui sont utilisés dans les expériences suivantes.

II.4. Redimensionnement des réseaux profonds pour la formation incrémentielle :

Notre banc de test est l'architecture d'apprentissage en profondeur de Caffe avec une configuration de réseau équivalente à celle proposée par Krizhevsky et al [5]. La procédure de mise à l'échelle et les réseaux pour chaque taille d'images sont décrits dans le tableau II.1.

	Paramètres de réseau				
	L1(conv)	L2 (conv)	L3 (conv)	L6 (full)	L8 (full)
Orig [5]	96(11)P3/2	256(5)P3/2	384(3)P-/-	4096	1000
2^5-éscalader (8x8)	3(0.3)P0/0	8(1)P1/1	12(3)P-/-	128	1000
2^4-éscalader (16x16)	6(0.7)P1/0	16(3)P2/1	24(3)P-/-	256	1000
2^3- éscalader(32x32)	12(1.3)P2/1	32(4)P3/2	485(3)P-/-	512	1000
Net-1	3(2)P1/1	-	-	128	1000
Net-2	6(2)P1/1	-	-	256	1000
Net-3	12(2)P1/1	-	-	512	1000
Net-4	6(2)P2/1	16(3)p3/2	-	256	1000

Tab II.1 : La procédure de mise à l'échelle des DNN pour l'apprentissage progressif

Dans le tableau, la structure du réseau correspond à Krizhevsky et al.[5] Et L# (L1-L8) fait référence à la couche correspondante. Les couches n'ont pas été éteintes dans nos expériences, leur étendue spatiale étant inférieure à 1 pixel. en conséquence, l'original a été mis à l'échelle (les nombres $256/2^5$ =8) sont donnés pour la comparaison avec les structures sélectionnées. Les nombres entre parenthèses pour les couches convolutionnelles correspondent à la taille des filtres à (N x N) Les tailles de filtre 1 n'ont pas été testées puisqu'elles ne sont pas des filtres Le pooling a été désactivé (NP: No Pooling) si sa taille spatiale serait inférieure à un pixel, le paramètre P1/1pooling correspond à la non-mutualisation.

Les résultats de l'entraînement sont présentés à la Figure II.4. Nous choisissons le net-3 comme l'original Réseau pour mettre en œuvre nos expériences suivantes. Parce qu'il a le plus de précision. L'architecture de net-3 contient une couche

convolutionnelle (conv1) et deux couches complètement connectées fct6 et fct8. Montré à la Figure II.5. Ce réseau est doté d'images à basse résolution et fournit des images haute résolution aux réseaux.

Avec la résolution croissante des images, de nouveaux détails doivent être appris par DNN_S. À la lumière de cela, les DNN_S pour les images hautes résolution nécessitent des filtres de plus grande taille ou plus de filtres que le réseau d'images à faible résolution. Ainsi, il existe deux types d'étapes de redimensionnement réseau conçues pour les expériences, les réseaux avec des filtres à l'échelle et les réseaux avec des filtres ajoutés. Le nombre de filtres et la taille des filtres dans la couche convolutionnelle sont indiqués dans le tableau II.2 qui sont facilement utilisés dans DNN_8, DNN_{16} et DNN_{32}.

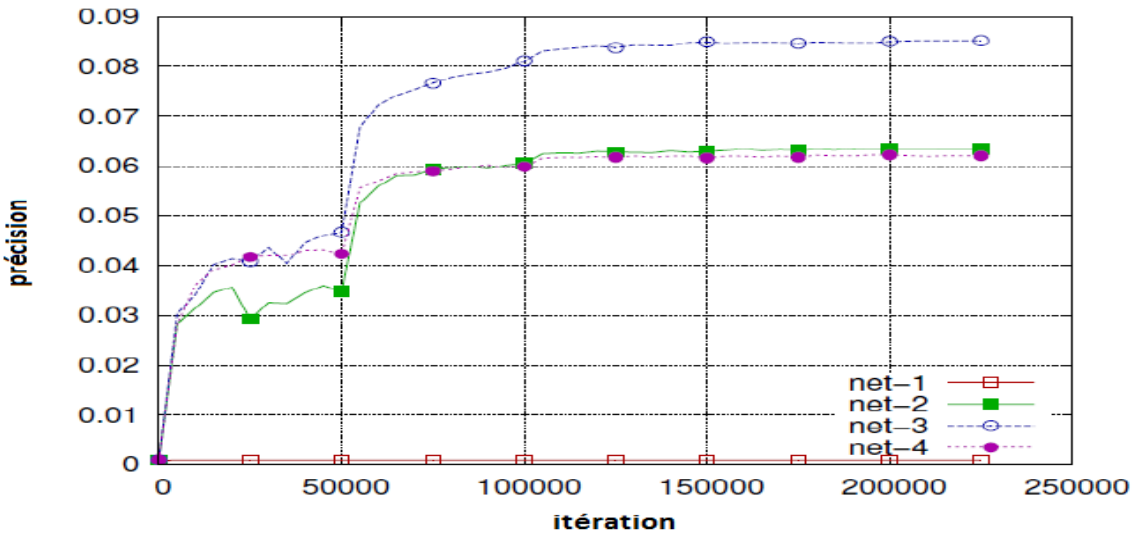


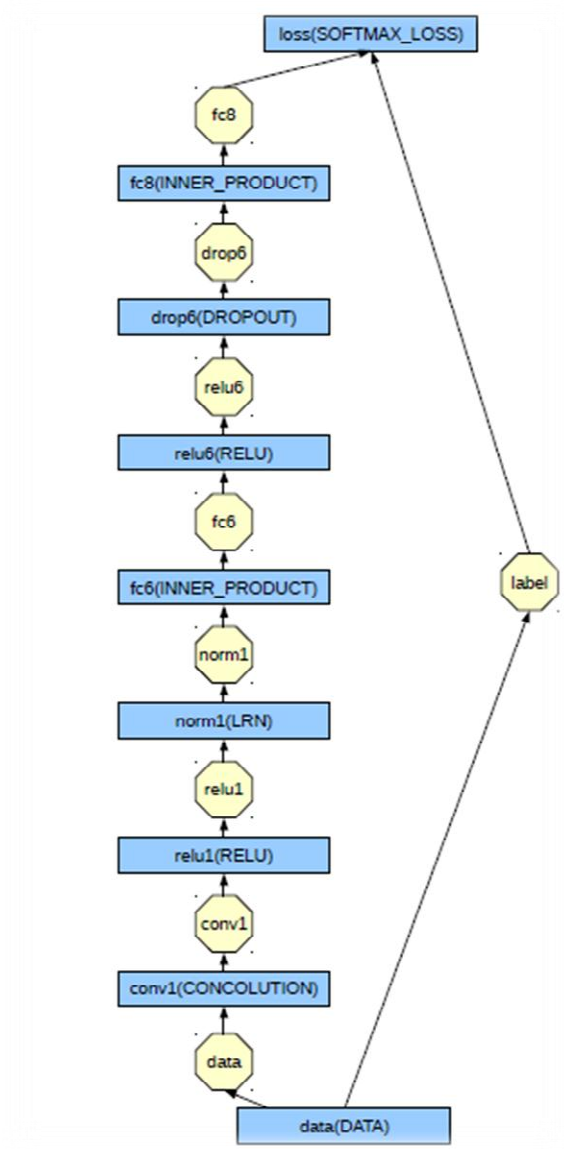
Fig II.4 :Résultats de la formation I_8 avec net-1, net-2, net-3 et net-4 pour l'ensemble de données ImageNet

	Nombre de filtres (Taille)		
	DNN_8	DNN_{16}	DNN_{32}
Escaladé	12(2)	12(4)	12(8)
Ajoutée	12(2)	16(4)	24(8)

Tab II.2 :Résumé des paramètres en couche convolutionnelle dans DNN_8, DNN_{16} et DNN_{32}.

- Les filtres à l'échelle: la taille des filtres convolutionnels est étendue à une résolution plus élevée, tandis que le nombre de filtres est maintenu de la même manière (12). Au cours des expériences, le filtre de 2x2 (désigné par K_2) est appliqué sur I_8, s'étend à K_4 pour I_{16} et s'étend à K_8 pour I_{32}
- Les filtres ajoutés: outre la taille des filtres, le nombre de filtres est passé de 12 en I_8, 16 en I_{16} et 24 en I_{32}.

Pour définir ces paramètres de DNN_S, Caffe fournit un fichier de configuration simple pour les définir,



FigII.5 : Architecture Net-3

où les réseaux sont définis par des schémas de texte brut [4]. Un exemple de fichiers de configuration DNN_8 est donné à l'Annexe A pour les expériences.

II.5. Formation des réseaux profond:

Basé sur la génération d'ensembles de données ImageNet et de redimensionnement de réseaux profonds, les ensembles de données et les réseaux ont été préparés pour les expériences. La dernière étape consiste à former le modèle «grossier» d'images basses résolution comme valeurs initiales pour le prochain réseau «fin». Les expériences commencent à partir du jeu de données des images de résolution la plus basse est et correspond Réseaux DNN_8 pour obtenir le modèle M_8. Avant le début de la phase de formation, un solveur est défini à l'Annexe B. De plus, le modèle exige de soustraire la moyenne d'image de chaque image [4, 5]. Qui peut être calculé par le programme fourni dans Caffe. Après avoir complété les fichiers prédéfinis, le réseau peut être formé pour obtenir le modèle.

Caffe fournit trois interfaces pour les utilisateurs [4], qui sont la ligne de commande, python et MATLAB. L'interface de ligne de commande a été choisie pour nos expériences. Pour former un modèle, la commande d'entrainement de Caffe est utilisée comme :

```
$ caffe train -solver path/to/solver.prototxt
```

Lorsque le code est exécuté, il fournit beaucoup de détails brièvement démontrés dans Figure II.6. Montrant les détails de ces connexions entre les couches et la forme des blobs. Par exemple. "Convl ->data" est la connexion entre la couche de données et la couche convulution, et (256, 12.7, 7) est la forme de convl blob.

```
I0325 09:47:19.580461 31004 net.cpp:67] Creating Layer conv1
I0325 09:47:19.580466 31004 net.cpp:394] conv1 <- data
I0325 09:47:19.580478 31004 net.cpp:356] conv1 -> conv1
I0325 09:47:19.580488 31004 net.cpp:96] Setting up conv1
I0325 09:47:19.580858 31004 net.cpp:103] Top shape: 256 12 7 7 (150528)
```

Fig II.6 : Les détailles Caffe lors de l'initialisation du réseau

Après l'initialisation, la phase d'entraînement commence à partir de l'itération 0, comme le montre la Figure II.7.

```
I0325 09:47:19.657320 31004 net.cpp:219] Network initialization done.
I0325 09:47:19.657325 31004 net.cpp:220] Memory required for data: 1299008
I0325 09:47:19.657382 31004 solver.cpp:41] Solver scaffolding done.
I0325 09:47:19.657399 31004 solver.cpp:160] Solving CaffeNet
I0325 09:47:19.657407 31004 solver.cpp:161] Learning Rate Policy: step
I0325 09:47:19.657426 31004 solver.cpp:264] Iteration 0, Testing net (#0)
I0325 09:47:22.079560 31004 solver.cpp:315]     Test net output #0: accuracy = 0.00112
I0325 09:47:22.079596 31004 solver.cpp:315]     Test net output #1: loss = 6.93571 (* 1 = 6.93571 loss)
I0325 09:47:22.098292 31004 solver.cpp:209] Iteration 0, loss = 6.96574
I0325 09:47:22.098330 31004 solver.cpp:224]     Train net output #0: loss = 6.96574 (* 1 = 6.96574 loss)
I0325 09:47:22.098342 31004 solver.cpp:445] Iteration 0, lr = 0.01
I0325 09:47:22.402977 31004 solver.cpp:209] Iteration 20, loss = 6.96712
I0325 09:47:22.403008 31004 solver.cpp:224]     Train net output #0: loss = 6.96712 (* 1 = 6.96712 loss)
I0325 09:47:22.403017 31004 solver.cpp:445] Iteration 20, lr = 0.01
```

Fig II.7: Les détailles Caffe dans la sortie pendant l'entraînement

Pour chaque itération de la formation, lr est le taux d'apprentissage de cette itération, et la perte est la formation fonction définie dans le solveur. Pour la sortie de la phase de test, le score 0 est la précision, et le score 1 est la fonction de perte de test, illustrée à la figure II.8.

```
I0325 09:47:37.036432 31004 solver.cpp:209] Iteration 980, loss = 6.33484
I0325 09:47:37.036463 31004 solver.cpp:224]     Train net output #0: loss = 6.33484 (* 1 = 6.33484 loss)
I0325 09:47:37.036469 31004 solver.cpp:445] Iteration 980, lr = 0.01
I0325 09:47:37.326050 31004 solver.cpp:264] Iteration 1000, Testing net (#0)
I0325 09:47:39.895401 31004 solver.cpp:315]     Test net output #0: accuracy = 0.0214201
I0325 09:47:39.895429 31004 solver.cpp:315]     Test net output #1: loss = 6.28455 (* 1 = 6.28455 loss)
I0325 09:47:39.904927 31004 solver.cpp:209] Iteration 1000, loss = 6.42154
I0325 09:47:39.904945 31004 solver.cpp:224]     Train net output #0: loss = 6.42154 (* 1 = 6.42154 loss)
I0325 09:47:39.904953 31004 solver.cpp:445] Iteration 1000, lr = 0.01
I0325 09:47:40.210124 31004 solver.cpp:209] Iteration 1020, loss = 6.35022
I0325 09:47:40.210155 31004 solver.cpp:224]     Train net output #0: loss = 6.35022 (* 1 = 6.35022 loss)
I0325 09:47:40.210181 31004 solver.cpp:445] Iteration 1020, lr = 0.01
```

Fig II.8 :Les détailles Caffe pendant l'entraînement et les tests.

Après la formation, le résultat final peut être obtenu en tant que «Test de sortie nette n ° 0: précision = 0: 0835801», et le modèle sera sauvegardé, comme "Snapshottingto ./TEMPWORK/caffe_ImageNet_tiny-8x8-sRGB-net-3_iter_120000.caffemodel ", visualisé à la figure II.9.

```
I0325 10:23:03.249635 31004 solver.cpp:209] Iteration 119980, loss = 5.40909
I0325 10:23:03.249668 31004 solver.cpp:224]     Train net output #0: loss = 5.40909 (* 1 = 5.40909 loss)
I0325 10:23:03.249676 31004 solver.cpp:445] Iteration 119980, lr = 0.0001
I0325 10:23:03.548398 31004 solver.cpp:334] Snapshotting to ./TEMPWORK/caffe_ImageNet-tiny-8x8-sRGB-net-3_iter_120000.caffemodel
I0325 10:23:03.554522 31004 solver.cpp:342] Snapshotting solver state to ./TEMPWORK/caffe_ImageNet-tiny-8x8-sRGB-
net-3_iter_120000.solverstate
I0325 10:23:03.566525 31004 solver.cpp:246] Iteration 120000, loss = 5.34311
I0325 10:23:03.566553 31004 solver.cpp:264] Iteration 120000, Testing net (#0)
I0325 10:23:06.134392 31004 solver.cpp:315]     Test net output #0: accuracy = 0.0835801
I0325 10:23:06.134421 31004 solver.cpp:315]     Test net output #1: loss = 5.41878 (* 1 = 5.41878 loss)
I0325 10:23:06.134428 31004 solver.cpp:251] Optimization Done.
I0325 10:23:06.134433 31004 caffe.cpp:121] Optimization Done.
```

Fig II.9 : Les détailles Caffe pendant l'achèvement de l'entraînement.

Cette procédure est répétée pour les réseaux tout DNNx, mais la transformation des paramètres de l'échelle inférieure à supérieure est expliquée dans la section suivante.

Pour haut niveau DNN, nous devons transférer des paramètres DNN grossiers à DNN fin avant l'entraînement.

Par la suite, la transformation des paramètres pour l'apprentissage progressif est proposée et analysée.

Les deux premières parties couvrent la transformation des paramètres dans différentes couches de réseaux haut de gamme décrits au debut de chapitre. En plus, plusieurs méthodes pour initialiser les paramètres de nouveaux filtres ajoutés dans DNN sont présentées. Enfin, la fonction de réglage précis fournie par Caffe peut être utilisée pour vérifier de nouveaux modèles avec des paramètres transformés.

II.6. Les réseaux avec filtres à l'échelle :

Les réseaux à filtres à l'échelle doivent étendre la taille des filtres. Selon le tableau II.2, k_2 est appliqué aux DNN_8, K16 aux DNN16 et K_8aux DNN_{32} dans les expériences. La mise à niveau DNN n'affecte pas seulement les filtres. Mais augmente également la dimension de la sortie. Qui est utilisé comme entrée pour la prochaine couche. Les paramètres (poids de sortie) doivent être initialisés dans les deux filtres et les connexions de couche suivantes. Dans cette section. La transformation des paramètres en couche convolutionnelle et sa prochaine couche. Couche de produit interne ; sont discutés respectivement.

II.6.1. Couches convolutionnelle:

Conceptuellement, les cartes des caractéristiques de sortie peuvent être calculées par une convolution à partir des images d'entrée avec un filtre linéaire ajoutant un biais et une fonction non linéaire. Si nous supposons la k^{eme} carte de sortie en y^k, le filtre convolutionnelle est définie par les poids w^k et le biais b^k, ce qui peut être écrit comme :

$$y_{ij}^k = \varphi((\mathbf{w^k} * \mathrm{x})_{ij} - b^k) \rightarrow \quad (2.1)$$

Où φ est la fonction non linéaire, et y_{ij}^{k} est la valeur (i, j) de Y^{k} pour simplifier le problème, seul la k^{eme} carte de sortie est discutée pour exploiter une solution. L'étiquette k est omise. Et une étiquette numérique est ajoutée pour distinguer la taille différente des images.

Tout d'abord, dans la couche convolutionnelle des DNN_8 , une matrice 8 x 8 $X^{(8)}$ est l'image d'entrée. Une matrice $Y^{(8)}$est la carte des caractéristiques de sortie. Une matrice de 2 x 2 W^8 et $b^{(8)}$ sont les poids et les biais du filtre.

Basé sur l'équation (2.1), la sortie peut être écrite comme:

$$\begin{cases} y_{11}^{8} = \varphi((\mathbf{w^8} * \mathrm{x})_{11} - b^{8}) \\ y_{12}^{8} = \varphi((\mathbf{w^8} * \mathrm{x})_{12} - b^{8}) \\ \cdots \end{cases} \qquad (2.2)$$

En outre, la même méthode peut être appliquée pour obtenir la sortie du couche convolutionnelle de DNN_{16} comme:

$$y_{11}^{16} = \varphi((\mathbf{w^{16}} * \mathrm{x})_{11} - b^{16})$$

$$y_{12}^{16} = \varphi((\mathbf{w^{16}} * \mathrm{x})_{12} - b^{16}) \qquad (2.3)$$

$$y_{13}^{16} = \varphi((\mathbf{w^{16}} * \mathrm{x})_{13} - b^{16})$$

…

Dans le modèle préformé, les paramètres de W^8 et b^8 peuvent être extraits pour utiliser provisoirement dans W^{16} et b^{16},et la sortie de DNN^{16} doit être maintenue identique. Notez que, dans nos expériences, la dimension de W^{16} est deux fois supérieure à W^8 , et le biais est le meme.

Pour le biais, la même valeur peut être utilisée dans DNN_{16}**.** La raison est qu'ils ont le même effet sur la sortie, ce qui est soustraire du résultat de la convolution. Sur la base des équations (2.2) et (2.3), le biais peut être simplement copie en tant que :

$$b^{16} = b^{8} \qquad (2.4)$$

Néanmoins,pour transformer les poids, W^8et W^{16} qui ont des dimensions différentes, la façon d'étendre les poids devient la principales préoccupation.

Ici, les exemples d'images sont introduits pour analyser le problème. I_{16} des images d'échantillons sont converties à partir de I_8 en utilisant la méthode de réplication de pixels, qui prolonge chaque pixel en quatre pixels avec la même valeur Dénommé I_{16s}, illustrée à la figure (2.1). Ainsi, les relations entre I_8 et I_{16s} sont :

$$x_{11}^{(16x)} = x_{12}^{(16x)} = x_{21}^{(16x)} = x_{22}^{(16x)} = x_{11}^{(8)}$$

$$x_{13}^{(16x)} = x_{14}^{(16x)} = x_{23}^{(16x)} = x_{24}^{(16x)} = x_{12}^{(8)}$$

$$\dots \qquad (2.5)$$

$$x_{31}^{(16x)} = x_{32}^{(16x)} = x_{41}^{(16x)} = x_{42}^{(16x)} = x_{21}^{(8)}$$

$$\dots$$

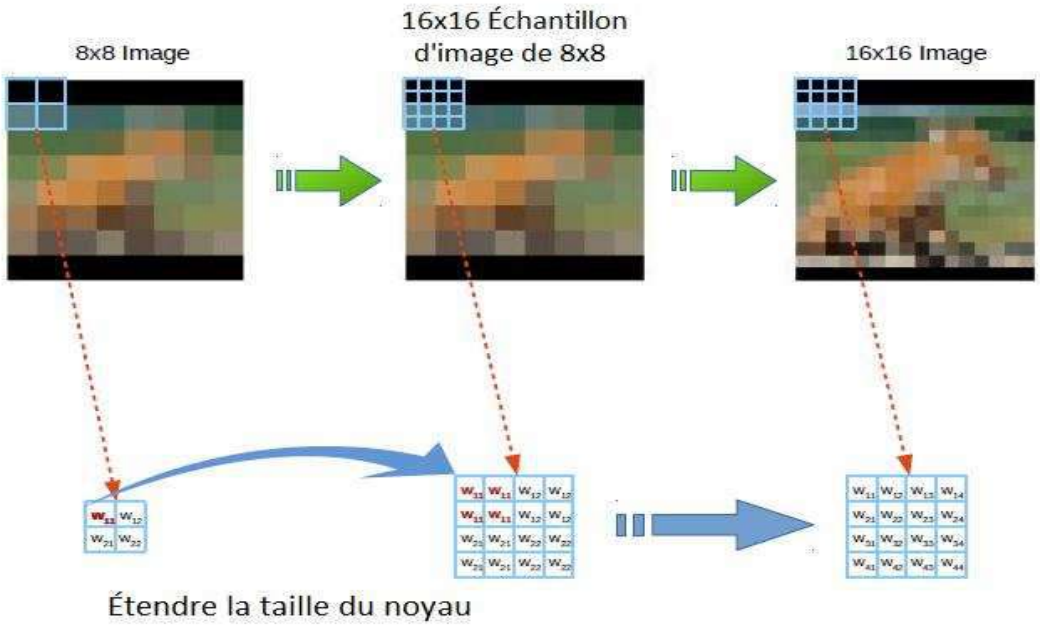


Fig II.10 : Paramètres de transformation en couche convolutionnelle de DNN_s

Basé sur l'équation (2.1), La sortie de I_{16s} peut être écrite de la même manière avec Y^{16}comme :

$$y_{11}^{(16x)} = \varphi((\mathrm{W}^{(\mathbf{16x})} * \mathrm{x})_{11} - b^{(16x)})$$

$$y_{12}^{(16x)} = \varphi((\mathrm{W}^{(\mathbf{16x})} * \mathrm{x})_{12} - b^{(16x)})$$

$$y_{13}^{(16x)} = \varphi((\mathrm{W}^{(\mathbf{16x})} * \mathrm{x})_{13} - b^{(16x)}) \qquad (2.6)$$

…

Si la même résultat a l'intention d'être gagné entre Y^8 et Y^{16s}, Les poids devraient également être copié et répété comme :

$$w_{11}^{(16x)} = w_{12}^{(16x)} = w_{21}^{(16x)} = w_{22}^{(16x)} = \frac{w_{11}^{(8)}}{4}$$

$$w_{13}^{(16x)} = w_{14}^{(16x)} = w_{23}^{(16x)} = w_{24}^{(16x)} = \frac{w_{12}^{(8)}}{4} \qquad (2.7)$$

…

$$w_{31}^{(16x)} = w_{32}^{(16x)} = w_{41}^{(16x)} = w_{42}^{(16x)} = \frac{w_{21}^{(8)}}{4}$$

…

La raison pour laquelle les valeurs des poids dans $W^{(16x)}$sont 1/4 à $W^{(8)}$est qu'un pixel dans I_8 est répété 4 fois en I_{16x}. Par conséquent, les poids doivent être divisés par 4 pour conserver la même sortie entre Y^8 et Y^{16}. En outre, lorsque les I_{16s} sont utilisés pour convolution avec le filtre, les mêmes résultats peuvent être notés avec Y^8 dans toutes les autres valeurs, ce qui signifie,

$$y_{11}^{(8)} = y_{11}^{(16x)}$$

$$y_{12}^{(8)} = y_{13}^{(16x)}$$

… (2.8)

$$y_{21}^{(8)} = y_{31}^{(16x)}$$

…

De plus, Les différentes valeurs entre elles peuvent être ignorées dans la prochaine couche.

Enfin, la solution des paramètres de transformation dans la couche convolutionnelle est proposée. Il est réalisé en étendant et en copiant de $W^{(8)}$ à $W^{(16)}$, et de $b^{(8)}$ à $b^{(16)}$Bien qu'il y ait une certaine différence entre I_{16} et I_{16s}, la performance peut être réduite un peu, même si elle n'a pas trop d'impact. La différence entre eux est que les nouveaux détails doivent être appris par DNN_{16}. Cela garde les anciennes informations dans I_8 et DNN_{16} continue à apprendre de I_{16}. L'ensemble du processus d'une transformation en couche convolutionnelle est illustré à la figure II.10.

II.6.2. Couche de produit interne :

Généralement, la couche de produit interne est également appelée couche entièrement connectée. Il traite l'entrée en tant que vecteur simple et prend tous les neurones pour produire des résultats. Dans les expériences. L'entrée de cette couche est la sortie de la couche convolutionnelle précédente. En fonction de la relation entre Y_8 et Y_{16}, la solution pour omettre les sorties indésirables et garder une sortie similaire doit être exploitée dans cette couche.

Tout d'abord, les couches de produits internes des DNN_8 produisent la sortie en tant que vecteur V^8 et calcule la multiplication matricielle avec les poids W^8 en tant que :

$$y^8 = w^8 * v^8 - b^8 \qquad (2.9)$$

Où w^8 et b^8 indiquent le poids et le biais dans la couche de produit interne respectivement. Le même produit intérieur de DNN_{16} est :

$$y^{16} = w^{16} * v^{16} - b^{16} \qquad (2.10)$$

Les paramètres de W^8 et b^8du modèle préformé peuvent être extraits et transformés dans W^{16}et b^{16} .Pour le biais, c'est similaire à celui du convolutionnel couche.

Néanmoins, W^8 et W^{16} n'ont pas les mêmes dimensions, ce qui cause le problème Sur la transformation entre eux. En conséquence, l'équation (2.8) est rappelée, La relation entre Y^8 et Y^{16}de la couche convolutionnelle est illustrée à la figure II.11.

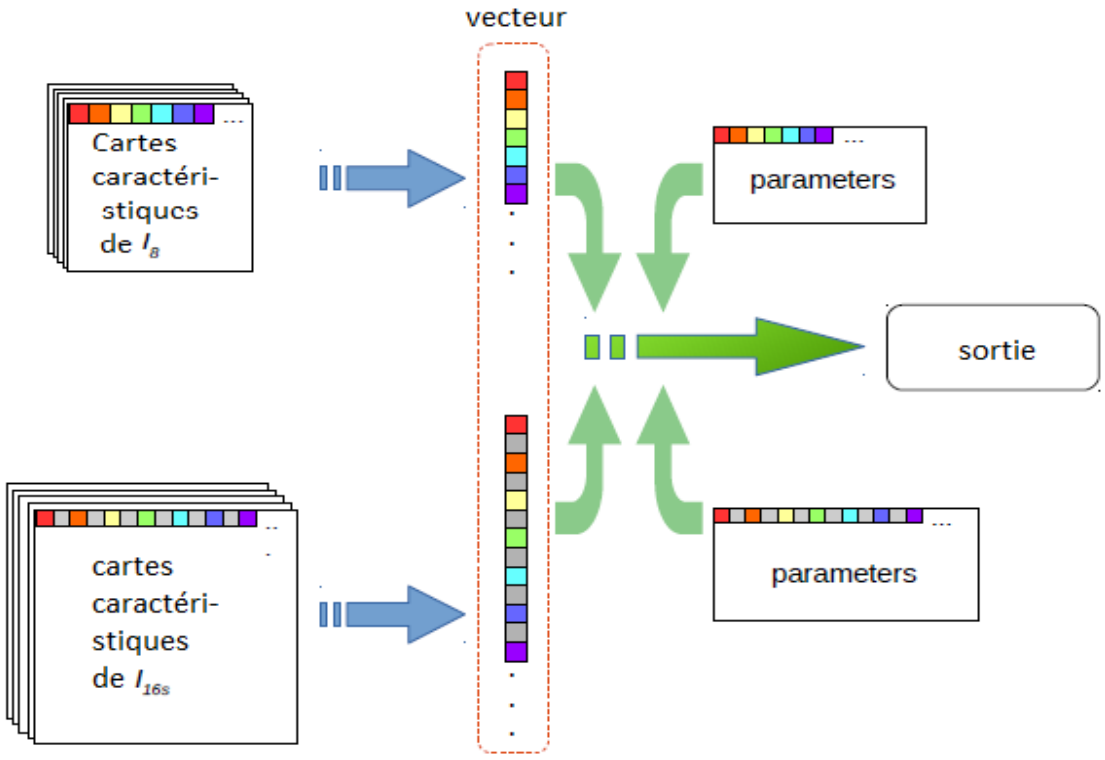


Fig II.11 : Paramètres de transformation dans les couches de produits internes des DNNs

Dans la figure (II.2), il existe deux ensembles de cartes de caractéristiques des couches convolutionnelles de DNN_8 et DNN_{16} à gauche. Les blocs colorés sont représentés par les pixels correspondants qui doivent être conservés, alors que les valeurs des blocs gris doivent être omises dans cette couche.

Ensuite, elles sont remodelées en tant que vecteur pour la prochaine étape de multiplication. Sur le côté droit de la figure, il existe deux matrices de paramètres dans

la couche de produit interne, qui ont également correspondu au codage des couleurs. Par exemple, les blocs rouges dans les deux cartes de caractéristiques et les vecteurs sont les pixels avec les mêmes valeurs, qui doivent être multipliés par les blocs rouges correspondants dans les matrices de paramètres.

Étant donné que les blocs colorés dans les cartes de caractéristiques ou les vecteurs de DNN_{16} doivent être conservés, les blocs dans les paramètres de DNN_8 doivent être mis à la position correspondante de DNN_{16} ,Les blocs gris dans les cartes et les vecteurs de caractéristique de DNN_{16} sont omis et les blocs gris dans les paramètres de DNN_{16} sont mis à zéro. Après avoir multiplié par des matrices de paramètres, les valeurs des blocs gris sont changées en zéros, ce qui n'a pas d'effet sur le résultat. En d'autres termes, on peut ajouter des zéros entre deux valeurs originales de W^8 en direction horizontale et verticale pour générer W^{16}.

Finalement, la solution des paramètres de transformation dans les couches de produits internes a été présentée. Pour les poids, ajoutez la valeur nulle entre deux valeurs originales W^8en nous pour gagner les paramètres transformés de W^{16} c-à-d Les valeurs de zéro peuvent aider à omettre la différence par rapport à la sortie de la couche convolutionnelle et à conserver la sortie similaire à celle de DNN_8 . Une solution directe consiste à copier le b^8 vers b^{16}. Pour les couches de produit internes suivantes avec la même entrée (par exemple, couche fc8), les paramètres peuvent être simplement copiés.

En général, cette sous-section fournit une solution pour traiter les paramètres des filtres à l'échelle supérieure pour la couche comvolutionale et la couche interne du produit. L'algorithme 1 résume la procédure de transformation des paramètres de M_8 à M_{16t}en utilisant notre idée d'apprentissage supplémentaire. Dans les expériences, on peut utiliser la même méthode pour transformer les paramètres de M_{16t} en M_{32t} En bref, les poids dans la couche convolutionnelle doivent être étendus et copiés, alors que dans la couche de produit interne, les zéros doivent être ajoutés. Les vecteurs de polarisation sont copiés dans les deux couches.

Algorithm 1 Transform parameters from M_8 to M_{16t}

```
Load the tiny images network DNN_8 and model M_8
Load the large images network DNN_16
Create the model file M_16t and initialise as 0
for i in all layers do
    copy the bias vector from M_8 to M_16t                ▷ transform the bias
    if i is 'conv1' then          ▷ transform the weights in convolutional layer
        copy and extend the weights by 1/4
    else if i is 'fc6' then       ▷ transform the weights in the first inner product layer
        add zeros and copy the weights
    else
        copy the weights      ▷ transform the weights in the other inner product layer
    end if
end for
```

II.7. Ajout des filtres convolutionnels :

Les réseaux avec des filtres supplémentaires, non seulement étendent les dimensions des filtres, qui sont identiques avec les réseaux avec des filtres à l'échelle, mais aussi ajoutent le nombre d'éléments à la couche convolutionnelle. Ceci nécessite l'initialisation de nouveaux filtres ajoutés dans les expériences, 12 filtres dénommés F_{12} sont appliqués à DNN_8 . F_{16} au DNN_{16} et F_{24} au DNN_{32}. Ensuite, nous présentons deux méthodes pour initialiser les paramètres des nouveaux filtres ajoutés.

II.7.1. Initialisation à Zero :

Le moyen simple est de mettre tous les paramètres des nouveaux filtres ajoutés à zéro. L'utilisation de zéro pour ignorer la sortie supplémentaire de nouveaux filtres dans chaque couche peut réduire l'influence de la sortie, ce qui a pour effet de transformer les paramètres dans la couche de produit interne avec des filtres à l'échelle. En bref, la première méthode consiste à définir tous les poids et la polarisation dans les nouveaux filtres ajoutés comme étant nulles.

II.7.2. Initialisation du basculement des filtres existants :

Après avoir formé un modèle DNN, les filtres convolutifs peuvent être appris et visualisés. Dans le document de Krizhevsky [5], il existe 96 filtres convolutionnels de taille 11 x 11 x 3 émis par la première couche convolutionnelle (conv1) Qui montrent

une variété de filtres sélectifs de fréquence et d'orientation, ainsi que de différentes gouttes colorées (figure II.12). Bien que le nombre de filtres mis en œuvre dans nos expériences soit moindre, ils peuvent encore en apprendre davantage. Ainsi, nous essayons de trouver un moyen d'initialiser les nouveaux filtres ajoutés en tant que filtres supplémentaires. Théoriquement, si les filtres supplémentaires sont nécessaires dans le réseau, ils peuvent stimuler les formations.

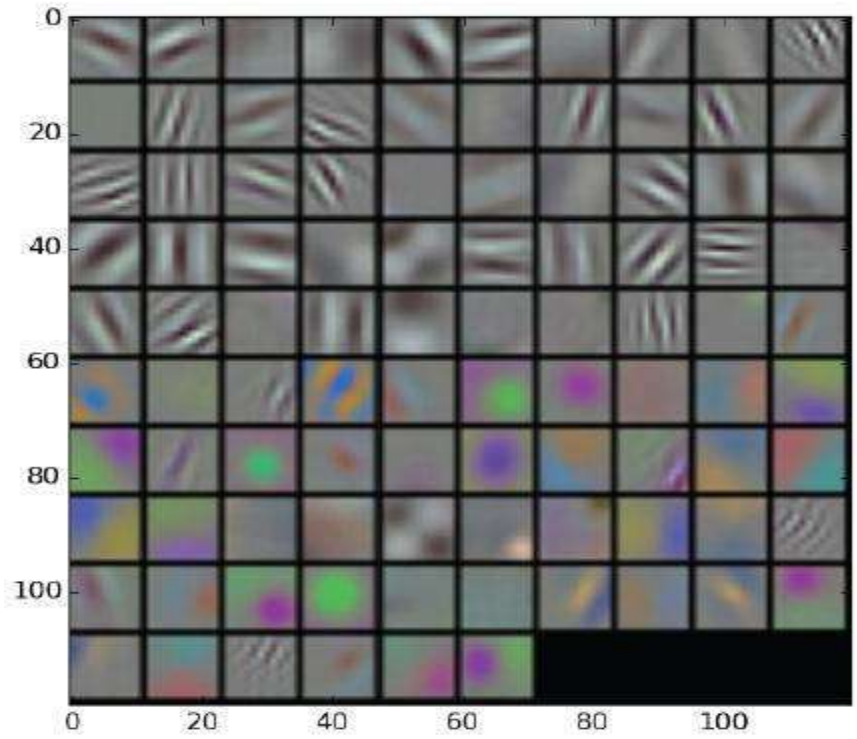


Fig II.12 : Les premières couches convolutives avec filtres par Krizhevsky [5].

De la figure II.12, on constate qu'il existe plusieurs filtres sélectifs par orientation. Si les opérations de copie et de basculement peuvent être exploitées, plus de directions des filtres seront acquises pour extraire des fonctionnalités. Par exemple, si le réseau a appris un filtre sélectif d'orientation dans un angle d'orientation, on peut obtenir plus d'orientations sous différents angles en basculant. Donc, L'idée flipping (de basculement) est proposée pour compléter les filtres, qui incluent le basculement horizontal, vertical, diagonal et central, illustré à la Figure II.13

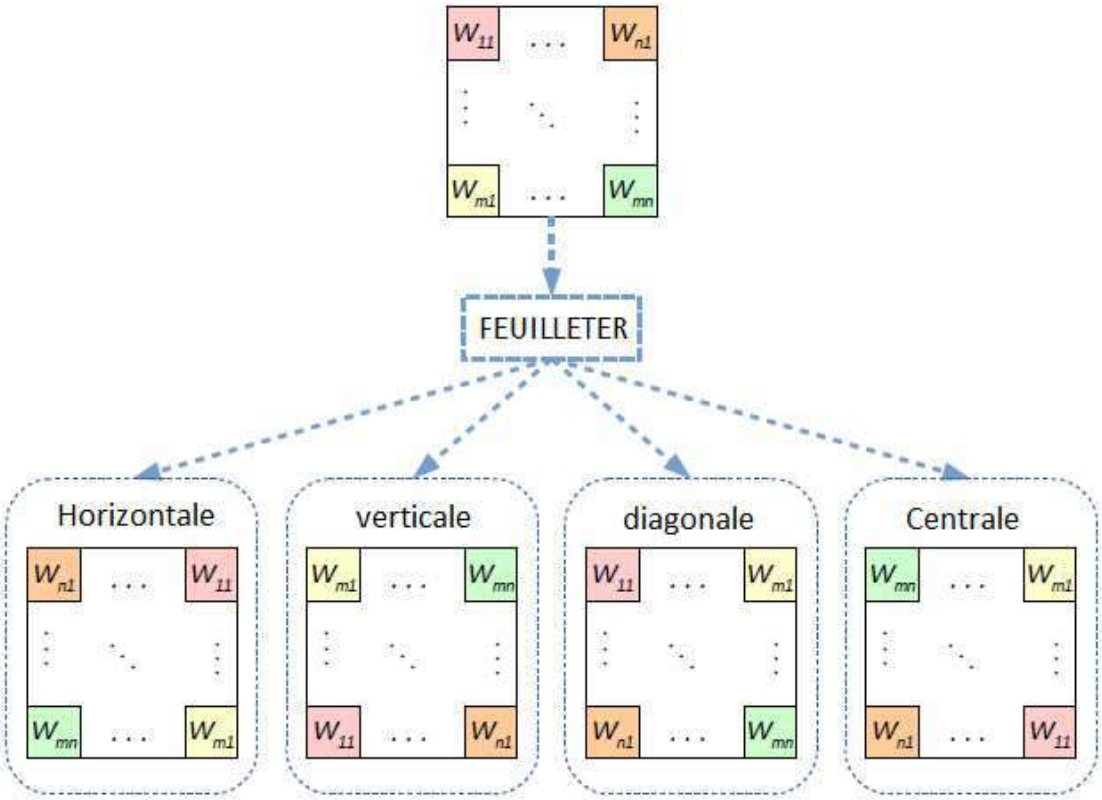


Fig II.13 : Feuilletage des filtres.

En résumé, une solution pour l'ajout de filtres convolutionnels est proposée. En général, la transformation de paramètres supplémentaires peut être divisée en deux parties. L'un est la même partie avec des filtres à l'échelle, qui peuvent être exécutés par la même méthode. L'autre est les nouveaux filtres ajoutés, différentes méthodes initialisées devraient être mises en œuvre. L'algorithme est similaire à Algorithme 1, qui doit initialiser de nouveaux filtres ajoutés. En bref, deux idées principales pour implémenter l'initialisation sont fournies, y compris le zéro, et le basculement des filtres existants. Dans les expériences, celles-ci sont testées par upscaling de DNN_8 au DNN_{16} et de DNN_{16} auDNN_{32}.

II.8. Réglage fin :

Après avoir traité une transformation de paramètres à l'échelle supérieure dans la section précédente, l'idée de l'apprentissage progressif est réalisée grâce à la formation DNN du modèle "fine". Caffe propose une fonction de réglage précis, qui prend un modèle déjà appris, adopte l'architecture et reprend la formation des poids des modèles déjà appris [4]. La commande correspondante est :

```
$ caffe train -solver path/to/solver.prototxt-weights
path/to/weights.caffemodel
```

Dans les expériences, le modèle formé M_8 a été mis à l'échelle et des filtres ont été ajoutés pour former le modèle M_{16t}, qui est utilisé pour I_{16} dans DNN_{16} pour le réglage fin. Une fois le réglage fin, le modèle formé M_{16} peut être obtenu. Et la même méthode est appliquée pour générer le modèle M_{32t}. Enfin, répétez l'étape de réglage fin à I_{32} dans DNN_{32}avec M_{32t}.En conséquence, l'apprentissage incrémental à trois étapes est terminé, en utilisant $DNN_{8,}$ DNN_{16}et DNN_{32}. La figure II.14 est un exemple de journal de réglage où la précision de l'itération 0 commence par celle obtenue avec DNN_8 .

```
I0325 10:48:52.009484 19520 net.cpp:219] Network initialization done.
I0325 10:48:52.009487 19520 net.cpp:220] Memory required for data: 2278208
I0325 10:48:52.009534 19520 solver.cpp:41] Solver scaffolding done.
I0325 10:48:52.009541 19520 caffe.cpp:115] Finetuning from ./TEMPWORK/trans_caffe_ImageNet-tiny-16x16-sRGB-net-3.caffemodel
I0325 10:48:52.018720 19520 solver.cpp:160] Solving CaffeNet
I0325 10:48:52.018745 19520 solver.cpp:161] Learning Rate Policy: step
I0325 10:48:52.018774 19520 solver.cpp:264] Iteration 0, Testing net (#0)
I0325 10:48:55.321952 19520 solver.cpp:315]     Test net output #0: accuracy = 0.0815802
I0325 10:48:55.321985 19520 solver.cpp:315]     Test net output #1: loss = 5.43995 (* 1 = 5.43995 loss)
I0325 10:48:55.343711 19520 solver.cpp:209] Iteration 0, loss = 5.58631
I0325 10:48:55.343741 19520 solver.cpp:224]     Train net output #0: loss = 5.58631 (* 1 = 5.58631 loss)
I0325 10:48:55.343754 19520 solver.cpp:445] Iteration 0, lr = 0.001
I0325 10:48:55.780313 19520 solver.cpp:209] Iteration 20, loss = 5.46389
I0325 10:48:55.780344 19520 solver.cpp:224]     Train net output #0: loss = 5.46389 (* 1 = 5.46389 loss)
I0325 10:48:55.780352 19520 solver.cpp:445] Iteration 20, lr = 0.001
```

Fig II.14 : Sortie du journal Caffe lors du réglage fin

Il est noté dans les instructions de Caffe des astuces pour le réglage fin, qui consiste à apprendre la dernière couche en premier et à réduire le taux d'apprentissage [4]. Sur la base de notre propre situation, seul le taux global d'apprentissage *base_lr* dans le solveur est diminué. Cela rend le modèle plus lentement pour apprendre les nouvelles données et préserve l'initialisation de la pré-formation. Dans les expériences, le taux d'apprentissage est descendu de 10, lors de l'utilisation du modèle transformé M_{16t} pour le réglage fin avec I_{16} et DNN_{16}, et aussi pour M_{32t}

II.9. Conclusion :

En conclusion, la méthode de mise en œuvre de l'apprentissage progressif dans DNN est élaborée, qui se compose de trois parties. Premièrement, il est nécessaire de construire les réseaux avec des filtres à l'échelle. Deuxièmement, les réseaux sont formés avec des filtres supplémentaires. Enfin, la méthode de réglage fin fournie par Caffe est décrite pour compléter l'apprentissage progressif. Sur la base des méthodes présentées ci-dessus, nous allons illustrer les résultats expérimentaux dans le prochain chapitre.

Chapitre III :

Expériences et résultats

III.1. Introduction :

Dans ce chapitre, les expériences sont résumées pour présenter les résultats sur l'ensemble de données ImageNet et le jeu de données Places205. Tout d'abord, les résultats en utilisant les réseaux incrémentaux avec des filtres à l'échelle sont affichés et comparés à la méthode de formation traditionnelle. De plus, il existe des résultats d'initialisation de nouveaux filtres ajoutés basés sur différentes méthodes d'initialisation. Enfin, les résultats des réseaux incrémentaux avec des filtres ajoutés sont illustrés, qui comportent deux évaluations comparatives. L'une est la méthode de formation traditionnelle et l'apprentissage progressif avec des filtres ajoutés, l'autre est le réseau avec des filtres à l'échelle et des filtres ajoutés. À partir des résultats, notre méthode atteint une bonne efficacité et des bonne performances.

III.2. Évaluation de l'apprentissage progressif avec des filtres échelonnés :

Voici le résultat de I_{16s} avec K_4 dans DNN_{16} transformé de I_8 avec K_2 dans DNN_8, Illustré à la figure III.1.

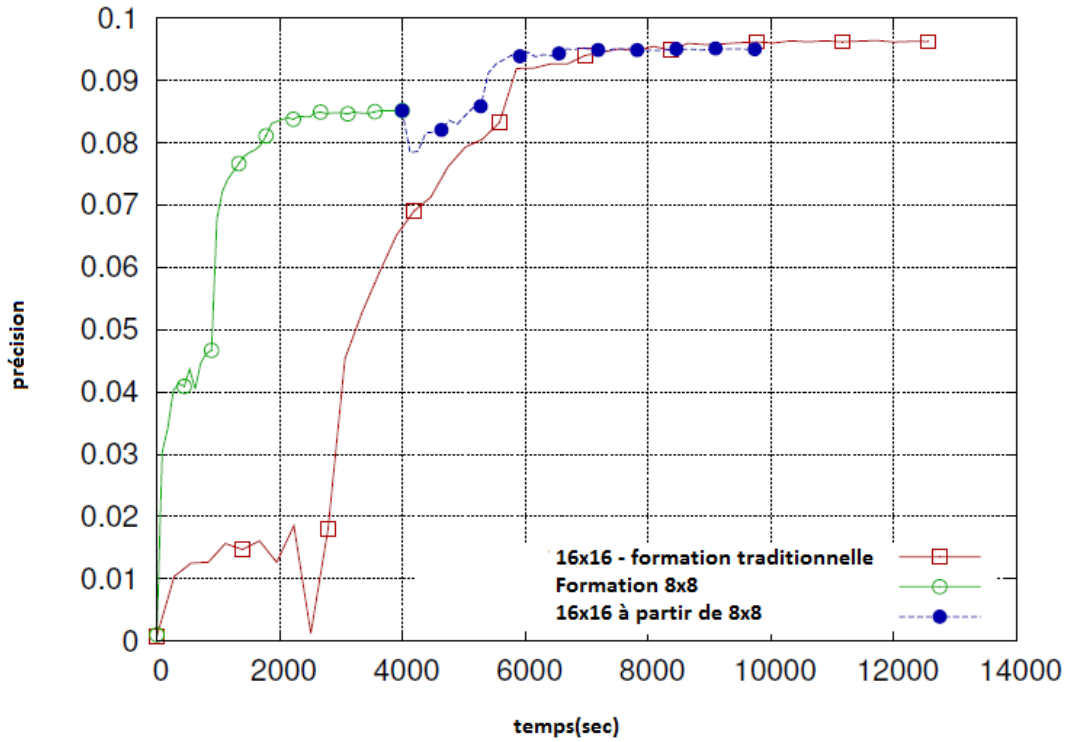


Fig III.1 :Comparaison des courbes de formation de la méthode traditionnelle et de l'apprentissage progressif avec des filtres à l'échelle sur l'ensemble de données ImageNet.

La courbe rouge avec les carrés représente la courbe d'entraînement par la méthode traditionnelle pour la formation des I_{16s} dès le début. Les courbes avec des cercles représentent la méthode d'apprentissage incrémental, alors que le vert et le bleu décrivent les courbes de formation de I_8 et I_{16s}, respectivement. À partir des courbes, il n'y a pas d'écart de performance lors de la transformation de modèles de M_8 à M_{16t} mais des résultats finaux similaires sont obtenus. Cela signifie que le model M_{16t} transformé conserve avec succès l'information originale du modèle grossier préformé M_8 et continue d'apprendre les nouvelles informations dans DNN_{16}. Par conséquent, l'idée de l'apprentissage progressif en transformant les paramètres DNN du réseau d'images grossières vers le réseau d'images fines semble fonctionner.

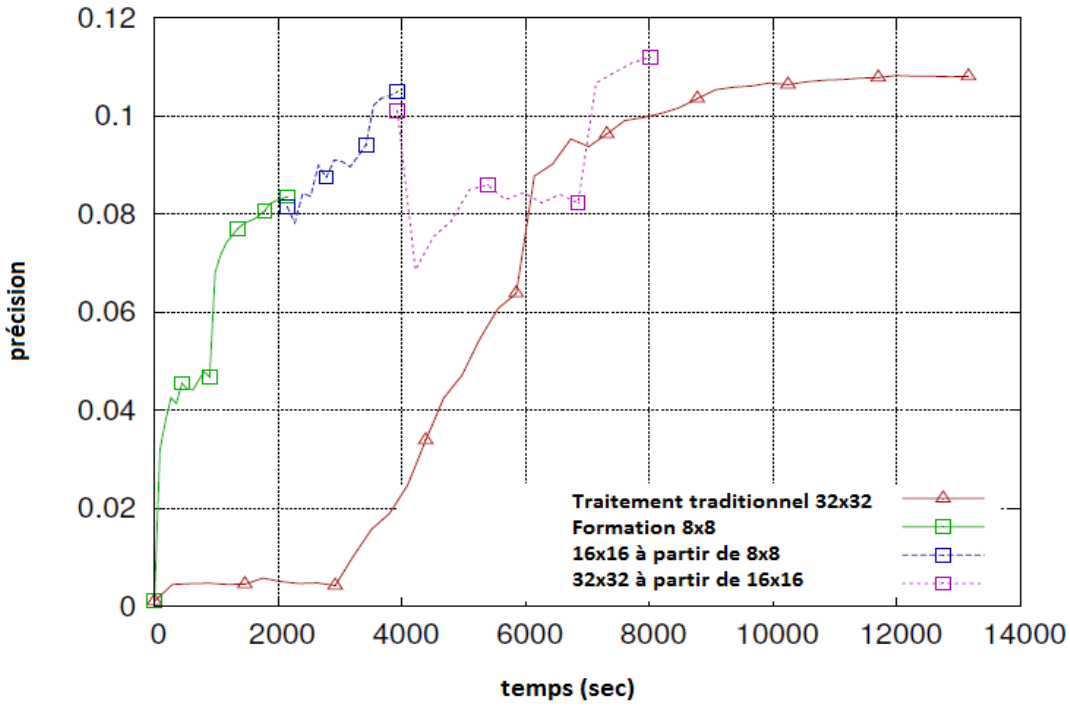


Fig III.2 : Comparaison des courbes de formation de la méthode traditionnelle et de l'apprentissage progressif avec des filtres à l'échelle sur l'ensemble de données ImageNet.

Dans la figure III.2, pour les réseaux avec des filtres à l'échelle, ils sont prolongés deux fois par la même méthode. Le premier est de DNN_8 avec K_2 à DNN_{16} avec K_4, l'autre est en continu de DNN_{16} avec K_4 à DNN_{32} avec K_8. De même, La courbe rouge avec des triangles représente la courbe de formation par méthode traditionnelle pour la formation I_{32} dès le début. Les courbes avec les carrés montrent la méthode d'apprentissage incrémentale, tandis que le vert, le bleu, le magenta décrivent respectivement les courbes d'entraînement de I_8, I_{16} et I_{32}. Il existe deux intervalles lorsque la taille des filtres est mise à l'échelle. La raison en est que de nouveaux détails ont été ajoutés à la contribution et doivent être appris par DNN. Par exemple, il existe un écart de transformation de DNN_8 à DNN_{16}. Il est prévu que I_{16} apporte de nouvelles fonctionnalités visuelles, le I_8 n'a pas, ce qui doit être appris par DNN_{16}. On sait que DNN_{16} apprend les nouveaux détails à mesure que la courbe s'améliore rapidement. La transformation de DNN_{16} à DNN_{32} a la même situation.

En conséquence, la précision peut être améliorée de 10,8% à 11,2% et ne coûte que 61% de temps, comme indiqué dans le tableau III.1.

	précision	Temps (sec)
Méthode traditionnel	10.8 %	13160
Filtres à l'échelle	11.2 %	8011

Tab III.1:Précision finale de la méthode traditionnelle et apprentissage progressif avec des filtres à l'échelle sur l'ensemble de données ImageNet.

En outre, le jeu de données Places205 est également appliqué pour vérifier l'idée de filtres à l'échelle, qui utilise la même structure de réseaux avec l'ensemble de données ImageNet. Les résultats sont présentés à la Figure III.4 et au Tableau III.2, où la méthode incrémentale enregistre 38% du temps de formation et augmente la précision de 14,1% à 15,9%

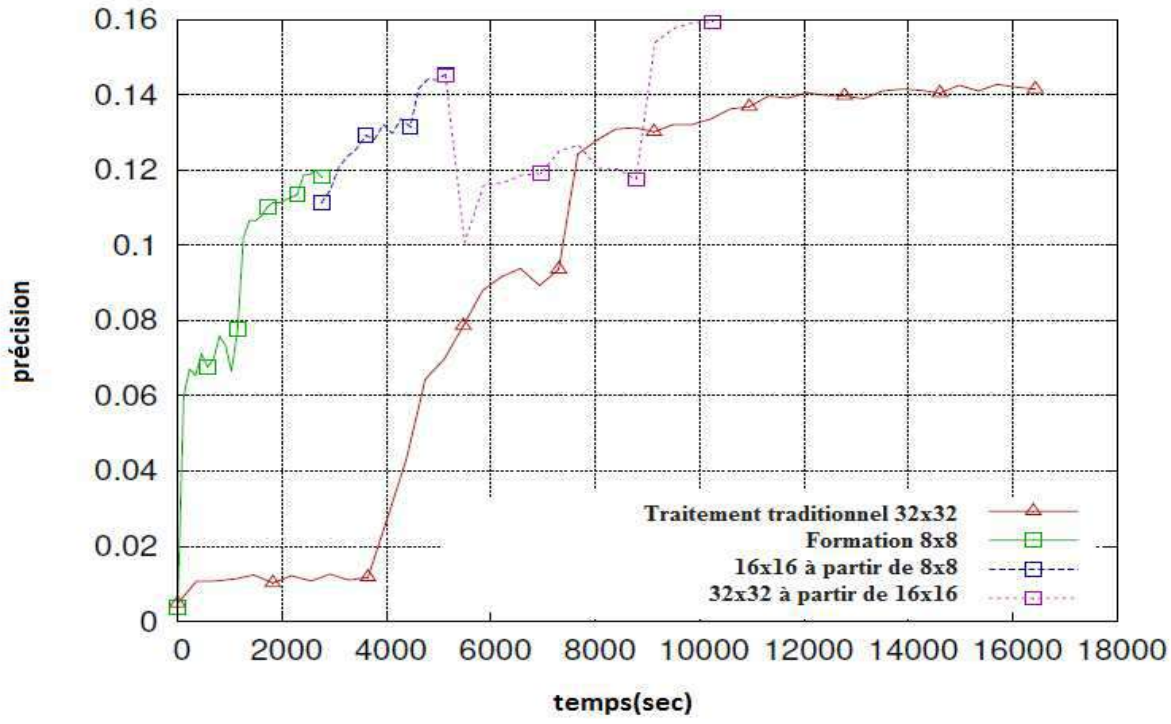


Fig III.3 : Comparaison des courbes de formation de la méthode traditionnelle et de l'apprentissage progressif avec des filtres à l'échelle sur le jeu de données Places205.

	précision	Temps (sec)
Méthode traditionnel	14,1 %	16431
Filtres à l'échelle	15,9 %	10239

Tab III.2 : Précisions finales de la méthode traditionnelle et apprentissage progressif avec des filtres à l'échelle sur le jeu de données Places205.

III.3. Expériences sur l'ajout et l'initialisation de nouveaux filtres convolutionnels :

Avant de montrer les résultats finaux de la formation progressive avec des filtres ajoutés, nous montrons les résultats de l'initialisation de nouveaux filtres par différentes méthodes.

Dans cette partie, les paramètres de la couche convolutionnelle sont F_{12} avec K_4 en I_{16} et F_{24} avec K_8 en I_{32}. La méthode des filtres à l'échelle est identique à l'expérience précédente et le nombre de filtres est doublé de 12 à 24. En conséquence, les 12 filtres supplémentaires sont nécessaires pour initialiser avec zéro des valeurs ou en retournant les filtres existants .

Ensuite, la méthode de basculement est implémentée, ce qui peut générer des filtres supplémentaires au DNN. Ici, quatre directions de basculement, horizontales, verticales, diagonales et centrales, sont utilisées dans les expériences, dont les courbes d'entraînement sont illustrées à la figure III.6. La méthode de basculement peut obtenir des résultats similaires à la fin, tandis que les points de départ sont très différents.

Étant donné que certains filtres sont nouveaux dans le réseau, ils peuvent influencer la sortie en donnant différentes valeurs au modèle préformé.

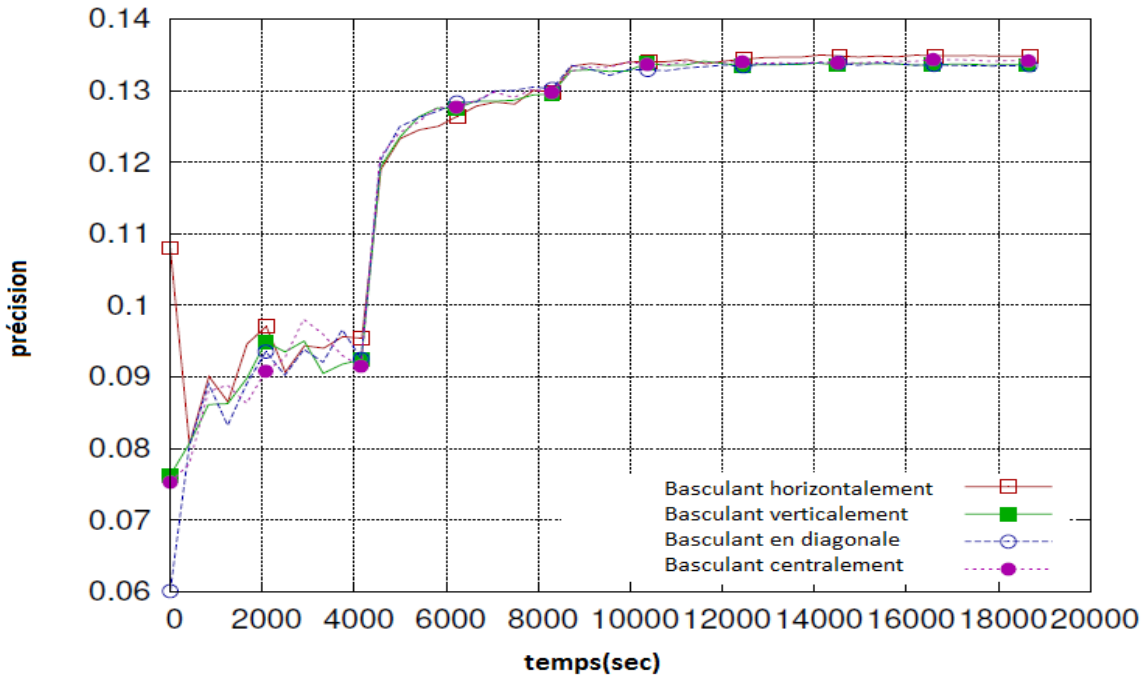


Fig III.4 : Résultats lors de l'initialisation de nouveaux filtres ajoutés comme basculant horizontalement, verticalement, en diagonale et centralement sur l'ensemble de données ImageNet.

Pour conclure, plusieurs méthodes sont fournies pour initialiser les nouveaux filtres ajoutés. La plupart d'entre eux peuvent obtenir des résultats similaires, de sorte que la manière originale, les valeurs gaussiennes aléatoires se divisent par 100, sont utilisées dans les expériences suivantes.

III.7 Évaluation de l'apprentissage incrémentiel avec des filtres ajoutés :

Pour évaluer l'effet des réseaux avec des filtres ajoutés, le nombre de filtres peut être séparé en deux parties. Un est le même nombre de filtres avec le réseau "grossier", l'autre est le nouveau filtre supplémentaire dans le réseau "fine". Pour la même partie, la méthode de mise à l'échelle des filtres est utilisée pour transformer les paramètres.

Pour la partie supplémentaire, la méthode d'initialisation est appliquée. Par exemple, la transformation de DNN_8 avec F_{12} de K_2 en DNN_{16} avec F_{16} de K_4 se compose de deux parties. Les 12 premiers filtres de DNN_{16} sont transformés des filtres de DNN_8, tandis que les 4 nouveaux filtres sont initialisés par des valeurs gaussiennes aléatoires divisées par 100.

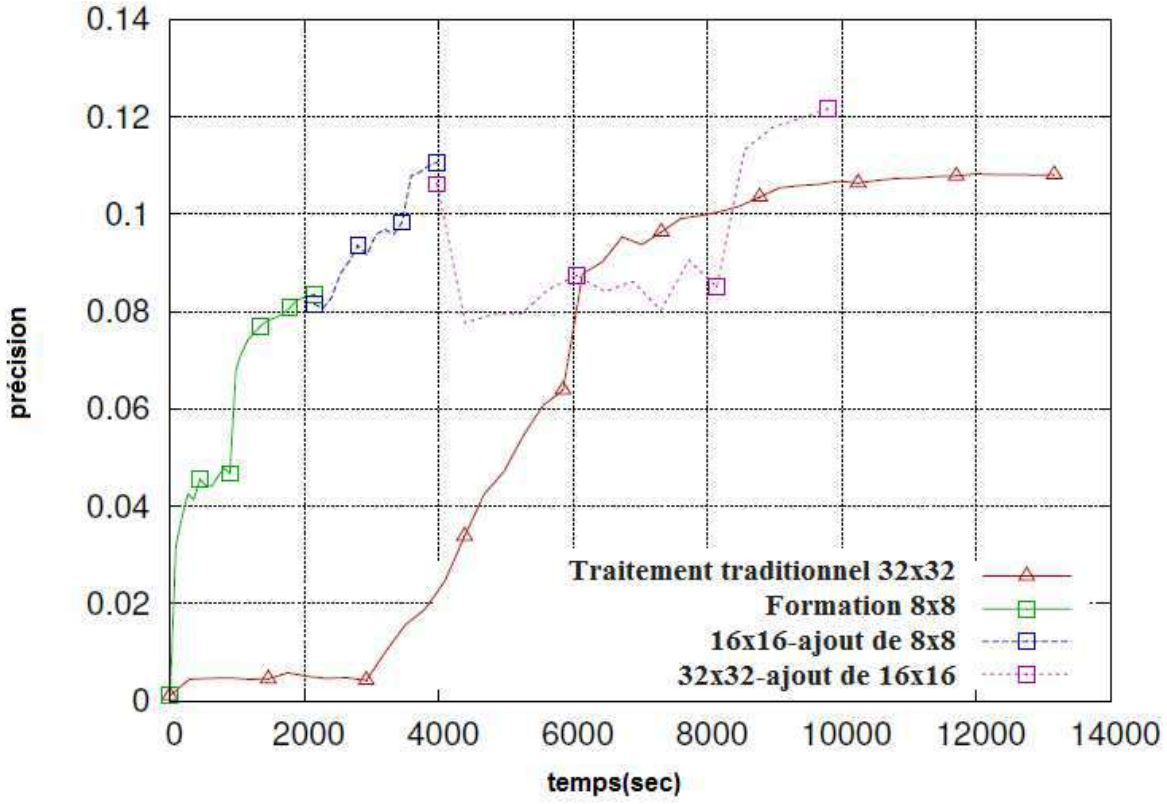


Fig III.5 : Résultats de la méthode traditionnelle et de l'apprentissage progressif avec des filtres ajoutés et à l'échelle pour l'ensemble de données ImageNet.

	précision	Temps (sec)
Méthode traditionnel	10,8 %	13160
Filtre ajouter	12,1 %	9794

Tab III.3 : Précision finale de la méthode traditionnelle et apprentissage progressif avec des filtres ajoutés sur l'ensemble de données ImageNet.

Dans les expériences, le nombre de filtres est augmenté deux fois par la même méthode. Le premier est de DNN_8 avec F_{12} de K_2 à DNN_{16} avec F_{16} de K_4. Deuxièmement, ajoutez progressivement à DNN_{32} avec F_{24} de K_8. Les résultats de la méthode traditionnelle et de la méthode d'apprentissage différentiel sont présentés à la Figure III.5. La courbe rouge avec des triangles représente la courbe de formation par la méthode traditionnelle pour la formation I_{32} dès le début. Les courbes avec les carrés montrent le résultat de la méthode d'apprentissage incrémentale, et le vert, le bleu, le magenta décrivent les courbes de formation de I_8, I_{16}, I_{32}, en conséquence. En conséquence, la précision peut être améliorée à 12,1% et n'utilise que 74% de temps, comme indiqué au tableau III.3.

Les résultats des réseaux avec des filtres ajoutés en utilisant le jeu de données Places205 sont présentés à la Figure III.6 et au Tableau III.4. Dans ces expériences, la même structure de réseaux avec l'ensemble de données ImageNet est implémentée dans l'ajout de filtres, ce qui augmente la performance de 14,1% à 17% et ne passe que 66% du temps.

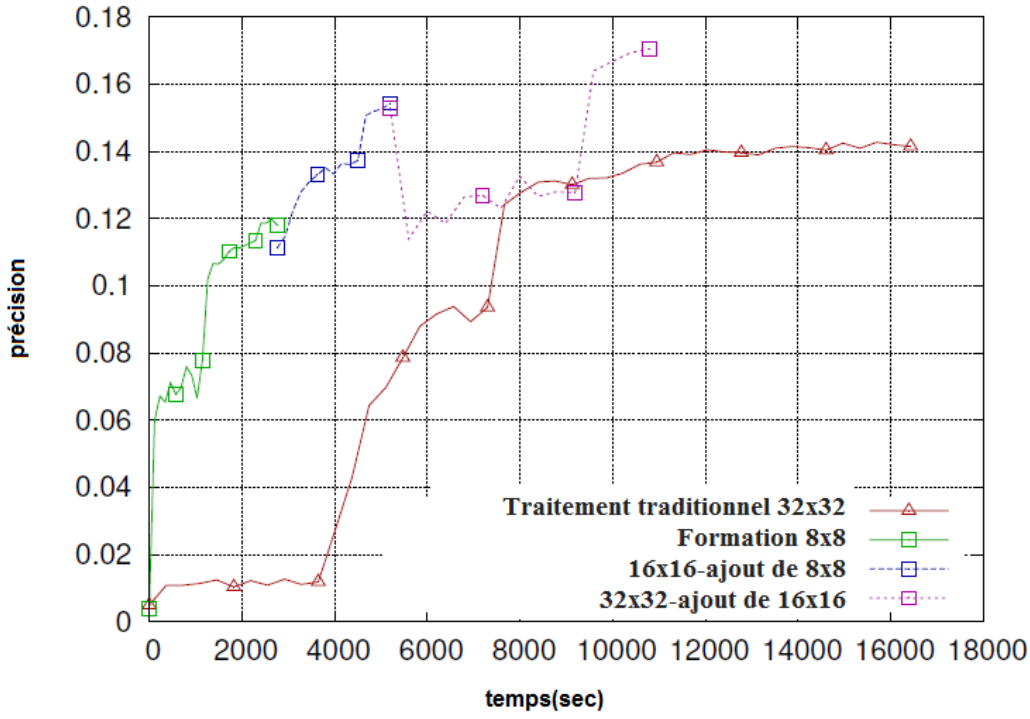


Fig III.6 :Résultats de la méthode traditionnelle et de l'apprentissage progressif avec des filtres ajoutés et à l'échelle pour l'ensemble de données Places205.

	Précision	Temps (sec)
Méthode traditionnel	14,1 %	16431
Filtre ajouter	17 %	10786

Tab III.4 : Précision finale de la méthode traditionnelle et de l'apprentissage progressif avec des filtres ajoutés sur le jeu de données Places205.

Enfin, on compare deux résultats de méthodes d'apprentissage incrémentales, notamment des filtres à l'échelle et des filtres ajoutés pour l'ensemble de données ImageNet et le jeu de données Places205, illustré à la Figure III.6. La courbe rouge avec des triangles représente la méthode d'entraînement traditionnelle pour I_{32}. La courbe cyan avec des diamants montre la procédure de formation de I_8. Les deux courbes bleues aux carrés représentent la méthode des filtres à l'échelle, tandis que les deux courbes vertes aux cercles sont la méthode des filtres ajoutés. Les deux méthodes consacrent moins de temps à une performance supérieure à celle de la méthode de formation traditionnelle.

Et voici un simple exemple qui démontre le concept de notre méthode utilisée :



Figure III.7 : Un DNN avec plusieur couches cachées

III.8 conclusion :

Les deux méthodes d'apprentissage progressif sont mises en œuvre avec succès. Les deux peuvent améliorer l'efficacité et augmenter leurs performances. Bien que le temps de formation des réseaux avec des filtres à l'échelle soit plus court, les DNN ont besoin de plus de filtres et d'une plus grande taille de filtres pour apprendre des images avec une résolution supérieure ou plus de détails.

Conclusion générale

Ce mémoire propose une méthode prometteuse, un apprentissage progressif, pour résoudre la tâche de la classification de l'image. Il se révèle être un outil efficace pour la tâche en utilisant des réseaux neuronaux profonds. L'un des avantages est qu'un modèle pré-formé peut être utilisé comme l'initialisation d'autres réseaux pour raccourcir le temps de formation. La formation longue durée des ensembles de données à grande échelle, p.ex. L'ensemble de données ImageNet est un problème inquiétant. La méthode d'apprentissage incrémentale peut atteindre une efficacité supérieure pour la formation sans sacrifier toute précision.
Un autre point bénéfique est que la performance peut s'améliorer légèrement. Donc, L'objectif principal du travail est d'améliorer l'efficacité et la performance grâce à des méthodes d'apprentissage progressives.

Il existe deux types de méthodes d'apprentissage incrémentales basées sur les DNN. L'un est les réseaux avec des filtres à l'échelle, qui peuvent améliorer la précision de 10,8% à 11,2% et de 14,1% à 15,9% pour l'ensemble de données ImageNet et le jeu de données Places205, respectivement. Bien qu'ils ne soient pas significativement améliorés, les temps de formation ont été réduits de 39% et de 37% pour ces deux ensembles de données, ce qui signifie qu'il faut moins de temps pour obtenir une précision similaire. L'autre méthode est les réseaux avec des filtres ajoutés, qui augmentent la précision à 12,1% pour l'ensemble de données ImageNet et à 17,0% pour le jeu de données Places205. Les temps de calcul ont diminué de 25% et 34% pour la formation, en conséquence. Ces résultats impliquent que la performance peut également être améliorée alors que l'apprentissage est plus efficace. Par conséquent, l'efficacité et la performance ont été améliorées par l'apprentissage progressif.

Les deux méthodes d'apprentissage progressif ont été mises en œuvre dans Caffe. D'une part, le nombre de filtres est conservé et la taille des filtres est étendue dans la couche convolutionnelle pour former les réseaux avec des filtres à l'échelle. Les paramètres des filtres sont copiés et étendus avec un petit ajustement du réseau "grossier" au réseau "fin" pour garder les informations acquises à partir d'images à basse résolution. D'autre part, le nombre de filtres et la taille des filtres sont modifiés pour construire les réseaux avec des filtres ajoutés.
De plus, une approche appropriée d'initialisation des nouveaux filtres ajoutés a été testée et les valeurs gaussiennes aléatoires divisées par 100 ont été jugées les meilleures.

L'idée principale de l'apprentissage progressif est de conserver et de transformer l'information savante du modèle pré-formé d'images à basse résolution en un nouveau modèle "fin" et d'apprendre continuellement parmi les images de résolution de plus en plus grande. Dans une mise en œuvre de ces méthodes, il faut faire attention aux paramètres de formation, sans oublier les paramètres pré-formés.

Les méthodes d'apprentissage progressif peuvent améliorer l'efficacité et la performance, et il existe des orientations futures prometteuses. Tout d'abord, la taille particulière des filtres à l'échelle peut être nécessaire, comme la transformation des paramètres des filtres de K_2 en K_3. Dans les expériences, la taille des filtres est seulement doublée par chaque étape d'apprentissage progressif.
Deuxièmement, une procédure de formation intégrée à partir d'images «grossières» en images de taille normale peut favoriser l'amélioration de l'efficacité et de la performance. Enfin, pour former des images à haute résolution, la méthode d'ajout de couches dans les CNN est un problème difficile et prometteur qui doit être étudié à l'avenir.

Exemple de fichier de configuration réseau Pour DNN8 :

```
name: "CaffeNet"
layers {
  name: "data"
  type: DATA
  top: "data"
  top: "label"
  data_param {
   source: "path/to/training/set"
   backend: LMDB
   batch_size: 256
  }
  transform_param {
   mean_file: "path/to/meanfile"
   mirror: false
  }
  include: { phase: TRAIN }
 }
  layers {
   name: "data"
   type: DATA
   top: "data"
   top: "label"
   data_param {
    source: "path/to/validation/set"
    backend: LMDB
    batch_size: 50
   }
```

```
    transform_param {
     mean_file: "path/to/meanfile"
     mirror: false
    }
    include: { phase: TEST }
}
layers {
    name: "conv1"
    type: CONVOLUTION
    bottom: "data"
    top: "conv1"
    blobs_lr: 1
    blobs_lr: 2
    weight_decay: 1
    weight_decay: 0
    convolution_param {
     num_output: 12 # the number of filters
     kernel_size: 2 # the size of filters
     stride: 1
     weight_filler {
      type: "gaussian"
      std: 0.01
     }
```

```
    bias_filler {
     type: "constant"
     value: 0
     }
    }
  }
layers {
  name: "relu1"
  type: RELU
  bottom: "conv1"
  top: "conv1"
}
layers {
  name: "norm1"
  type: LRN
  bottom: "conv1"
  top: "norm1"
  lrn_param {
   local_size: 5
   alpha: 0.0001
   beta: 0.75
  }
}
layers {
   name: "fc6"
   type: INNER_PRODUCT
   bottom: "norm1"
   top: "fc6"
```

```
    blobs_lr: 1
    blobs_lr: 2
    weight_decay: 1
    weight_decay: 0
    inner_product_param {
     num_output: 512
     weight_filler {
      type: "gaussian"
      std: 0.005
     }
     bias_filler {
      type: "constant"
      value: 1
     }
   }
}
layers {
  name: "relu6"
  type: RELU
  bottom: "fc6"
  top: "fc6"
}
layers {
  name: "drop6"
  type: DROPOUT
  bottom: "fc6"
  top: "fc6"
```

```
dropout_param {
     dropout_ratio: 0.5
    }
layers {
   name: "fc8"
   type: INNER_PRODUCT
   bottom: "fc6"
   top: "fc8"
   blobs_lr: 1
   blobs_lr: 2
   weight_decay: 1
   weight_decay: 0
   inner_product_param {
    num_output: 1000
    weight_filler {
     type: "gaussian"
     std: 0.01
    }
    bias_filler {
     type: "constant"
     value: 0
    }
  }
 }
```

```
layers {
  name: "accuracy"
  type: ACCURACY
  bottom: "fc8"
  bottom: "label"
  top: "accuracy"
  include: { phase: TEST }
}
layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "fc8"
  bottom: "label"
  top: "loss"
}
```

Annex B : Exemple d'un fichier de résolution de réseau

```
net: "path/to/network/configuration/file"
test_iter: 1000
test_interval: 1000
base_lr: 0.01
lr_policy: "step"
gamma: 0.1
stepsize: 50000
display: 20
max_iter: 120000
momentum: 0.9
weight_decay: 0.0005
snapshot: 10000
```

```
snapshot_prefix: "path/to/save/snapshot/file"
solver_mode: GPU
```

Bibliographie

[1] Graham Taylor. Unsupervised learning - tutorial on deep learning for vision. In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, 2014.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. In Neural Information Processing Systems (NIPS), 2012.

[3] Simon J.D. Prince. Computer Vision: Models Learning and Inference. Cambridge University Press, 2012.

[4] Li Deng and Dong Yu. Deep learning: Methods and applications. Technical Report MSR-TR-2014-21, Microsoft Research, May 2014.

[5] Simon Haykin. Neural Networks: A Comprehensive Foundation. Prentice Hall PTR

[6] Michiel Hermans and Benjamin Schrauwen. Training and analysing deep recurrent neural networks. In Neural Information Processing Systems (NIPS)

[7] Li Deng and Dong Yu. Deep learning: Methods and applications. Technical Report MSR-TR-2014-21, Microsoft Research, May 2014.

[8] Yoshua Bengio. Learning deep architectures for AI. Foundations and Trends in Machine Learning, 2(1):1–127, January 2009.

[9] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In Neural Information Processing Systems (NIPS)

[10] K. Ramesha, N. Srikanth, K.B. Raja, K.R. Venugopal, and L.M. Patnaik. Advanced biometric identification on face, gender and age recognition. In Advances in

Recent Technologies in Communication and Computing, 2009. ARTCom '09. International Conference

[11] Yuichi Hirano and Tomoharu Nagao. Feature transformation using filter array for automatic construction of image classification. In Computational Intelligence and Applications (IWCIA), 2014 IEEE 7th International Workshop

[12] Wikipedia. Statistical classification, 2015. [The last modified on 13 June 2015].

[13] Ashley Walker Robert Fisher, Simon Perkins and Erik Wolfart. Hypermedia Image Processing Reference. Online Book, 2000.

[14] Jiawei Han and Micheline Kamber. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers Inc., second edition edition, 2005.

[15] Wikipedia. Test set, 2015. [The last modified on 20 June 2015].

[16] Ke-Lin Du and M. N.S. Swamy. Neural Networks and Statistical Learning. Springer Publishing Company, Incorporated, 2013.

[17] Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between class attribute transfer. In Computer Vision and Pattern Recognition (CVPR), 2009 IEEE Conference

[18] Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. Attribute-based classification for zero-shot visual object categorization. Pattern Analysis and Machine Intelligence, IEEE Transactions

[19] Daniel N. Osherson, Joshua Stern, Ormond Wilkie, Michael Stob, and Edward E. Smith. Default probability. Cognitive Science

[20] Charles Kemp, Joshua B. Tenenbaum, Thomas L. Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. In Proceedings of the 21st National Conference on Artificial Intelligence

[21] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition, 2011.

[22] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference

[23] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[24] Thomas Berg, Jiongxin Liu, Seung Woo Lee, Michelle L. Alexander, David W. Jacobs, and Peter N. Belhumeur. Birdsnap: Large-scale fine-grained visual categorization of birds. In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference

[25] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In Computer Vision and Pattern Recognition (CVPR), 2009 IEEE Conference

[26] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

[27] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander

C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. International Journal of Computer Vision (IJCV)

[28] Igor Aleksander and Helen Morton. An introduction to neural computing. Chapman and Hall, 1990.

[29] Thomas M. Mitchell. Machine Learning. McGraw-Hill, Inc., New York, NY, USA,edition, 1997.

[30] Wee Kheng Leow and R. Miikkulainen. Representing visual schemas in neural networks for scene analysis. In Neural Networks, 1993., IEEE International Conference

[31] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J. Lang. Phoneme recognition using time-delay neural networks. Acoustics, Speech and Signal Processing, IEEE Transactions

[32] Hakan Arslan and Ahmet Kuzucu. A robot control application with neural networks.In Industrial Electronics, 1997. ISIE ’97., Proceedings of the IEEE International Symposium

[33] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors.

[34] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J.Williams. Learning internal representations by error propagation. In David E. Rumelhart, James L.McClelland, and CORPORATE PDP Research Group, editors, Parallel Distributed Processing: Explorations in the Microstructure of Cognition

[35] Christopher M. Bishop. Pattern Recognition and Machine Learning. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[36] Andrew R. Webb and Keith D. Copsey. Statistical Pattern Recognition. Wiley, edition, 2011

[37] Arthur Earl Bryson and Yu-Chi Ho. Applied Optimal Control: Optimization, Estimation and Control. Blaisdell Publishing, 1969.

[38] Stuart J. Russell and Peter Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall, 2010.

[39] Paul J. Werbos. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Harvard University Press, 1974.

[40] Ethem Alpaydin. Introduction to Machine Learning. The MIT Press, 2nd edition, 2010.

[41] Richard O. Duda, Peter E. Hart, and David G. Stork. Pattern classification. Wiley, 2001.

[42] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning, Data Mining, Interference, and Prediction. Springer, 2009.

[43] BernardWidrow and Marcian E. Hoff. Adaptive switching circuits. In James A. Anderson and Edward Rosenfeld, editors, Neurocomputing: Foundations of Research

[44] Alex Graves. Supervised sequence labelling with recurrent neural networks. Springer, 2012.

[45] Alex Graves, Marcus Liwicki, Santiago Fernandez, Roman Bertolami, Horst Bunke, and Jurgen Schmidhuber. A novel connectionist system for unconstrained

handwriting recognition. Pattern Analysis and Machine Intelligence, IEEE Transactions

[46] Jurgen Schmidhuber. Deep learning in neural networks: An overview. Neural Networks

[47] Herbert Jaeger. A tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the "echo state network" approach. Technical report, Fraunhofer Institute for Autonomous Intelligent Systems (AIS), 2013.

[48] Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. Computing Research Repository (CoRR)

[49] Yoshua Bengio and Aaron Courville. Deep learning of representations. In Monica Bianchini, Marco Maggini, and Lakhmi C. Jain, editors, Handbook on Neural Information Processing, pages 1–28. Springer, 2013.

[50] Jurgen Schmidhuber. Multi-column deep neural networks for image classification. In Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference

[51] Marc'Aurelio Ranzato. Supervised deep learning - tutorial on deep learning for vision. In Computer Vision and Pattern Recognition (CVPR)

[52] Andrew Ng, Jiquan Ngiam, Chuan Yu Foo, Yifan Mai, Caroline Suen, Adam Coates, Andrew Maas, Awni Hannun, Brody Huval, Tao Wang, and Sameep Tandon. Unsupervised feature learning and deep learning. Technical report, Stanford University

[53] Wikipedia. Convolutional neural network, 2015. [The last modified on 15 June 2015].

[54] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE

[55] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. Science

[56] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. Neural computation

[57] Ilya Sutskever and Geoffrey E. Hinton. Learning multilevel distributed representations for high-dimensional sequences. In Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, (AISTATS)

[58] Geoffrey E. Hinton. Deep belief networks. Scholarpedia, 4(5):5947, 2009.

[59] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. Computing Research Repository (CoRR), abs/1303.5778, 2013.

[60] Andrea Vedaldi and Karel Lenc. Matconvnet - convolutional neural networks for MATLAB. Computing Research Repository (CoRR), abs/1412.4564, 2014.

[61] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In International Conference on Learning Representations (ICLR), 2014.

[62] MediaWiki. Incremental learning, 2015. [The last modified on 20 January 2015].

[63] Xin Geng and Kate Smith-Miles. Incremental learning. In StanZ. Li and Anil Jain, editors, Encyclopedia of Biometrics, pages 731–735. Springer US, 2009.

[64] Wikipedia. Normal distribution, 2015. [The last modified on 4 July 2015].

Résume

Au cours du temps la classification d'image s'est devenue un problème, les réseaux neuronaux artificiel résoudre les différents problèmes de classification et parmis eux la classification d'image.

Dans ce mémoire on a étudié la classification d'image dans les DNN et l'amélioration de la résolution de cette dernière.

Mots-clé : Les réseaux DNN, réseaux de neuronaux, jeux de données

---

Summary

Over time image classification of image has become a problem; artificial neuron networks resolve different kind of classification and one of this is classification of image.

On this memoir, we practice the classification of image on the DNN and ameliorate resolution of image.

Key-words: DNN, neural networks, datasets

ملخص

مع مرور الوقت أصبح تصنيف الصور مشكل و الشبكة العصبونية الاصطناعية تحل مختلف مشاكل التصنيفات و من بينها تصنيف الصور

في هذه المذكرة تطرقنا الى دراسة تصنيف الصور و تحسين جودة الصور.

كلمات مفتاحية: الشبكة DNN, الشبكة العصبونية, لعبة المعلومات