

PFE

Khalil and Samy

10 juin 2016

Table des matières

1 Recherche d'image par le contenu	2
1.1 Introduction	2
1.2 La conception d'un système CBIR	3
1.3 Calcul de similarité	3
1.4 Problème du faussé sémantique	4
1.5 Types de CBIR [Shr et al. 13]	4
1.5.1 Basée-région	4
1.5.2 Basée-objets	4
1.5.3 Basée-exemples	5
1.5.4 Basée-Feedback	5
1.6 Systèmes de recherche d'images existants	5
1.7 Conclusion	7
2 Apprentissage profond	8
2.1 Introduction	8
2.2 Apprentissage automatique	8
2.3 Concepts de base de l'apprentissage automatique	9
2.3.1 Les tâches T	9
2.3.2 Les mesures de performance P	10
2.3.3 L'expérience E	10
2.3.4 Exemple d'algorithme d'apprentissage : Les réseaux de neurones	11
2.4 Introduction à l'apprentissage profond	12
2.5 Les architectures de l'apprentissage profond	14
2.5.1 Deep Belief Networks	14
2.5.2 Réseau de neurones à convolution	14
2.5.3 Deep AutoEncoders	17
2.6 Les exploits des techniques d'apprentissage profond [Site2]	18
2.7 Conclusion	19
3 Représentation sémantique d'images	20
3.1 Introduction	20
3.2 Recherche d'images par le contenu basée exemple	20
3.3 Architecture du réseau à convolution	22
3.4 Entraînement du réseau	26
3.5 Extraction des caractéristiques	27
3.5.1 Dernière couche Soft-max 1000	27
3.5.2 Avant avant dernière couche 4096a	28
3.5.3 Avant dernière couche 4096b	29
3.5.4 Résultats	29
3.6 La description sémantique	30
3.6.1 Réseau à déconvolution	30
3.6.2 Visualisation des caractéristiques	31
3.7 Description sémantique avec les Autoencoders	33
3.7.1 Autoencoder 4x1x4	33
3.7.2 Autoencoder 4x2x1x2x4	34

3.7.3	Denoising Autoencoder 4x1x4	35
3.7.4	Résultats	36
3.8	La force des Deep Autoencoders	37
3.9	Recherche avec descripteur d'image	38
3.9.1	4096b avec descripteur	38
3.9.2	Denoising Autoencoder avec descripteur	39
3.9.3	Résultats	40
3.10	Comparaison avec d'autres systèmes CBIR	40
3.11	Conclusion	43
4	Application	44
4.1	Introduction	44
4.2	Outils utilisés	44
4.2.1	Le langage Python	44
4.2.2	Theano	44
4.2.3	Autres Frameworks	47
4.3	Interface de l'application	47
4.4	Tests	49
4.5	Conclusion	49

Introduction générale

Durant ces dernières années, beaucoup de chercheurs de plusieurs domaines et surtout dans de très grands laboratoires de recherche, se mettent à suivre une nouvelle tendance, connue sous le nom de Deep Learning ou apprentissage profond. Cette technologie d'apprentissage automatique est basée sur les réseaux de neurones artificiels, et elle a complètement bouleversé le domaine de l'intelligence artificielle.

Même en étant très récent, l'apprentissage profond commence à faire partie de notre quotidien, de la reconnaissance faciale, le tagging automatique de morceaux de musique, la reconnaissance vocale, l'étiquetage automatique d'images et même la conception de nouvelles molécules pharmaceutiques, etc.

Nous visons en premier, dans notre travail, à explorer et à essayer de comprendre ce phénomène. Nous essayerons de comprendre les nouvelles techniques que propose l'apprentissage profond en les appliquant dans un problème très connu en vision artificielle, qui est la recherche d'image par le contenu. Dû aux résultats très prometteurs dans beaucoup de domaines, et surtout dans celui du traitement d'image, qui ont été atteint grâce à l'apprentissage profond. Nous nous attendons à ce que les nôtres soient appréciables.

L'idée de notre approche est de permettre à la machine de trouver une représentation plus significative de son contenu. Cette représentation sera différente des représentations classiques qui utilisent des descripteurs, tel que la texture, la couleur, la forme, etc. et ce, afin de pouvoir effectuer des recherches plus correctes.

En premier lieu, nous explorerons les réseaux à convolutions et leurs potentiel à générer des représentations complexes et multiples des images. Puis nous aborderons un autre concept de l'apprentissage profond qui sont les Deep Autoencoders qui vont nous permettre d'unifier les représentations des images. Enfin, nous introduirons des descripteurs de textures aux représentations pour voir si cela peut apporter des améliorations dans les résultats de recherche.

Comme les méthodes d'apprentissage profond sont des méthodes d'apprentissage automatique. L'une des complications à laquelle nous allons faire face est d'essayer de comprendre et d'interpréter les résultats obtenus. Nous chercherons à trouver un support théorique à ce que l'apprentissage (qui a toujours été perçu comme une boîte noire) accompli. Nous considérons ceci comme étant l'un des objectifs principaux de notre travail.

Nous essayerons, enfin, de présenter quelques perspectives et idées qui pourront faire l'objet d'autres recherches et de permettre l'obtention de meilleurs résultats.

Chapitre 1

Recherche d'image par le contenu

1.1 Introduction

De nos jours, un grand nombre d'images est généré et transmis sous format numérique sur Internet, et il existe une variété d'outils permettant d'accéder à ces images numériques. Cependant, lors de la recherche d'une image à partir d'Internet en utilisant un moteur de recherche d'images (par exemple : Google [Figure 1.1]), les résultats récupérés ne répondent pas forcément au besoin de l'utilisateur. En effet, les images sont retrouvées en se basant sur les métadonnées associées comme les mots clés, les titres, etc.



FIGURE 1.1 – Recherche d'image de Google.

La recherche d'image par texte (Text Based Image Retrieval - TBIR) [Figure 1.2] dépend de la qualité des annotations d'une image qui doivent couvrir tous les termes désirés, cette tâche peut être donc difficile, coûteuse ou même encore infaisable quand la collection d'images est trop large.

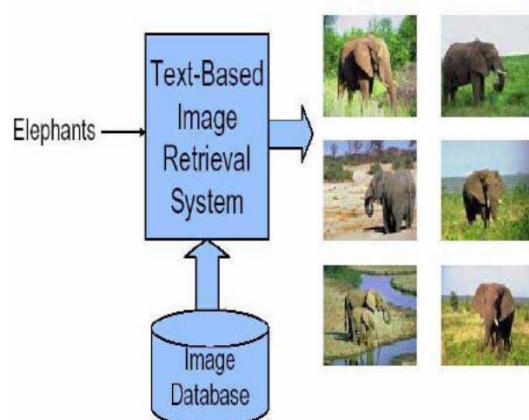


FIGURE 1.2 – Recherche d'image par texte. [Shr et al. 13]

La recherche d'image par le contenu (Content Based Image Retrieval - CBIR) [Figure 1.3] propose des techniques pour éviter l'utilisation des descriptions textuelles, et permet de récupérer les images pertinentes à partir d'une grande collection de base d'image. Les techniques de récupération d'images se basent sur certaines caractéristiques récupérées du contenu de l'image, comme la texture, la couleur, la forme, etc.

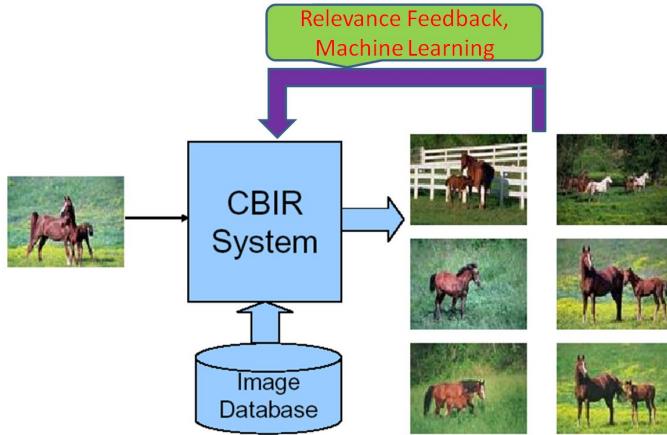


FIGURE 1.3 – Recherche d'image par le contenu. [Shr et al. 13]

Pour réaliser un système de recherche d'image par le contenu, des techniques basées sur le contenu des images ont été développées, et qui font appel à deux domaines de recherches :

Gestion de base de données : plus le nombre d'images dans une base est grand et plus il est nécessaire d'optimiser les méthodes de stockages, l'indexation multidimensionnelle des images et le temps de réponse à une requête.

Vision artificielle : ce domaine propose des techniques et approches pour la représentation et extraction des caractéristiques du contenu des images.

1.2 La conception d'un système CBIR

Les systèmes de recherche d'images par le contenu sont généralement basés sur des caractéristiques ou descripteurs d'images, par exemple :

- **Descripteur de couleur** : grâce à leurs représentations compactes et de faible complexité, la comparaison directe des histogrammes de couleur est communément utilisée.
- **Descripteur de forme** : descripteur de Fourier et moment invariant.
- **Descripteur de texture** : ces techniques permettent de donner une description des textures sous forme de mesures d'énergie, de contraste, d'homogénéité, etc. comme les matrices de co-occurrence (GLCM)[Har 79], ou bien sous forme d'histogramme comme le Local Binary Pattern (LBP) [Oja et al. 02].
- **Descripteur local** : l'information numérique est dérivée de l'analyse locale d'une image caractérisant son contenu visuel de la façon la plus indépendante possible de l'échelle, du cadrage, de l'angle d'observation et de l'exposition. C'est des descripteurs assez récents tel que Scale-invariant feature transform (SIFT) [Low 99].

1.3 Calcul de similarité

Généralement, des fonctions rigides de calcul de distance comme la distance euclidienne ou la similarité basée sur le cosinus sont utilisées pour la recherche de ressemblance entre les

caractéristiques extraites.

Par contre, ces fonctions de calcul de distance et de similarité ne sont pas toujours optimales pour la tâche complexe de recherche d'images par le contenu, cela est dû au problème du fossé sémantique (Semantic Gap) entre les caractéristiques bas niveau extraites par l'ordinateur, et le haut niveau de perception humaine.

Récemment, de grands efforts ont été fournis dans la recherche de différentes méthodes de calcul de similarité en utilisant les techniques de l'apprentissage automatique. Parmi ces dernières, des travaux se sont concentrés sur l'apprentissage de hachage. Des propositions ont été faites pour des méthodes d'apprentissage de mappage de données de hautes dimensions vers des codes binaires qui préservent la similarité sémantique. [Sin et al. 15]

1.4 Problème du faussé sémantique

Parmi les problèmes les plus difficiles qui font face à la recherche d'image par le contenu, nous trouvons le fossé sémantique. Il est défini comme la différence entre la faible niveau de représentation de l'information (y compris, mais pas seulement des images) dans l'ordinateur et la sémantique de haut niveau derrière elle (en d'autres mots, le sens qu'il doit représenter).

Récemment, de nouvelles approches pour résoudre ce problème, consistent en appliquant des techniques d'apprentissage automatique pour permettre à l'ordinateur de construire sa propre représentation hiérarchique d'une image, au lieu de calculer les caractéristiques manuellement.

1.5 Types de CBIR [Shr et al. 13]

Nous allons présenter quatre exemples d'approches conceptuelles différentes qui ont été développées pour les systèmes de recherche d'image par le contenu :

1.5.1 Basée-région

Netra [Ma et al. 99] et Blobworld [Car et al. 99] sont deux anciens systèmes de recherche d'images par le contenu basées région. Au moment de la recherche, le système fourni à l'utilisateur des régions segmentées de l'image requête, et il lui demande d'attribuer plusieurs propriétés, tel que les régions à mettre en correspondance, les caractéristiques des régions ou même le poids des différentes caractéristiques.

1.5.2 Basée-objets

Les systèmes de récupération d'image basées sur les objets retrouvent des images à partir d'une base de données en se basant sur l'apparence des objets physiques dans ces images. Ces objets peuvent être des éléphants, des panneaux d'arrêt, des hélicoptères, des bâtiments, des visages ou tout autre objet que l'utilisateur souhaite trouver. Une façon courante pour rechercher des objets dans les images est d'abord de segmenter toutes les images dans la base de données puis de comparer chaque région segmentée avec une région de l'image requête présentée par l'utilisateur. Ces systèmes de récupération d'image donnent de très bons résultats pour les images qui contiennent des objets qui peuvent être facilement séparés de l'arrière-plan, et qui ont des couleurs distinctives ou même des textures.

1.5.3 Basée-exemples

Appelée aussi recherche d'image par le contenu basée-requête, nous nous intéresserons dans nos approches à cette catégorie qui sera détaillée dans le troisième chapitre. Les utilisateurs donnent une image requête, ou une partie d'une image, le système l'utilisera comme base pour sa recherche. Il devra trouver à l'aide de caractéristiques extraites du contenu de l'image, toutes les images semblables qui partagent ce même contenu.

1.5.4 Basée-Feedback

Pour rechercher une image, le système affiche à l'utilisateur un échantillon de photos et demande à l'utilisateur de les classer. En utilisant ce classement, le système ré-exécute les requêtes et répète cela jusqu'à ce que la bonne image soit trouvée.

1.6 Systèmes de recherche d'images existants

Les systèmes de recherche d'image par le contenu ont été utilisés dans un grand nombre d'applications telles que : le diagnostic médical [Figure 1.4], les archives photographiques, l'identification des empreintes digitales et la reconnaissance faciale.

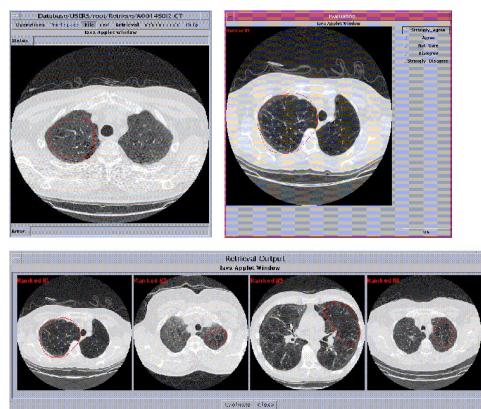


FIGURE 1.4 – Application Médicale. [Yod 08]

Nous allons maintenant présenter quelques systèmes connus de recherche d'image par le contenu :

QBIC

Query By Image Content (QBIC) [Nib et al. 93][Fli et al. 95], signifie requête par contenu d'image, il est le premier système de recherche d'images par le contenu commercial. Il fournit des framework et des techniques de base pour de nombreux systèmes de récupération d'images. QBIC supporte les requêtes basées sur des exemples d'images, d'images construites par l'utilisateur, des dessins et modèles de couleur et de texture, etc. Les fonctions de couleur utilisées dans QBIC sont (R, G, B), (Y, i, q), (L, a, b), MTM (transformée mathématique Munsell), et un histogramme de couleur à k-élément. Dans son nouveau système, la recherche de texte par mot-clé peut être combinée avec la recherche de similarité par contenu. La [Figure 1.5] montre une démo de QBIC.

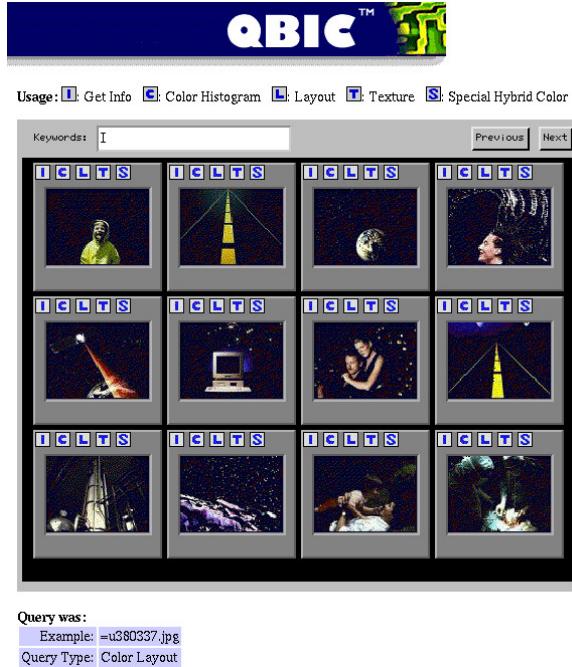


FIGURE 1.5 – Interface de la démo de QBIC.

LIRE

LIRE [Lux et al. 08][Lux et al. 13] est une bibliothèque Java qui fournit un moyen simple permettant de rechercher des photos et des images en fonction de leurs caractéristiques de couleur et de texture. LIRE crée un index de Lucene des caractéristiques de l'image pour la récupération de son contenu. Elle est facile à utiliser, des méthodes pour rechercher l'index et les résultats sont fournies par LIRE.

La bibliothèque LIRE et l'application de démo [Figure 1.6] ainsi que tous les fichiers sources de test et de développement sont disponibles sous la licence GNU GPL.

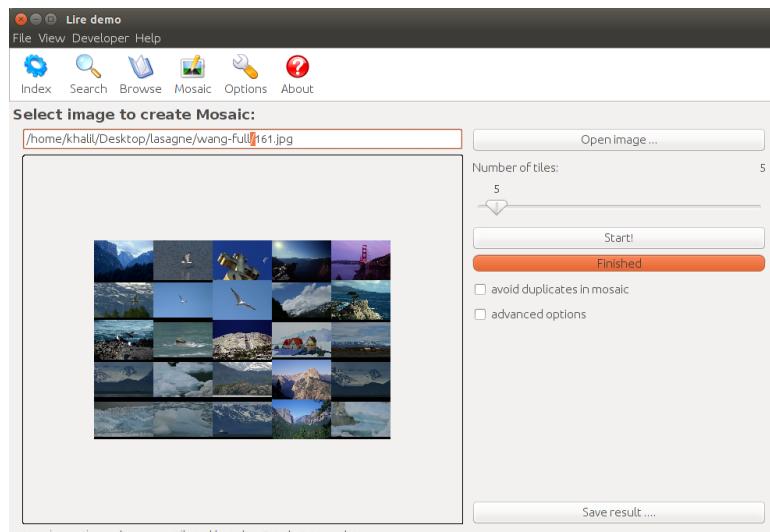


FIGURE 1.6 – Interface de la démo de LIRE.

FIRE

Flexible Image Retrieval Engine (FIRE) [Des et al. 08], c'est un système de recherche d'images par le contenu que Thomas Deselaers a développé avec d'autres chercheurs de The Human Language Technology and Pattern Recognition Group de RWTH Aachen University.

L'objectif principal de FIRE est d'étudier différents descripteurs d'image et d'évaluer leur performance. FIRE a été développé en C ++ et Python et est destiné à être facilement extensible. Une démo est disponible comme le montre la figure suivante :



FIGURE 1.7 – Interface de la démo de FIRE.

DigiKam

DigiKam est un logiciel gratuit et open-source organisateur d'image et éditeur de tag écrit en C++, c'est aussi une application de gestion de photos étendue construite avec des bibliothèques de KDE. Il offre, en plus d'autres nombreuses fonctionnalités : les recherches inversées pour les images de la collection locale, la détection de doublons et la recherche floue par des dessins.

1.7 Conclusion

Dans ce chapitre nous avons fait une introduction sur les systèmes de recherche d'image par le contenu, puis nous avons ensuite présenté quelques types de CBIR dont la recherche d'image par le contenu basée-exemple sur laquelle notre implémentation se base. On a conclu ce chapitre par des exemples de systèmes connus de recherche d'image par le contenu.

Nous allons présenter dans le prochain chapitre des techniques d'apprentissage automatique et principalement d'apprentissage profond pour essayer de proposer des solutions au problème du fossé sémantique des images.

Chapitre 2

Apprentissage profond

2.1 Introduction

Quelles sont les limites de l'intelligence des ordinateurs ? C'est une question toujours ouverte, même en étant l'une des toutes premières à être posées depuis l'invention des ordinateurs. Dans ce chapitre nous allons présenter l'une des techniques utilisées en intelligence artificielle à savoir l'apprentissage automatique qui permet de doter l'ordinateur d'une capacité d'apprendre et de s'adapter en fonction de la tâche qu'il doit accomplir.

Nous aborderons ensuite l'une des techniques les plus récentes de l'apprentissage automatique qui est l'apprentissage profond (Deep Learning), elle propose une nouvelle approche plus approfondie et plus ressemblante au comportement du cerveau.

2.2 Apprentissage automatique

On peut trouver dans un dictionnaire que l'apprentissage est : "La capacité d'acquérir et d'appliquer les connaissances". C'est une notion que l'être humain développe depuis son enfance. Question : et si les ordinateurs arriveraient à développer la capacité d'apprendre ?

La tâche de l'apprentissage automatique est de concevoir un algorithme d'apprentissage fiable pouvant être utilisé pour résoudre différents problèmes et être appliqué dans différents domaines. Son utilisation pourrait couvrir plusieurs tâches très distinctes telles que la prévision du marché boursier, la découverte de caractéristiques et motifs dans les données scientifiques ou même la reconnaissance d'objets dans les images.

En effet, à travers l'exploration du processus d'apprentissage du cerveau, les scientifiques ont fait l'hypothèse qu'il pourrait y avoir un algorithme d'apprentissage unique utilisé pour une variété de tâches. Cet algorithme unique pourrait donc à lui tout seul s'adapter à tous les problèmes qu'il rencontre.

Une expérience [figure 3.1] a été conçue en neuroscience au Département du MIT de Brain and Cognitive Sciences [Roe et al. 92], où ils ont coupé le lien entre l'oreille et le cortex auditif de furets, et lui ont relié les nerfs optiques (Le cortex auditif étant la partie du cerveau responsable des processus qui traitent l'information auditive capturé par l'oreille). Ils ont découvert que le cortex auditif a pu apprendre à traiter les données optiques. En d'autres termes, la partie du cerveau qui une fois a appris à entendre, maintenant apprend à voir.

Sensor representation in the brain

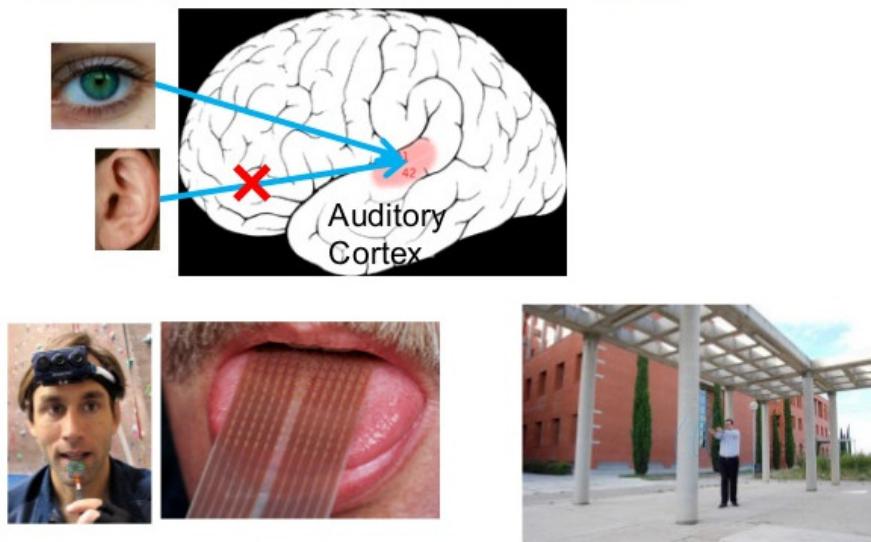


FIGURE 2.1 – Représentation des sensations dans le cerveau. [Roe et al. 92]

La même expérience a été reproduite avec succès à plusieurs reprises et avec des configurations différentes (différentes régions du cerveau, des espèces différentes, etc.) qui ont conduit aux mêmes résultats.

L'apprentissage automatique vise à concevoir ce genre d'algorithme, capable à s'adapter à toutes les situations possibles. L'objectif est de trouver des modèles dans les données utilisées pour l'apprentissage, et les généraliser pour construire des modèles mathématiques capables de faire des prédictions pour de nouvelles données.

2.3 Concepts de base de l'apprentissage automatique

D'après la définition de [Mit et al. 97] "Un programme informatique apprend d'une expérience E à effectuer une tâche T si sa performance P mesurée pour cette tâche T s'améliore avec l'expérience E ."

La description des notions d'expérience, tâche et performance comme donnée dans le livre [Goo et al. 16] est très claire et peut être vue comme suit :

2.3.1 Les tâches T

L'apprentissage automatique intervient dans des tâches où l'algorithme classique est incapable de donner des résultats satisfaisants, ou prendrait simplement trop de temps. Ces tâches peuvent être par exemple des tâches de :

Classification : Où le programme doit trouver la classe $c \in C$ à laquelle une entrée X appartient. En d'autres termes, on cherche à apprendre une fonction $F : \mathbb{R}^n \rightarrow C$ (C étant un ensemble discret $1..k$).

Classification avec manque de données : Elle rajoute un niveau de difficulté à la tâche de classification en supposant que certaines données peuvent être manquantes. Le classifieur devrait dans ce cas apprendre plusieurs fonctions avec différents sous ensembles des

attributs de l'entrée X .

Régression : Cette tâche ressemble à une classification, sauf que le résultat qui était la classe devient une valeur numérique, et l'espace de sortie devient continu. Nous cherchons donc à trouver une fonction $F : \mathbb{R}^n \rightarrow \mathbb{R}$.

Transcription : Elle ressemble encore à la classification, sauf que ce que nous cherchons en sortie au lieu d'être une classe, est une séquence de symboles. Comme par exemple, la traduction d'une phrase dans une autre langue.

Détection d'anomalie : Où le programme examine constamment une série d'événements ou de données d'entrée et signale s'il trouve quelque chose d'inhabituel. Une de ses applications est la détection de fraude dans l'utilisation de cartes de crédit.

Dé-bruitage : Permet d'apprendre à "nettoyer" une entrée X' qui pourrait être corrompue par un certain processus de corruption, en retournant l'entrée originale X .

2.3.2 Les mesures de performance P

La mesure de performance sert d'outil pour déterminer si un programme effectue bien la tâche T souhaitée. Elle diffère d'une tâche à une autre.

La mesure se fait sur les données dont on dispose et qu'on utilise pour l'apprentissage, mais aussi sur une partie de données mises de côté (non utilisées pour l'apprentissage). Le plus important étant de savoir si le programme retourne de bons résultats sur des instances qu'il n'a jamais rencontrées.

Plusieurs mesures de performance existent pour les tâches de classification, la mesure de précision étant généralement la plus utilisée. Mais pour d'autres tâches, le choix de la mesure peut être plus difficile à faire.

2.3.3 L'expérience E

L'expérience peut être définie par un scénario que le programme est sensé suivre durant son apprentissage, ce scénario est appelé un algorithme d'apprentissage classé en trois grandes catégories :

Non-supervisé : Le programme reçoit un ensemble de données (des exemples) qu'il se doit d'étudier pour satisfaire une tâche précise qui revient soit à minimiser un coût (pour augmenter la performance), soit à trouver des caractéristiques qui lient les attributs des exemples entre eux, afin de trouver une manière de regrouper ces derniers en catégories. Par exemple : clustering, PCA, etc.

Supervisé : Le résultat du traitement de chaque exemple est connu à l'avance. À chaque fois que le programme retourne une valeur de la fonction à apprendre, il compare cette dernière avec la valeur juste (connue), si elle est conforme le programme est sur la bonne voie, sinon des modifications sur la fonction apprise (paramètres appris) doivent être effectuées. Par exemple : Les réseaux de neurones, SVM, arbre de décision, etc.

Apprentissage par renforcement : c'est un type différent des deux premiers dans le sens où le programme est perçu comme un agent qui interagit dans un environnement,

et qui reçoit des feedbacks (réponses ou retours d'information) lui permettant d'établir une "politique" qui vise à maximiser ses "gains" ou "récompenses". Par exemple : Q-learning.

2.3.4 Exemple d'algorithme d'apprentissage : Les réseaux de neurones

Nous allons maintenant présenter l'une des techniques les plus utilisées pour l'apprentissage automatique supervisé (et aussi pour le non-supervisé), à savoir les réseaux de neurones (Perceptron Multicouche).

Un réseau de neurones artificiel essaye d'imiter le fonctionnement des réseaux de neurones biologiques. Appelé aussi perceptron multicouche, ces derniers sont comme leur nom l'indique, formés de plusieurs couches de neurones, la première est la couche d'entrée, ensuite viennent celles du milieu qui sont les couches cachées et à la fin une dernière couche appelée couche de sortie. La figure [Figure 2.2] ci-dessous montre un exemple d'un réseau de neurones à une seule couche cachée :

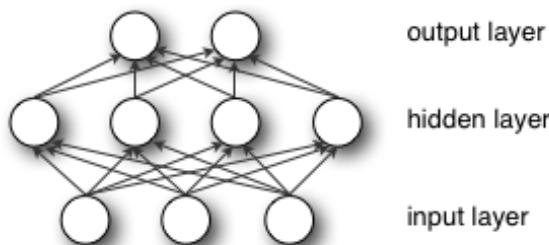


FIGURE 2.2 – Réseau de neurones (FC-MLP) à une seule couche cachée.

Un réseau de neurones fonctionne comme suit : La couche d'entrée reçoit d'abord une combinaison de valeurs qu'elle va transmettre à tous les neurones, de la couche cachée, qui la suivent. Chaque neurone de la couche cachée reçoit en entrée une combinaison linéaire d'entrées (vecteur de valeurs), la multiplie par le poids w du neurone, et y rajoute un biais b . Le résultat sera ensuite envoyé à une fonction d'activation s qui peut être par exemple la fonction $Sigmoide(x) = 1/(1 + e^x)$.

Une fonction d'activation de la couche de sortie G est choisie, cette dernière est choisie selon la tâche voulue (par exemple : régression, classification, etc.). Tout le processus est schématisé par un calcul matriciel. L'expression de la fonction f que le réseau de neurones essaye d'apprendre est comme suit :

$$f(x) = G(b^{(2)} + W^{(2)}(s(b^{(1)} + W^{(1)}x)))$$

Avec x vecteur d'entrée et $w1$, $b1$, $w2$ et $b2$ étant les matrices de poids et vecteur biais entre respectivement la couche d'entrée et la couche cachée, et entre la couche cachée et la couche de sortie. Ce sont ces paramètres qui seront appris.

Nous allons ensuite réaliser un apprentissage du réseau de neurones et donc essayer de se rapprocher de la fonction qu'on souhaite apprendre. Les paramètres à apprendre ($w1$, $w2$, $b1$ et $b2$) seront changés à chaque itération grâce à différents algorithmes, le plus utilisé étant l'algorithme rétro-propagation du gradient (backpropagation of errors). Ayant le résultat souhaité (réel) et le résultat retourné, l'idée est de définir une fonction "coût" (l'inverse de la performance) qui décrira à quel point la fonction f est éloignée de ce qui est recherché. Ensuite dériver f en fonction de chacun de ses paramètres, et changer ces derniers dans le sens qu'il faut (rajouter à sa valeur, ou en soustraire), dans le but de faire diminuer la valeur du coût.

Comme mentionné précédemment, l'apprentissage d'une tâche T est sensé permettre une application de ce qui a été acquis sur de nouvelles données, ceci est appelé la capacité de généralisation. La généralisation d'un apprentissage peut faire face à plusieurs problèmes [figure 2.3], les plus importants sont :

Le sur-apprentissage : Quand le programme retourne de bons résultats sur les données d'apprentissage, mais n'arrive pas à généraliser sur des données qu'il n'a jamais rencontrées auparavant.

Le sous-apprentissage : Dans le cas où le programme n'arrive même pas à trouver un modèle qui satisfait les données de l'apprentissage.

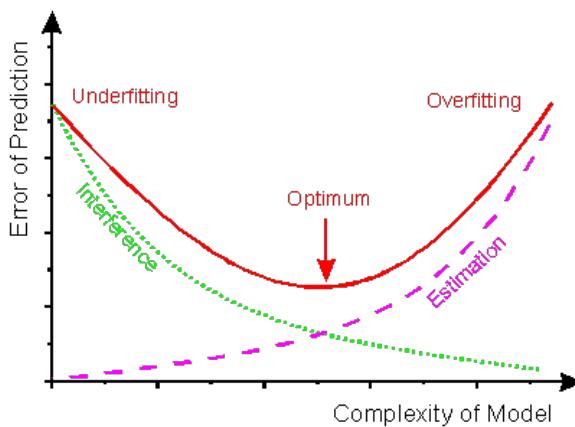


FIGURE 2.3 – Sur-apprentissage et sous-apprentissage. [Mar et al. 89]

2.4 Introduction à l'apprentissage profond

Comme nous l'avons mentionné, l'une des techniques les plus récentes en apprentissage automatique est l'apprentissage profond, elle est composée de deux mots : apprentissage et profond. La première partie se réfère au fait que ce soit un concept d'apprentissage automatique, et la deuxième partie : profond, se réfère à sa structure.

Principalement en raison de la faible puissance de calcul des machines, les algorithmes d'apprentissage n'étaient pas très complexes et puissants. Mais grâce à l'évolution exponentielle de la puissance de traitement (loi de Moore [Moo 65]) de nouvelles perspectives ont été mises en lumière. Depuis 2010, l'utilisation de la puissance des GPUs pour les calculs est devenue accessible pour le grand public, les scientifiques sont en mesure d'effectuer efficacement des calculs parallèles sur de très grosses matrices, et donc, ils ont réussi à élargir leurs modèles d'algorithme d'apprentissage pour atteindre de manière significative de meilleurs résultats et performances.

L'apprentissage profond permet de développer des approches d'apprentissage automatique de plus en plus profondes (plus volumineuses et donc plus complexes), mais il permet aussi une très grande flexibilité qui lui permet de combiner plusieurs techniques différentes. Pourtant, beaucoup de problèmes restent à régler, un des plus gros étant la soi-disant curse of dimensionality (la malédiction de la dimensionnalité). En fait, les données en entrée à un algorithme d'apprentissage sont de haute dimensionnalité. Une image par exemple, est une matrice de milliers, ou peut-être même de millions de pixels, chacun peut prendre des centaines de valeurs différentes (les valeurs RGB). Si nous prenons par exemple une image de 100×100 , le nombre de configurations distinctes qu'elle peut prendre est :

$$256^3 \times 100 \times 100 = 1677721600.$$

Cet espace de valeurs est évidemment trop grand pour pouvoir en couvrir même une fraction. Mais le problème serait plus à propos des fonctions de hautes variations que nous essayons de reproduire (fonctions d'apprentissage). En fait, si la fonction n'est pas trop complexe, même dans des dimensions élevées, quelques exemples pourraient être suffisants pour l'apprentissage de sa représentation, mais si la fonction cible a beaucoup de variations, nous aurons besoin en conséquence de plusieurs exemples d'apprentissage, comme le montre la figure [Figure 2.4] ci-dessous :

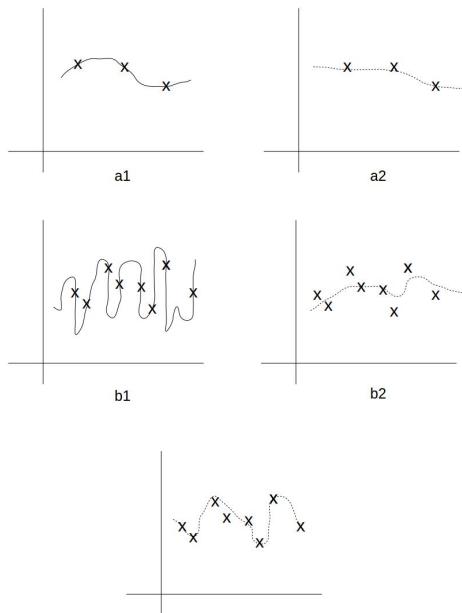


FIGURE 2.4 – Fonctions d'approximation.

Pour une fonction à deux dimensions (deux attributs), nous avons dans l'exemple (a) le graphe (a1) qui montre une fonction simple que le modèle cherche à approximer, et le graphe (a2) montre le résultat de l'approximation obtenu par le modèle. Ce résultat est satisfaisant malgré le nombre réduit d'exemples.

Quant à l'exemple (b), le graphe (b1) montre une fonction beaucoup plus complexe que celle du graphe (a1), la tâche d'approximation devient plus difficile. Les tentatives du modèle (représentées respectivement par les graphes (b2) et (b3)) de ne pas passer par tous les exemples, ou de passer par tous les exemples échouent à trouver un résultat satisfaisant dans les deux cas. On vise à trouver une approximation de la fonction qui passe par tous les exemples, et donc si on trouve une fonction qui passe par tous les exemples mais qui ne ressemble pas à la fonction qu'on veut approximer, on fera face à un problème de sur-apprentissage.

Il est clair que pour une dimension D plus grande de donnée en entrée (exemple : age, salaire, etc.) ou sur une variété de dimension D, le nombre de variations peut croître de façon exponentielle avec D , d'où résulte le grand nombre d'exemples requis. [Ben 14]

2.5 Les architectures de l'apprentissage profond

Bien que le principe des algorithmes d'apprentissage existants est unique (en consistant à passer par un tas de données pour trouver des modèles significatifs, et arriver à construire des modèles mathématiques afin de réaliser une tâche d'apprentissage, permettant de s'adapter à des objectifs différents). Plusieurs concepts et architectures sont appliqués pour l'apprentissage profond, parmi eux :

2.5.1 Deep Belief Networks

Réseaux profond de croyance (DBN) est une pile de machines de Boltzmann restreintes (Restricted Boltzmann Machines RBM) qui se termine généralement avec une unité de classification. Un RBM est un réseau de neurones à deux-couches (une visible et l'autre cachée) stochastique (l'activation de neurones est probabiliste). Les réseaux profonds de croyance sont utilisés pour le pré-apprentissage non-supervisé, dans le but d'obtenir une meilleure initialisation des poids du réseau.

2.5.2 Réseau de neurones à convolution

L'idée a été inspirée du travail de Hubel et Wesley sur le cortex visuel du chat [Hub et al. 68]. Un réseau de neurones à convolution ou Convolutional Neural Network (ConvNets), vise à émuler le réseau de neurones biologiques. La mise en œuvre la plus commune d'un réseau de neurones à convolutions se retrouve dans le réseau LeNet de [Lec et al. 98], il empile différentes couches de piles de convolution et de sous-échantillonnage, suivies à la fin de perceptrons multicouches entièrement connectées, comme indiqué sur la figure suivante :

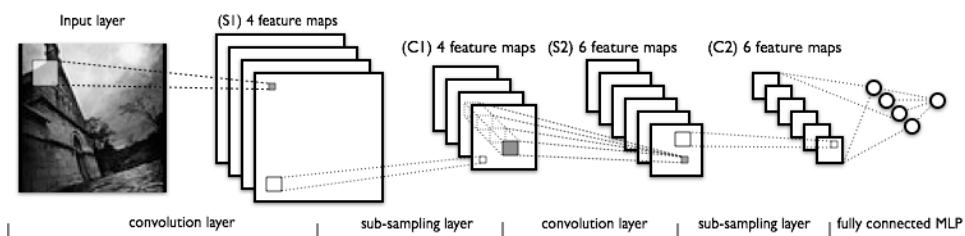


FIGURE 2.5 – Réseau à convolution LeNet. [Lec et al. 98]

Nous allons maintenant décrire des notions de bases qui peuvent être utilisées dans les couches d'un réseau de neurones à convolution, telles que le padding, sous-échantillonnage (pooling), normalisation et dropout.

La convolution

La convolution est un terme mathématique, elle est définie comme l'application d'une fonction de façon répétée à travers la sortie d'une autre fonction. Dans ce contexte, on applique un filtre sur une image en prenant en compte tous les décalages possibles. Comme le montre la figure suivante :

a ₀₀	a ₀₁	a ₀₂	a ₀₃	a ₀₄	a ₀₅	a ₀₆	a ₀₇	I
a ₁₀	a ₁₁	a ₁₂	a ₁₃	a ₁₄	a ₁₅	a ₁₆	a ₁₇	
a ₂₀	a ₂₁	a ₂₂	a ₂₃	a ₂₄	a ₂₅	a ₂₆	a ₂₇	
a ₃₀	a ₃₁	a ₃₂	a ₃₃	a ₃₄	a ₃₅	a ₃₆	a ₃₇	
a ₄₀	a ₄₁	a ₄₂	a ₄₃	a ₄₄	a ₄₅	a ₄₆	a ₄₇	
a ₅₀	a ₅₁	a ₅₂	a ₅₃	a ₅₄	a ₅₅	a ₅₆	a ₅₇	
a ₆₀	a ₆₁	a ₆₂	a ₆₃	a ₆₄	a ₆₅	a ₆₆	a ₆₇	
a ₇₀	a ₇₁	a ₇₂	a ₇₃	a ₇₄	a ₇₅	a ₇₆	a ₇₇	

M ₀₀	M ₀₁	M ₀₂		h
M ₁₀	M ₁₁	M ₁₂		
M ₂₀	M ₂₁	M ₂₂		

$$a_{33} = a_{22}M_{00} + a_{23}M_{01} + a_{24}M_{02} + a_{32}M_{10} + a_{33}M_{11} + a_{34}M_{12} + a_{42}M_{20} + a_{43}M_{21} + a_{44}M_{22}$$

FIGURE 2.6 – Exemple de convolution.

Soit une image I et un filtre de convolution h , appliquer un filtre de convolution sur un pixel, par exemple le pixel a_{33} , signifie donner une nouvelle valeur à ce pixel en appliquant le filtre sur ce pixel et ses voisins, comme le montre la formule de la figure.

Les couches de piles de convolution appliquent simplement une multitude de convolutions sur les couches du réseau en utilisant différents filtres.

Le padding

L’application d’une convolution ne peut pas se faire sur les pixels des bords de l’image, cela donc réduira la taille de l’image après qu’elle soit filtrée. Afin de pouvoir mieux contrôler la taille des images obtenues après l’application d’une convolution sur une image en entrée, on peut utiliser ce qu’on appelle "padding" (remplissage) avant l’application d’un filtre de convolution. L’une des techniques utilisées pour le padding est le zero-padding, pour cela il suffit de rajouter des zéros aux bords de l’image. Pour un zero-padding de 1, on a le résultat de la figure suivante :

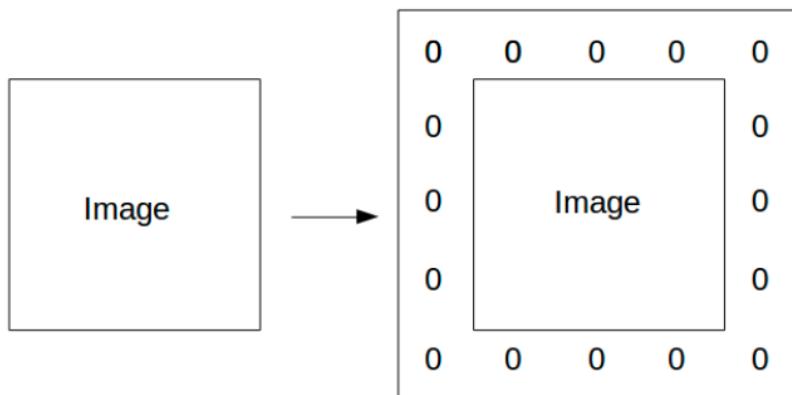


FIGURE 2.7 – Application d’un zero-padding sur une image

Nous pouvons aussi utiliser le zero-padding pour conserver exactement la taille spatiale d’une image après l’application d’une convolution, de sorte à ce que la largeur d’entrée et de sortie et la longueur restent les mêmes, comme nous le verrons dans le prochain chapitre. [CS231n 16]

Le sous-échantillonnage

Le sous-échantillonnage (subsampling) se réfère à la réduction de la taille globale d'un signal. Applicable dans de nombreux cas, comme la compression audio pour les fichiers de musique, le sous-échantillonnage se fait simplement pour réaliser une réduction de la taille des données.

La méthode de sous-échantillonnage spécifique proposé par LeCun dans LeNet est connue comme max-pooling. Cela implique le fractionnement d'une matrice (image) en de petits fragments qui ne se chevauchent pas (plus les fragments sont grands, plus la réduction est importante), et de prendre la valeur maximale dans chaque grille en tant que valeur dans la matrice réduite. Ceci permet, entre autres, l'obtention d'une invariance aux translations. La figure suivante montre un exemple de l'application d'un max-pooling :

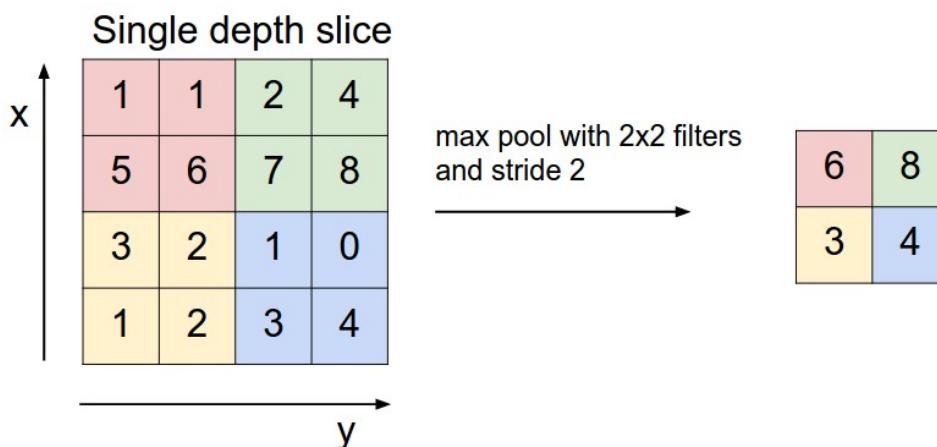


FIGURE 2.8 – Exemple de max-pooling.

Normalisation

Une des techniques utilisée, inspirée de la neuroscience computationnelle, est l'ajout de couches de normalisation après une couche de convolution [Kri et al.,12]. Néanmoins, l'application d'une normalisation sur ces couches semble avoir un impact très minime et n'est plus utilisée. Ils ont été dominés par d'autres techniques telles que le "dropout" ainsi qu'une meilleure initialisation des poids des neurones et les algorithmes d'entraînement du réseau. [CS231n 16]

Perceptron multicouche entièrement connecté

A la fin du réseau, après plusieurs couches de mise en commun de convolution et de max-pooling, le traitement final dans le ConvNets se fait via des couches de perceptrons entièrement connectés. Une couche entièrement connectée prend tous les neurones de la couche précédente et les relie à chaque neurone dont elle dispose.

Dropout

On applique un dropout lors de la phase d'entraînement d'un réseau de perceptron multicouche entièrement connecté (FC-MLP), précisément lors de la rétro-propagation. Son but est d'éviter le sur-apprentissage, par exemple pour un taux de dropout de 30%, la rétro-propagation ne s'appliquera pas sur 30% des neurones d'une couche (ces neurones sont choisis aléatoirement à chaque rétro-propagation). Dans ce cas les neurones d'une même couche

n'auront pas rencontré (n'apprendront pas) les même exemples (images). Cela procurera au réseau une plus grande capacité de généralisation.

2.5.3 Deep AutoEncoders

Les Autoencoders sont l'un des concepts introduits par l'apprentissage profond. Nous avons déjà parlé du problème de curse of dimentionality, les autoencoders proposent un moyen de résoudre ce problème. Pour cela, ils visent à trouver une représentation compressée des données en entrée.

En d'autres termes, les autoencoders sont un type de réseaux de neurones entraînés d'une manière supervisée par rétro-propagation pour reproduire leurs données en entrée après les avoir fait passer à travers des couches cachées de dimension inférieure à celle de l'entrée, nous donnerons plus de détails sur le fonctionnement d'un autoencoder dans le prochain chapitre.

Un "deep autoencoder" est construit en empilant un certain nombre de couches de machines de Boltzmann restreintes RBM pré-entraînées (formant un réseau profond de croyance DBN). On applique à la fin sur ces couches un ajustement ou une mise au point (fine tuning) avec de la rétro-propagation (backprop). La figure suivante montre un exemple d'autoencoder :

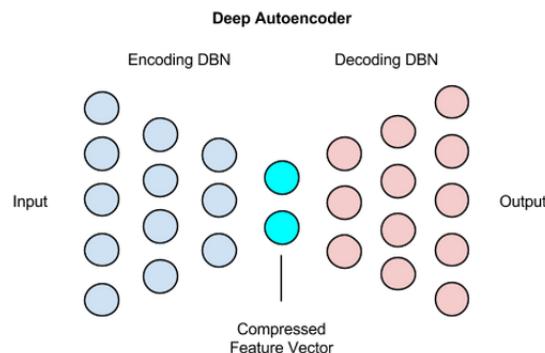


FIGURE 2.9 – Exemple d'un modèle d'autoencoder utilisant les DBN.[Site1]

Un exemple d'utilisation d'un autoencoder est la recherche d'une représentation de dimension inférieure pour une image. La raison pour laquelle cette représentation pourrait exister est donnée par l'idée suivante : Si vous essayez de générer une image aléatoire, vous vous retrouverez très probablement avec une image comme la figure suivante :

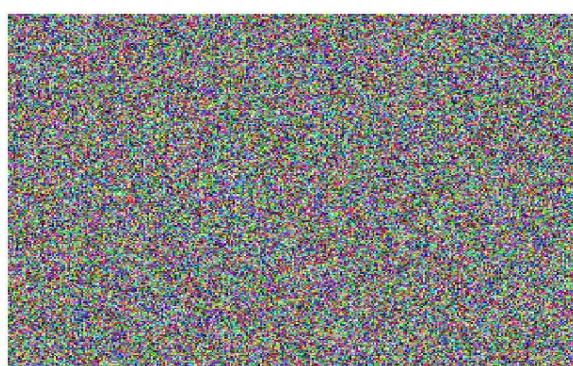


FIGURE 2.10 – Image générée aléatoirement.

Peu importe combien de fois vous essayerez, vous n'obtiendrez probablement jamais une image d'un chien, d'une voiture, ou une autre chose significative. Ainsi, l'espace des images

significatives forme seulement une fraction de l'espace total défini par les valeurs des pixels. L'objectif de l'autoencoder est alors de trouver une représentation significative d'une image dans cet espace.

2.6 Les exploits des techniques d'apprentissage profond [Site2]

L'Apprentissage profond a surperformé les techniques de vision artificielle traditionnelles au cours des dernières années. Dans le concours ImageNet en 2010 (ImageNet est un projet qui a pour but de fournir aux chercheurs une très grande base d'images facilement accessible), le meilleur algorithme de vision artificielle traditionnelle avait obtenu un taux d'erreur de 28,2%, ce qui signifie qu'il a obtenu de 71,8% d'images correctes. En 2011, le meilleur algorithme a obtenu un taux d'erreur de 25,8%.

En 2012, cependant, le premier processus d'apprentissage profond est entré dans la compétition et a dépassé en performance toutes les méthodes traditionnelles, en obtenant un taux d'erreur de 16,4%. Après cela, le premier algorithme d'apprentissage profond a ouvert les portes, la compétition a été inondée avec d'autres équipes d'apprentissage profond qui ont atteint un taux d'erreur de 11,7% en 2013 et 7.405% en 2014. Le vainqueur de l'édition 2014 a reconnu presque 93% des images, un énorme bond en avant surtout par rapport au taux de 72% atteint par les techniques de vision artificielle traditionnelles en 2010. La figure suivante montre les résultats des dernières années du concours d'ImageNet :



FIGURE 2.11 – Résultats du concours d'ImageNet. [Site2]

En voyant ces résultats phénoménaux, beaucoup de laboratoires se sont convertis vers ces approches et y consacrés une grande partie de leurs efforts. On peut citer les laboratoires de grandes université tel que : l'université de Berkeley, Stanford, Bonn, Berlin, Oxford, Montréal, Toronto. Mais aussi des laboratoires de grande entreprises tel que : Facebook AI Research, Google Research, Twitter's Deep Learning Group, Microsoft Research.

L'Intérêt croissant des entreprises au domaine de l'apprentissage profond fait naître des initiatives et des prototypes pour des produits plus intéressants les uns que les autres, et pousse donc la recherche. Parmi ces produit on peut noter les véhicules autonomes avec **Google Self-Driving Car**, mais aussi le lancement du **Tesla Autopilot** sur les nouvelles

voitures Tesla depuis déjà quelques mois. Et enfin les testes en échelle réelle d'un convois d'une douzaine de grands camions auto-pilotés des grands constructeurs Volvo, Daimler, Iveco, MAN, DAF et Scania. Le convoi a parcouru 2000km à travers l'Europe et fut un grand succès. La figure suivante montre quelque exemple des Self-Driving cars :



FIGURE 2.12 – Respectivement de gauche à droite, et de haut en bas : **Google Self-Driving Car** ; **Tesla Autopilot** ; Le convoi des camions auto-pilotés ; le parcours des camions à travers l'Europe.

2.7 Conclusion

Nous avons présenté dans ce chapitre la notion d'apprentissage automatique et sur quelle théorie elle se base, et comme exemple nous avons décrit les réseaux de neurones. L'apprentissage profond est l'une des techniques d'apprentissage automatique les plus récentes, nous avons ensuite fait une introduction sur les différents concepts d'apprentissage profond les plus utilisés.

Nous nous intéresserons dans le prochain chapitre à notre implémentation d'un système de recherche d'images par le contenu en utilisant des concepts d'apprentissage profond, qui sont les réseaux à convolutions et les Deep Autoencoders. Nous effectuerons des mesures de performances pour comparer toutes nos approches, et nous donnerons à chaque fois nos interprétations des résultats obtenus.

Chapitre 3

Représentation sémantique d'images

3.1 Introduction

Dans ce chapitre nous allons présenter nos différentes approches pour la réalisation d'une recherche d'image par le contenu (CBIR) basée exemple. Contrairement aux techniques traditionnelles qui utilisent les formes, textures, couleurs, etc. pour comparer les images (calculer la ressemblance), nous allons utiliser dans nos approches des techniques de l'apprentissage profond. Parmi les techniques d'apprentissage profond auxquelles nous allons nous intéresser se trouvent les réseaux à convolution (ConvNets) et les Deep AutoEncoders.

Nos premières tentatives consisteront en l'exploration de différentes méthodes permettant d'extraire la description d'une image à partir d'un réseau à convolution. Puis nous allons essayer réduire cette description vers un espace de représentation plus petit à l'aide des Deep Autoencoders, et observer ce que cette réduction apporte en terme de performance. Enfin, nous introduirons des descripteurs de texture à nos descriptions extraites par la technique d'apprentissage profond et constater si cela apporte des améliorations dans l'extraction des images ressemblantes. Nous allons aussi donner pour chacune de nos approches, des interprétations bien fondées des différents résultats que nous avons obtenus, ainsi qu'une étude comparative avec des systèmes connus de recherche d'images par le contenu qui utilisent les descripteurs d'images traditionnels.

3.2 Recherche d'images par le contenu basée exemple

Parmi les types de la recherche d'images par le contenu mentionnés au chapitre 1, nous avons choisi celui basée-exemple pour sa facilité d'implémentation. Notre algorithme de recherche d'image par le contenu basée exemple nécessite une grande base d'images, pour cela nous avons utilisé deux bases différentes, la première est celle de WANG [Wan et al.,01][Li et Wan.,03] contenant 1000 images (10 classes de 100 images) [Figure 3.1].



FIGURE 3.1 – Exemples d'image de la base WANG.

La deuxième est Caltech101 (100 classes) [Fei et al. 04] contenant plus de 9000 images

[Figure 3.2]. Nous avons choisi ces deux bases d’images pour la simple raison que les images sont déjà classées (une unique classe par image), ce qui facilitera le calcul des mesures de performance, on pourra donc vérifier facilement si deux images appartiennent à la même classe.



FIGURE 3.2 – Exemples d’image de la base Catlech-101.

Afin de comparer nos différentes approches, nous allons calculer pour chacune les mesures de performance obtenues. Pour cela nous allons appliquer notre algorithme de recherche d’image par le contenu basée-exemple comme suit :

Fonction $\text{CBIR}(\text{Base} : \text{Base d’images}, \text{Comp} : \text{Approche de Comparaison})$:
Mesures de performance

1. Diviser Base en deux parties :
 - a Query : Extraire aléatoirement de chaque classe de Base 20% d’images.
 - b Test : Les 80% restantes.
 2. Pour chaque classe C de Query faire
 - (a) Pour chaque image $Q \in C$ de Query faire
 - a Pour chaque image T de Test faire
 - | Calculer la ressemblance R entre Q et T avec l’approche Comp .**Fin Pour**
 - b $\text{Resemb} =$ liste des images de Test triées selon R par ordre décroissant.
 - c Comparer la classe C de Q avec les classes des images triées Resemb
 - d Calculer les mesures de performance Perf .**Fin Pour**
 - (b) Calculer la moyenne des mesures de performance de C .**Fin Pour**
 3. Retourner la moyenne des mesures de performance Totale.
- Fin**

Algorithme 1 – Recherche d’images par le contenu basée exemple

La première étape de notre algorithme est de diviser une base contenant plusieurs images en deux parties : Nous appellerons la première *Query*, contenant 20% des images (choisies aléatoirement) qui seront utilisées en tant que requêtes. Les 80% des images restantes seront dans la deuxième partie que nous appellerons *Test*.

Notre but étant que pour chaque image requête de *Query*, nous allons trouver les images qui lui ressemblent le plus de la partie *Test*. Nos méthodes de comparaison (calcul de ressemblance) entre les images ce feront avec plusieurs approches que nous allons présenter plus bas.

En ayant la liste des images les plus ressemblantes à une requête donnée triée par ordre décroissant (de l'image la plus ressemblante à la moins ressemblante), et pour comparer nos différentes approches que nous avons réalisé, nous devons calculer les mesures de performance qui sont comme suit :

1. **Précision à 1** : 1-Précision, elle est égale à 100% si l'image la plus ressemblante à la requête qui est retournée par l'approche est effectivement de la même classe que l'image en entrée, et 0% dans le cas contraire.
2. **Précision à N** : ou N-Précision N étant un nombre d'images (5,10,20,40 ou 60). Cette mesure donne le pourcentage d'images pertinentes (qui appartiennent à la même classe que l'image requête), parmi les N images les plus ressemblantes retrouvées par l'algorithme.

$$N - \text{Précision} = \frac{\text{Nombre des images pertinentes retrouvées}}{\text{Nombre } N \text{ des images retrouvées}}$$

3. **Rappel** : C'est le pourcentage d'images pertinentes, parmi le nombre total des images pertinentes (les images qui appartiennent réellement à cette classe).

$$R = \frac{\text{Nombre des images pertinentes retrouvées}}{\text{Nombre total des images pertinentes}}$$

4. **Taux d'erreur** : C'est l'événement contraire de la 1-Précision.

$$\text{Taux d'erreur} = 1 - \text{1-Précision}$$

Contrairement à la base d'images WANG, nous avons remarqué que les classes de la base Caltech101 (101 classes) ont un nombre d'images qui est différent entre chaque classe, nous allons donc d'abord calculer les moyennes de performance d'une classe donnée, ensuite calculer la moyenne des performances totale de l'approche. Comme ça les classes ayant un nombre très grands (très petits) d'images, ne seront pas plus avantagées (moins avantagées) par rapport aux autres classes.

3.3 Architecture du réseau à convolution

Comme nous l'avons cité précédemment, notre approche utilisera des techniques d'apprentissage profond pour la comparaison des images à la place des techniques traditionnelles (texture, forme, couleur, etc.). Pour cela, nous avons utilisé les réseaux à convolution (ConvNets) introduits dans le chapitre précédent.

L'architecture VGG-CNN-S du réseau à convolution que nous avons utilisé dans notre algorithme, a été proposée par l'équipe de recherche Visual Geometry Group Department of Engineering Science, Université d'Oxford (VGG) [Cha et al.14]. Lors de la compétition ImageNet 2014 (introduite au chapitre 2) ils ont pu obtenir une erreur de classification (top-5) de 7.405%. L'architecture VGG-CNN-S du ConvNets a été entraînée sur une base d'images

très grande ILSVRC-2012 (plus de 1.2 millions d'images [Rus et al.15]). Cette base contient 1000 classes différentes, elle inclut une très grande variété d'images et a donc beaucoup de potentiel pour être généralisable, c'est-à-dire elle peut donner aussi de bons résultats sur d'autres bases d'images qui sont plus petites que celles de ILSVRC-2012.

Nous allons donc commencer par décrire l'architecture VGG-CNN-S du réseau à convolution que nous avons utilisée. Le tableau ci-dessous donne une descriptions de chacune des couches de l'architecture du ConvNets utilisé qui est comme suit :

ConvL	ConvL	ConvL	ConvL	ConvL	FC-MLP	FC-MLP	FC-MLP
96x7x7 st. 2,pad 0 LRN,x3 pool	256x5x5 st. 1,pad 1 x2 pool	512x3x3 st. 1,pad 1 -	512x3x3 st. 1,pad 1 -	512x3x3 st. 1,pad 1 x3 pool	4096 drop-out	4096 drop-out	1000 soft-max

TABLE 3.1 – Architecture de VGG-CNN-S [Cha et al.14].

Légende : ConvL Couche de pile de convolutions
FC-MLP Perceptron multicouche entièrement connecté

Ce réseau à convolution comporte une couche d'entrée (non incluse dans le tableau), 7 couches cachées et une couche de sortie (la dernière colonne du tableau). Les données en entrée au réseau à convolution sont des images ayant une taille fixe de 224 x 224 pixels RGB. Pour forcer cette taille à toutes les images, elles seront d'abord redimensionnées de telle sorte à ce que la plus petite des deux dimensions (longueur ou bien largeur) soit égale à 256 pixels, ensuite on prendra les 224 x 224 pixels qui sont au centre de ces images. Le choix de la valeur de 224 pixels est nécessaire pour que les données en entrée s'adaptent à l'architecture proposée. Le seul pré-traitement effectué est la soustraction de la valeur moyenne RGB de chaque pixel de l'image, son interprétation géométrique est de centrer le nuage de données autour d'une origine [CS231n 16].

L'image passe premièrement à travers des piles de couches de convolution où nous appliquons plusieurs filtres. La première couche de convolution applique 96 filtres différents de taille 7x7 pixels avec un stride de 2 et sans zero-padding, comme le montre la figure suivante :

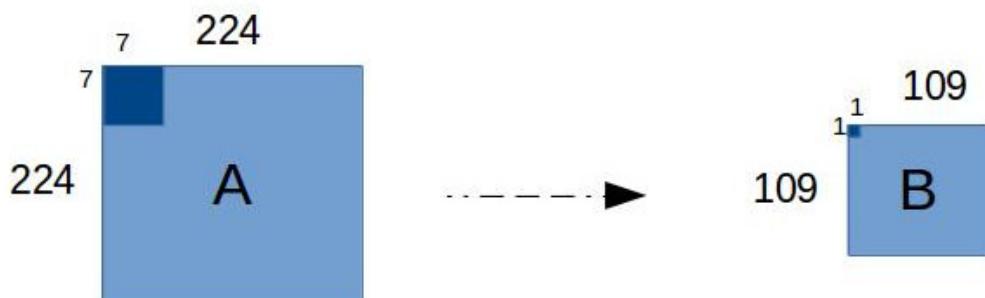


FIGURE 3.3 – Application d'un filtre de convolution.

La valeur du stride représente le nombre de pixels de déplacement d'un filtre sur une image après chaque opération de convolution. Le passage d'une image à travers une pile de convolution aura pour effet de changer la dimension (nombre de pixels) de l'image. Posons S la valeur du stride, P la valeur du zero-padding, W_1 le nombre de pixels d'une dimension

(longueur ou bien largeur) d'une image et F la taille des filtres de convolution. Pour calculer la taille obtenue en sortie W_2 , on applique la formule suivante :

$$W_2 = (W_1 - F + 2 * P)/S + 1$$

$$\text{Exemple de la première convolution : } (224 - 7 + 2 * 0)/1 + 1 = 109.5$$

Si on obtient un nombre non entier (avec une virgule) comme l'exemple ci-dessus, ce qui représente une opération de convolution incomplète, on ne prend alors que la partie entière. Comme les images en entrée ont la même largeur et longueur (images carrées), il suffit seulement de calculer cette valeur (nouvelle dimension) pour une seule dimension. Cette première couche de convolution est la seule couche sur laquelle on applique une normalisation [Kri et al.,12].

Après cela, on applique un max-pooling sur la sortie résultante des convolutions, comme le montre la figure suivante :

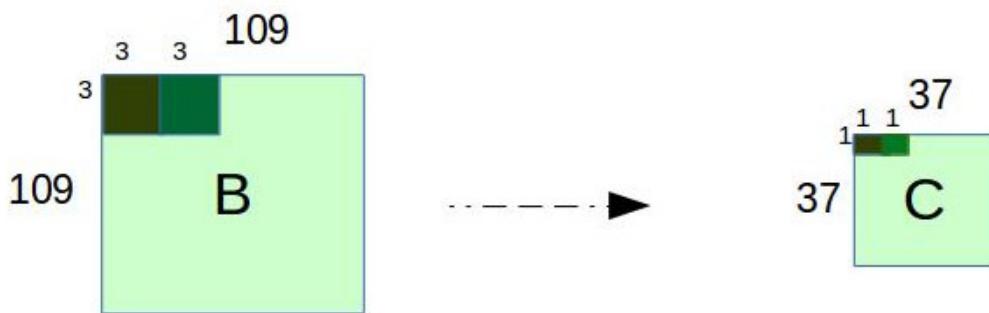


FIGURE 3.4 – Application d'un max-pooling.

En posant W_1 le nombre de pixels d'une dimension d'une image, F la taille du max-pooling et S la valeur du stride qui est égale à F . On peut calculer la taille du résultat W_2 , comme suit :

$$W_2 = (W_1 - F)/S + 1$$

Exemple du premier max-pooling :

$$(109 - 3)/3 + 1 = 36.33$$

Ces étapes sont presque les mêmes pour chaque couche de convolution, chacune applique des filtres de convolution d'une certaine taille, suivis ou non d'un max-pooling. On remarque sur la figure qui va suivre dans la troisième couche de convolution, que la taille de l'entrée est de 17x17 et de la sortie 17x17, comme on a dit dans le chapitre précédent cela est dû au zero-padding qui permet de contrôler la dimension des sorties. On remarque aussi que la plus petite taille des filtres de convolution est de 3x3 pixels, et cela pour capturer la notion de gauche / droite, haut / bas, au centre.

Les sorties obtenues après le passage d'une image sur ces couches de convolutions sont 512 matrices de taille 6x6 pixels, 512 étant le nombre de filtres appliqués lors de la dernière convolution, et la taille 6x6 pixels est la taille de la sortie de cette même couche après application des filtres de convolution et du max-pooling.

L'empilement des couches de convolution est suivi par trois couches (les 3 dernières couches du tableau) entièrement connectées (Fully-Connected Layers), pour faire passer en entrée les valeurs des sorties des convolutions on devra aplatisir les matrices obtenues précédemment (521 matrices de 6x6 pixels) en un seul vecteur unidimensionnel.

La figure suivante montre comment plusieurs matrices seront aplaties en un seul grand vecteur à une seule dimension :



FIGURE 3.5 – Applatissement des résultats du ConvNets.

Le vecteur unidimensionnel sera composé de la succession des lignes de la première matrice, ensuite les lignes de la deuxième matrice, et ainsi de suite jusqu'à la dernière matrice. Donc dans notre cas, à partir de 512 matrices de 6x6 nous aurons un vecteur unidimensionnel de taille : $512 * 6 * 6 = 18432$.

Les deux premières couches entièrement connectées ont 4096 neurones chacune, la troisième et dernière couche du réseau est la couche soft-max. Elle réalise la classification de la base d'image ILSVRC-2012 et contient donc 1000 neurones (un pour chaque classe), cette couche soft-max donnera comme sortie un vecteur de 1000 valeurs contenant les probabilités d'appartenance d'une image aux 1000 classes.

La figure suivante montre l'architecture complète contenant la taille de chaque entrée et sortie de chaque couche du réseau à convolution :

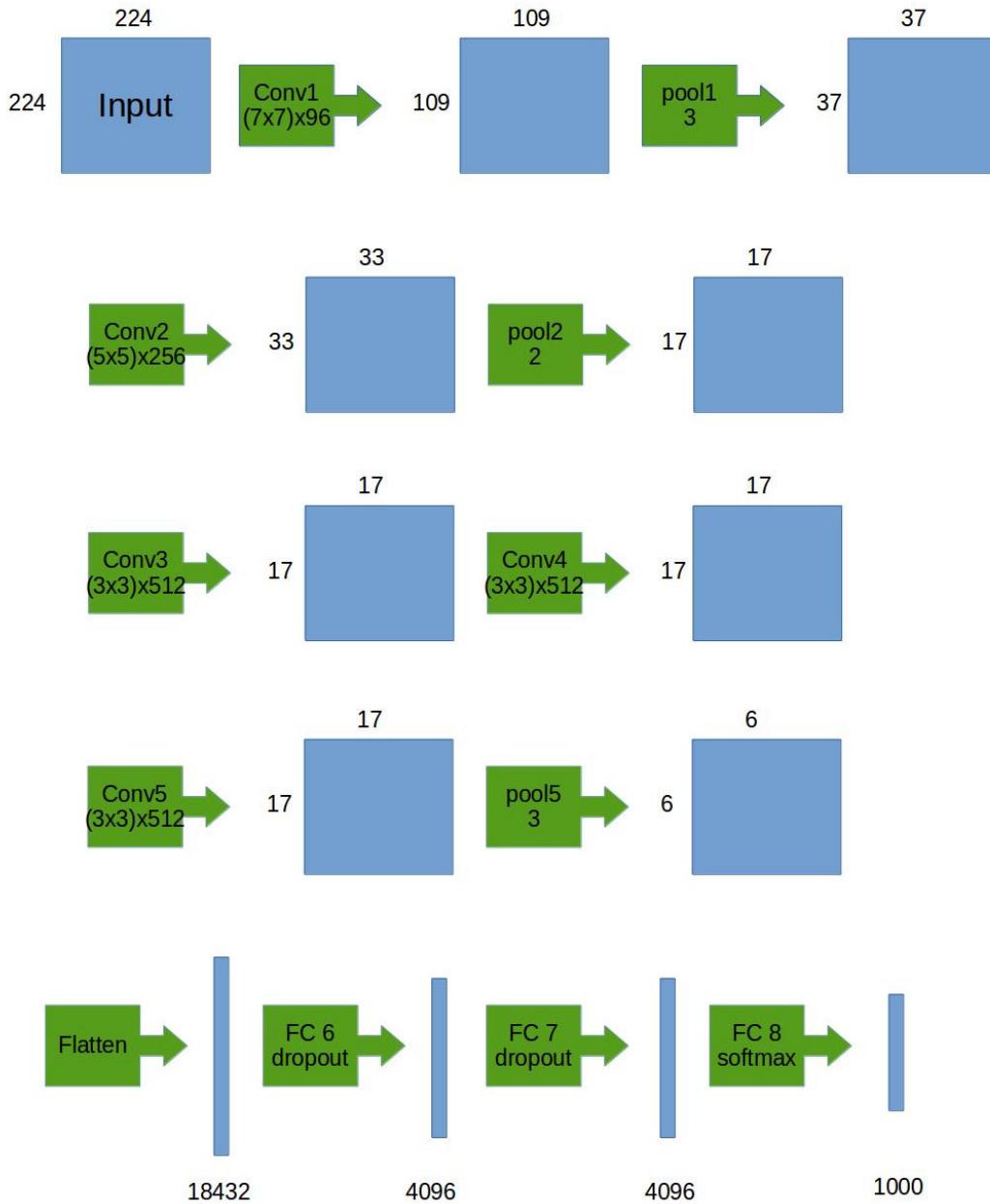


FIGURE 3.6 – Architecture complète du ConvNets.

3.4 Entraînement du réseau

Comme pour un réseau de neurones ordinaire, un réseau de neurones à convolution a besoin d'être entraîné sur une base d'exemples (base d'images dans notre cas), plus le réseau a une grande taille (plusieurs poids) plus on aura besoin d'un plus grand nombre d'exemples pour son entraînement.

La base utilisée pour l'entraînement (apprentissage) de ce réseau à convolution est ILSVRC-2012 (plus de 1.2 million d'images), et l'apprentissage s'est fait avec une descente du gradient stochastique avec moment. Les paramètres empiriques (hyper-paramètres) sont comme suit : Un moment de 0.9, une dégradation des pondérations (weight decay) de $5 * 10^{-4}$, et un taux d'apprentissage initial (learning rate) de 10^{-2} . [Cha et al.14]

Les valeurs initiales des poids ne peuvent pas être toutes égales à zéro (ou une autre

valeur), puisque tous les poids vont calculer le même gradient lors de la rétro-propagation, ce qui résultera en l'application des mêmes mis-à-jours sur tous les poids. En d'autres termes, il n'y aura pas d'asymétrie entre les valeurs des poids si elles sont toutes initialisées à une même valeur. [CS231n 16] Il serait donc plus judicieux d'initialiser les valeurs des poids à des valeurs aléatoires proches de zéro. Pour l'architecture VGG-CNN-S, les valeurs des poids de chaque couche ont été initialisées avec une distribution Gaussienne avec une moyenne de 0 et une variance de 10^{-2} . [Cha et al.14]

Il est évident que pour réaliser l'entraînement d'un aussi grand réseau requiert de très grands temps de calculs et une très grande puissance CPU/GPU de calculs. L'équipe Visual Geometry Group a utilisé pour cela une carte graphique NVIDIA GTX Titan, l'entraînement du réseau à convolution a duré pendant 3 semaines, sans inclure le temps de paramétrage des hyper-paramètres (moment, learning rate, etc.).

Comme dans notre cas, nous ne disposons pas d'une aussi grande puissance de calcul ni le temps nécessaire pour paramétrier le réseau à convolution, nous ne pouvons pas réaliser l'entraînement de ce réseau. L'équipe VGG a eu donc la courtoisie de publier pour les chercheurs les poids que leurs réseaux (VGG-CNN-S et autres) ont appris durant la phase entraînement, nous n'avons donc pas eu à refaire ce qui a été fait.

3.5 Extraction des caractéristiques

L'architecture du réseau à convolution utilisée contient une succession de piles de convolutions qui se termine par des couches entièrement connectées, nous allons nous intéresser à ces dernières dans nos approches de comparaisons pour notre algorithme de recherche d'image par le contenu.

Pour cela nous allons prendre chaque image et l'envoyer en entrée du réseau à convolution, et comme nous avons les différents poids de l'architecture VGG-CNN-S, nous pourrons ainsi récupérer les différentes valeurs que prend une image dans chaque couche du réseau à convolution. Comme les dernières couches du ConvNets sont des couches entièrement connectées, et donc seront sous le format d'une liste de valeurs réelles, nous pourrons manipuler les valeurs obtenues dans ces dernières couches. Le but étant de comparer une même couche de deux images différentes, la comparaison (calcul de ressemblance) se fait en calculant la distance (distance euclidienne) entre ces deux listes de nombres réels.

Nous allons maintenant présenter nos résultats obtenus (mesures de performance) de notre algorithme de recherche d'image par le contenu décrit au début de ce chapitre, en utilisant les trois dernières couches de l'architecture du réseau à convolution (les couches entièrement connectées) comme méthode de comparaison des images. Rappelons aussi que les bases utilisées dans notre algorithme sont les bases WANG et Caltech-101, nos premières approches sont donc comme suit :

3.5.1 Dernière couche Soft-max 1000

Pour notre première approche, nous avons utilisé pour comparer les images la dernière couche du réseau à convolution qui est la couche Soft-max, cette couche contient 1000 valeurs (une valeur pour chaque classe de ILSVRC-2012) rappelons que cette couche contient les probabilités d'appartenance d'une image aux 1000 classes. Les résultats que nous avons obtenu sont comme suit :

Mesure	1-Préc	5-Préc	10-Préc	20-Préc	40-Préc	60-Préc	Rappel
WANG	0.86	0.846	0.804	0.715	0.606	0.531	0.442
Caltech-101	0.551	0.512	0.479	0.431	-	-	0.321

TABLE 3.2 – Résultats obtenus avec la dernière couche de 1000 valeurs.

Nous remarquons que pour la base WANG nous avons un très bon taux de 1-Précision de 86% soit un taux d'erreur de 14% seulement. En d'autres termes notre algorithme de recherche d'image avec cette approche arrive à trouver dans 86% des cas que l'image la plus ressemblante à une image requête appartient à la même classe que la requête. Quand à la base Caltech-101 nous avons un taux de 55.1% de 1-Précision, rappelons que cette base contient beaucoup plus d'images et surtout plus de classes par rapport à la base WANG, ce qui diminuera la probabilité que l'algorithme arrive à trouver que l'image la plus ressemblante appartient à la même classe que l'image requête.

Globalement nous remarquons aussi que plus nous demandons un nombre plus grand d'images ressemblantes à une requête : top-1 avec la 1-Précision, top-5 avec la 5-Précision et ainsi de suite jusqu'au Rappel qui essaye de trouver exactement le nombre total d'images de la même classe, et plus notre algorithme a du mal à les trouver.

Ces résultats sont acceptables, sachant que les classes des base WANG (10 classes) et Caltech-101 (101 classes) ne sont pas incluent dans les 1000 classes de la base ILSVRC-2012 d'entraînement du réseau. Donc ce sera plus difficile à la couche Soft-max de trouver une approximation des classe des deux bases en utilisant les 1000 probabilité d'appartenances de ILSVRC-2012. On peut donc dire qu'avec cette couche, nous effectuons une recherche par étiquette (label) de classes de ILSVRC-2012.

3.5.2 Avant avant dernière couche 4096a

Dans cette deuxième approche nous intéresserons à l'avant avant dernière couche du réseau à convolution, c'est la première couche entièrement connecté qui se trouvent après les piles de convolution. Cette couche contient 4096 valeur, par conséquent on la notera comme la couche **4096a**. Contrairement à l'approche précédente, cette couche n'est pas une soft-max donc elle ne contient pas la classification de la base ILSVRC-2012 en 1000 classes, mais plutôt elle contient une représentation de l'aplatissement des 512 matrices 6x6 obtenues à partir des piles de convolutions (cumul des apprentissages des convolutions), et elle a été transformée en 4096 valeurs dans cette couche. En utilisant les 4096 valeurs de cette couche pour le calcul des ressemblance, nous avons obtenu les résultats suivants :

Mesure	1-Préc	5-Préc	10-Préc	20-Préc	40-Préc	60-Préc	Rappel
WANG	0.840	0.796	0.762	0.715	0.653	0.583	0.514
Caltech-101	0.654	0.521	0.454	0.371	-	-	0.257

TABLE 3.3 – Résultats obtenus avec la couche 4096a.

Nous remarquons que ces résultats sont assez différents de ceux obtenus avec la couche soft-max de 1000. Pour la base WANG : la 1-Précision, 5-Précision et 10-Précision ont diminué de 2% à 5%, mais à partir de 20-Précision jusqu'au Rappel nous avons une augmentation de reconnaissance, donc l'algorithme a plus de difficulté à trouver les images les plus ressemblantes, mais arrive à trouver mieux toutes les images ressemblante de la même classe.

Quant à la base Caltech-101 : c'est le contraire, il arrive mieux à retrouver les images les plus ressemblantes mais moins toutes les images ressemblantes de la même classe.

En résumé la couche 4096a ne donne pas forcément de meilleurs résultats que la couche soft-max 1000. Cela dépend du nombre de classes et d'exemples de chaque base d'images. On peut dire que contrairement à la recherche par étiquette effectuée dans la couche Soft-max 1000, nous effectuons avec la 4096a ce que nous appelons comme recherche sémantique. Sémantique parce-que elle compare les images grâce aux informations cumulées à partir des piles de convolutions.

3.5.3 Avant dernière couche 4096b

La couche qui sera utilisée pour notre troisième approche est l'avant dernière couche entièrement connectée du réseau à convolution. Elle contient 4096 valeurs et se trouvent entre la couche Soft-max 1000 et la couche 4096a, on la notera donc comme la couche **4096b**.

La couche 4096b ajoute un traitement plus avancé des connaissances acquises à partir de la couche 4096a qui la précède, et n'étant pas une Soft-max donc elle ne donne pas une représentation en classes, elle est par conséquent moins spécialisé que la dernière couche Soft-max 1000.

Mesure	1-Préc	5-Préc	10-Préc	20-Préc	40-Préc	60-Préc	Rappel
WANG	0.97	0.947	0.932	0.900	0.855	0.795	0.716
Caltech-101	0.773	0.681	0.627	0.540	-	-	0.405

TABLE 3.4 – Résultats obtenus avec la couche 4096b.

Les résultats des mesures de performances en utilisant cette couche pour la comparaison sont nettement meilleurs qu'en utilisant les couches des deux approches précédentes, pour les deux bases WANG et Caltech-101 nous avons une très grande amélioration dans chacune des N-Précisions ainsi que le Rappel. Ces améliorations varient de 10% jusqu'à plus de 20% d'augmentation.

Pour la base WANG nous avons une 1-Précision de 97%, l'algorithme avec à trouver presque dans tous les cas la bonne image la plus ressemblante. Quand aux résultats de la base Caltech-101, ils ne sont pas aussi bons que ceux de WANG mais nous obtenons de très bonnes améliorations sur toutes les mesures de performance. Nous remarquons donc que la couche 4096b est la meilleure couche pour comparer les images avec notre algorithme, mais nous pouvons dire aussi que c'est la meilleure couche qui permet de donner la meilleure description d'une image donnée, une description sémantique qui ne dépend pas de la classe de l'image mais de l'information sémantique de son contenu. Avec cette couche nous arrivons à mieux rassembler les informations sémantiques obtenus des convolutions et de la couche 4096a sans être spécialisée en classes comme la couche Soft-max 1000.

3.5.4 Résultats

Pour mieux visualiser les résultats obtenus des approches de l'extraction des caractéristiques et voir la différence entre elles, voici les deux graphes suivants qui montrent le récapitulatif des résultats obtenus avec les trois approches :

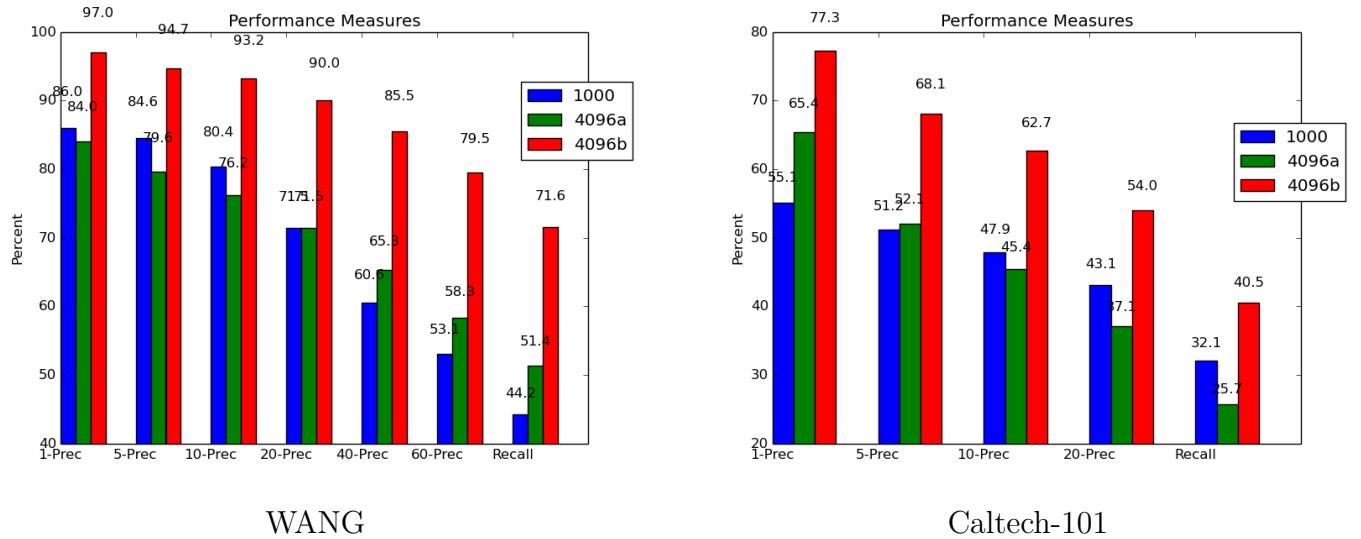


FIGURE 3.7 – Résultats obtenus sur les deux bases.

Il est clair que la représentation sémantique profonde de la couche 4096b dépasse de loin la représentation sémantique non profonde de la couche 4096a mais aussi la représentation en étiquette de la couche Soft-max 1000.

3.6 La description sémantique

Pour expliquer les résultats que nous avons obtenus dans les trois approches décrites précédemment, nous devons comprendre ce qu'il se passe dans le réseau à convolution. Jusqu'à dernièrement, il n'y avait pas beaucoup de preuves sur ce qui se passait durant l'apprentissage, ce qui n'était pas satisfaisant d'un point de vue scientifique.

L'approche de Matthew Zeiler [Zei et al. 14] nous a permis de trouver une théorie bien fondée pour expliquer l'apprentissage qu'effectuent les réseaux à convolution dans les parties contenant les piles de convolutions. Il montre dans sa recherche au fur et à mesure de la phase d'apprentissage, comment les valeurs des différents masques de convolutions changent et finissent à la fin par se spécialiser en apprenant à reconnaître un motif précis.

Les filtres des premières couches de convolutions apprennent à reconnaître des motifs très simples par exemple : des traits, des coins, etc. Les motifs deviennent de plus en plus complexe plus on avance à travers les couches de convolutions, jusqu'à arriver aux filtres de convolution des couches les profondes du réseau, qui apprennent à reconnaître des motifs beaucoup plus complexes en combinant les résultats des couches de convolution précédentes. Ces motifs complexes représenteront l'information sémantique que les réseaux à convolution reconnaissent. Ainsi, on trouve qu'un réseau à convolution imite finalement bien le comportement observé dans le cerveau.

3.6.1 Réseau à déconvolution

Matthew Zeiler réalise donc deux architectures, la première est un réseau à convolution qu'il a implémenté en se basant sur LeNet de LeCun [LeCun et al. 89] et AlexNet de Krizhevsky [Kri et al. 12], comme le montre la figure suivante :

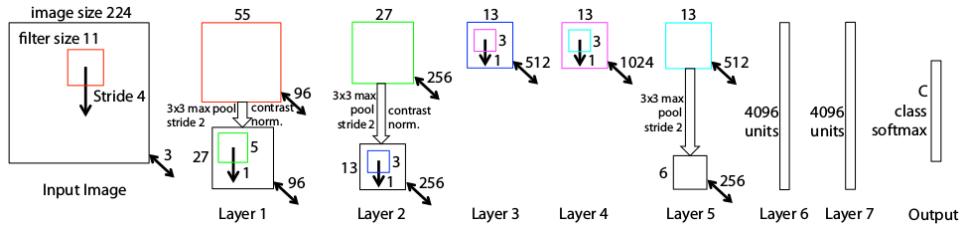


FIGURE 3.8 – Architecture utilisé pour le conv net. [Zei et al. 14]

Il a entraîné ce réseau à convolution sur la base d'image de ILSVRC-2012. Il a utilisé la Cross-Entropy pour le calcul d'erreur. L'entraînement fut effectué par descente du gradient avec des mini-batch d'une taille de 128 images, un Learning-Rate de 0.01 et un dropout sur les dernières couches d'une valeur de 0,5. Les poids furent initialisés à 0,01 et les biais à 0.

La deuxième architecture que Matthew Zeiler a proposé dans son travail, offre une nouvelle manière de retracer les activités des caractéristiques détectées jusqu'à arriver vers les pixels qui les ont activés. Cette architecture s'intitule : *réseaux de déconvolution*, ces derniers reprennent la même architecture que le réseau à convolution de la première architecture, mais d'une manière inverse. Ils effectuent dans l'ordre suivant ces étapes :

1. **Unpooling** : Le max-pooling qui est généralement utilisé dans les réseaux à convolution n'est pas inversible. L'Unpooling essaye d'approximer l'inverse du max-pooling, l'idée est donc de sauvegarder l'emplacement du maximum pour chaque région au moment de l'application d'un max-pooling. Comme le montre la figure suivante :

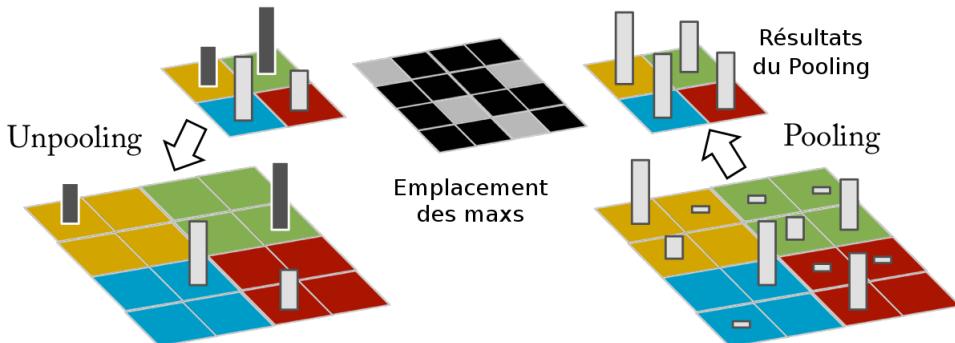


FIGURE 3.9 – Schéma descriptif du unpooling. [Zei et al. 14]

2. **Rectification** : Elle s'effectue à travers l'application de la fonction de non-linéarité "ReLU", la rectification ne fait que prendre le maximum entre la valeur en entrée de la fonction et 0 pour garder seulement les résultats positifs, les négatifs deviendront des zéros.
3. **Filtrage** : la dernière étape est d'appliquer l'opération inverse des filtres (matrices) de convolution. Elle est considérée comme étant la transposé des filtres (matrices) de convolution apprises.

3.6.2 Visualisation des caractéristiques

Après avoir implémenté le réseau à déconvolution, Zeiler passe à la tache de visualisation où il tente de donner un sens à l'apprentissage qu'il effectue un réseau à convolution, en d'autres termes : montrer ce que chaque masque de convolution détecte. L'approche est donc de tenter de trouver les zones d'images en entrée qui sont à l'origine de l'activation de chaque masque.

Par soucis de temps, nous n'avons pas pu implémenter les réseaux à déconvolutions sur notre implémentation. Les résultats obtenus par Zeiler [Zei et al. 14] sont représentés sur la figure ci-dessous, elle représente le top-9 des zones qui causent le plus grand taux d'activation pour un masque donné.



FIGURE 3.10 – Visualisation des différences. [Zei et al. 14]

On peut voir la nature hiérarchique de la reconnaissance, dans les premières couches, des formes basiques sont détectées (la détection de différents traits et lignes par la couche 1, différents types de coins par la couche 2). Ces formes basiques se combinent d'une couche à une autre pour arriver à des représentations plus complexes (des textures dans la couche 3), jusqu'à pouvoir reconnaître des objets plus significatifs dans les dernières couches (humain, chien, roue, fleur, etc.), c'est cette dernière reconnaissance qu'on appellera comme *description sémantique*.

Matthew Zeiler [Zei et al. 14] observe ainsi l'évolution des caractéristiques (features) re-

connues durant l'entraînement et fait une analyse de sensibilité (sensitivity analysis). Comme nous ne sommes pas certain si la détection se fait par rapport aux objets eux mêmes, ou bien à ce qu'ils ont autour (l'entourage des objets). Il propose pour remédier à ce problème, de cacher certaines parties des images(avec des rectangles gris), les résultats des tests montrent que le taux de reconnaissance diminue d'une façon radicale, ce qui prouve donc que la détection se fait bien sur les objets eux mêmes et non leur entourage.

Nous pouvons donc assumer qu'au final, que la représentation retournée par le réseau est bien une représentation sémantique du contenu de l'image, elle décrit donc d'une certaine façon ce contenu. C'est cette représentation qui sera traduite en étiquettes (1000 classes) à travers la dernière couche Soft-max. Cette représentation force le réseau à utiliser des termes (étiquettes) bien précis pour exprimer ce qu'il a appris, c'est pour cette raison qu'on retrouve donc de meilleur résultats lors de l'utilisation des sorties de l'avant dernière couche (4096b) où les concepts appris sont encore libres de cette contrainte. Enfin, plus les représentations sont traitées en profondeur, plus elles sont significatives, et plus le réseau apprend des concepts plus complexes. C'est par ceci que sont expliqués les résultats pas très bons de l'avant avant dernière couche (4096a).

3.7 Description sémantique avec les Autoencoders

Après avoir décrit ce qu'est la description sémantique d'une image, nous nous sommes intéressés dans nos approches suivantes sur l'utilisation des Deep Autoencoders, en les appliquant sur les descriptions sémantiques que nous avons obtenu à partir du réseau à convolution que nous avons utilisé. Pour cela, nous avons choisi de prendre comme description sémantique celle de la couche 4096b, vu les bons résultats qu'on a pu obtenir avec couche, qu'on essayera d'améliorer avec l'ajout des Deep Autoencoders. Les Deep Autoencoders que nous avons implémentés auront pour but créer une représentation compressée des 4096 valeurs de la couche 4096b en seulement 1000 valeurs. Le choix de la valeur 1000 a été fait pour comparer entre les performances de la représentation par étiquette de la couche Soft-max 1000 et la représentation sémantique compressée de la couche 4096b avec les Deep Autoencoders.

Comme pour un réseau de neurones classique, les Deep autoencoders ont doivent être entraînés sur une base d'exemple. Nous avons utilisé une nouvelle base d'image qui est IAPR TC-12 [Gru et al. 06] contenant 20000 images différentes. Nous avons choisi cette base d'image pour ne pas obtenir des résultats biaisés lors des calculs des ressemblances entre les images et avoir une meilleure généralisation. De cette manière, les Deep Autoencoders que nous avons implanté créeront des représentations compressées des images des bases WANG et Caltech-101, en ayant appris à faire cela sur une toute autre base qui est IAPR TC-12.

Chacune de nos architectures des autoencoders que nous avons implanté ont été entraînées pendant plusieurs heures, entre 7 heures et 10 heures pour chaque architecture, pour un nombre d'epoch de 1000, un learning-rate de 0.1 et une taille de batch de 20.

3.7.1 Autoencoder 4x1x4

Nous avons implanté notre première architecture de Deep Autoencoder de sorte à obtenir une description compressée de la couche sémantique 4096b en 1000 valeurs seulement. Nous avons donc choisi une architecture très simple, comme le montre la figure suivante :

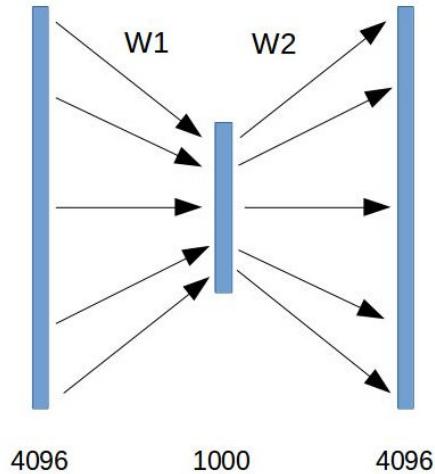


FIGURE 3.11 – Architecture Autoencoder 4x1x4.

Cette architecture a pour entrée une couche de 4096 valeurs où qui contiendra la représentation sémantique 4096b d'une image, une couche cachée de 1000 valeurs où le Deep Autoencoder créera une représentation compressée des 4096 valeurs, et finalement la couche de sortie de 4096 valeurs où le Deep Autoencoder essaiera de reconstruire l'entrée de 4096 valeurs à partir des 1000 valeurs compressées. On notera cette architecture en **Autoencoder 4x1x4**. Les résultats des mesures de performances que nous avons obtenus sont comme suit :

Mesure	1-Préc	5-Préc	10-Préc	20-Préc	40-Préc	60-Préc	Rappel
WANG	0.925	0.915	0.888	0.853	0.784	0.715	0.637
Caltech-101	0.524	0.429	0.369	0.304	-	-	0.216

TABLE 3.5 – Résultats obtenus avec l'Autoencoder 4x1x4.

Nous remarquons que les résultats pour la base WANG ne sont pas meilleurs que ceux obtenus avec la représentation sémantique profonde de la couche 4096b, mais sont quand même meilleurs que ceux de la représentation sémantique non profonde de la couche 4096a, mais surtout meilleur que la représentation par étiquette de la couche Soft-max 1000. Par contre pour la base d'image Caltech-101 ce n'est pas le même cas, la représentation avec l'autoencoder 4x1x4 ne donne pas de de meilleure résultats par rapport à toutes les autres approches précédente. Comme les résultats ne sont pas les mêmes entre les deux bases d'images que nous avons testé, nous allons essayer d'autres architecture de Deep Autoencoder.

3.7.2 Autoencoder 4x2x1x2x4

Comme l'approche précédente, la prochaine architecture de Deep Autoencoder que nous avons implémenté aura pour but d'obtenir une description compressée de la couche sémantique 4096b en 1000 valeurs. Cette architecture sera par contre plus profonde que la précédente, comme le montre la figure suivante :

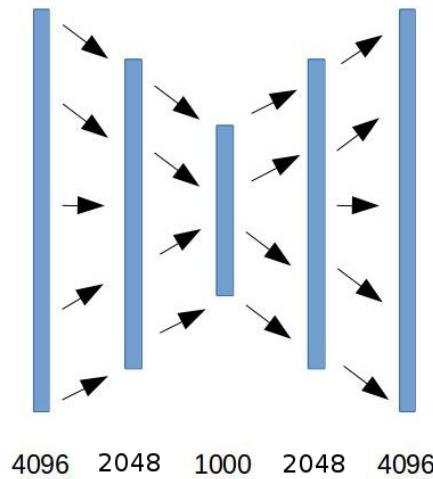


FIGURE 3.12 – Architecture Autoencoder 4x2x1x2x4.

Cette architecture est un Deep Autoencoder plus profond que le précédent. La couche d'entrée aura 4096 valeurs qui seront la représentation sémantique de la couche 4096b, la même chose pour la couche de sortie où on essaiera de reconstruire l'entrée du réseau. Par contre, nous avons trois couches cachées contrairement à une seule couche cachée seulement pour l'architecture précédente. La première couche cachée contiendra 2048 valeurs, la deuxième 1000 valeurs, et la troisième 2048 valeurs. Cette architecture aura comme objectif de compresser les 4096 valeurs en 2048 valeurs ensuite en 1000 valeurs, et puis reconstruire les 2048 valeurs à partir des 1000, et reconstruire les 4096 valeurs à partir des 2048 de la troisième couche. On notera cette architecture en **Autoencoder 4x2x1x2x4**. Les résultats que nous avons obtenus sont comme suit :

Mesure	1-Préc	5-Préc	10-Préc	20-Préc	40-Préc	60-Préc	Rappel
WANG	0.95	0.934	0.925	0.907	0.872	0.826	0.756
Caltech-101	0.772	0.688	0.636	0.556	-	-	0.420

TABLE 3.6 – Résultats obtenus avec l'Autoencoder 4x2x1x2x4.

Pour les deux bases WANG et Caltech-101, nous avons obtenus des résultats presque similaires entre la couche 4096b et cette architecture Autoencoder 4x2x1x2x4. La représentation sémantique de 4096b donne de meilleures performances lors de la recherche d'images avec les Précisions 1-Préc, 5-Préc et 10-Préc. Par contre pour le reste des mesures surtout le Rappel, l'architecture Autoencoder 4x2x1x2x4 est un peu meilleur que celle de 4096b.

La différence entre les deux approches est qu'avec celle de 4096b nous comparons 4096 valeurs, quand à la représentation Autoencoder 4x2x1x2x4 nous ne comparons que 1000 valeurs seulement vu que c'est une représentation compressée.

3.7.3 Denoising Autoencoder 4x1x4

En regardant les résultats deux architecture de Deep Autoencoder, nous avons remarqué qu'il n'y avait pas vraiment d'amélioration ou seulement une légère amélioration. Au lieu de changer d'architecture de Deep Autoencoder, nous avons pensé à implémenter un autre type de Deep Autoencoder qui sont les Deep Denoising Autoencoder. L'architecture d'un Deep Denoising Autoencoder ressemble à l'architecture d'un Deep Autoencoder, c'est la manière d'envoyer les valeurs en entrée qui sera changée, comme le montre la figure suivante :

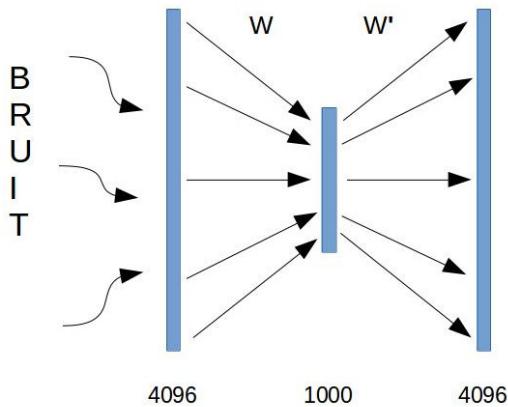


FIGURE 3.13 – Architecture Deep Denoising Autoencoder 4x1x4.

Cette architecture ressemble donc à l’architecture Autoencoder 4x1x4, comme nous le remarquons chaque donnée en entrée va être corrompue avec un certain taux de corruption. Donc les valeurs en entrée du réseau seront les valeurs de la couche sémantique 4096b, mais qui auront des valeurs modifiées à un certain taux. Le Deep Denoising Autoencoder que nous avons implémenté aura pour but dans ce cas de reconstruire à partir des 4096 valeurs corrompues, les 4096 valeurs non corrompus.

Notre implémentation de Deep Denoising Autoencoder 4x1x4 a un taux de corruption de 30% donc $4096 * \frac{30}{100} = 1228$ valeurs corrompues, les résultats obtenus sont comme suit :

Mesure	1-Préc	5-Préc	10-Préc	20-Préc	40-Préc	60-Préc	Rappel
WANG	0.965	0.95	0.936	0.926	0.905	0.870	0.808
Caltech-101	0.813	0.748	0.705	0.638	-	-	0.511

TABLE 3.7 – Résultats obtenus avec le Denoising Autoencoder 4x1x4.

Les résultats de cette approches sont les meilleurs de tous les résultats des approches précédentes, que ce soit pour base WANG ou pour la base Caltech-101. Une amélioration des performances allant jusqu’à 10% dans le Rappel par rapport à l’approche de la couche 4096b. Nous pouvons donc déduire que notre approche avec le Deep Denoising Autoencoder a pu construire une représentation compressée de 4096b en seulement 1000 valeurs, où il garde seulement les informations utiles de la représentation sémantique 4096b, et cela améliore nettement la comparaison des images et donc la recherche d’image par le contenu.

3.7.4 Résultats

Les deux graphes suivants montrent une comparaison entre l’approche 4096b et ceux des différentes architectures d’Autoencoders que nous avons implémentées :

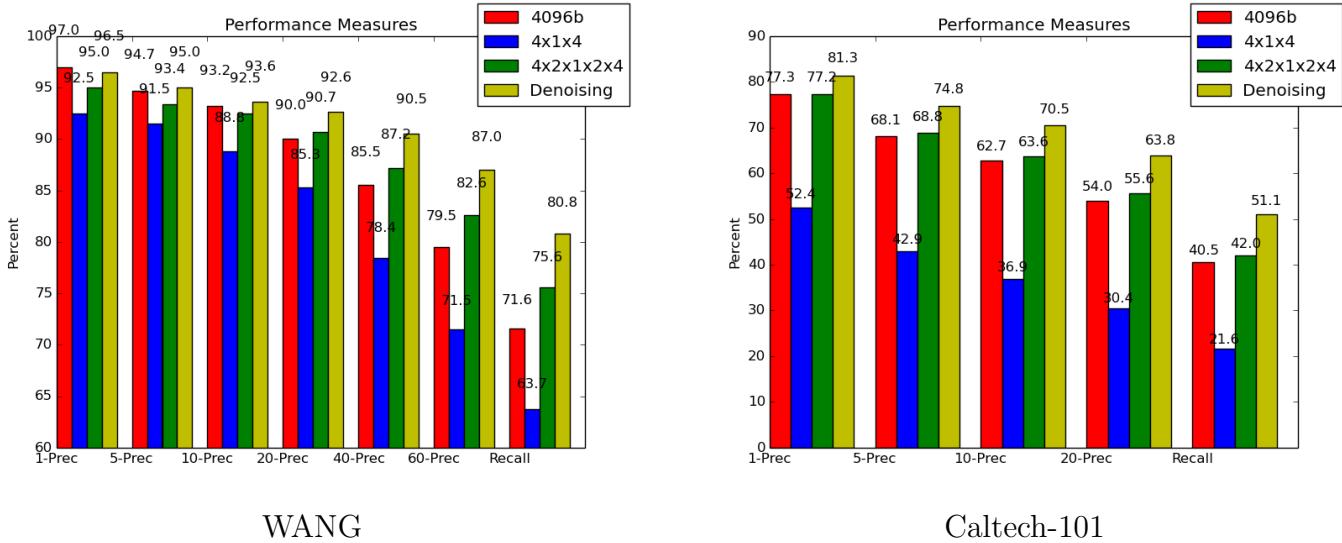


FIGURE 3.14 – Résultats avec Autoencoder obtenus sur les deux bases.

Nos deux premières architectures d’Autoencoders qui sont Autoencoder $4 \times 1 \times 4$ et $4 \times 2 \times 1 \times 2 \times 4$ ne donne pas de meilleurs résultats par rapport à l’approche 4096b, mais avec notre troisième architecture du Denoising Autoencoder, on remarque une très grande amélioration. Cela est dû à l’ajout des bruits (noises) et donc de rajouter des contraintes supplémentaires au Denoising Autoencoder pour créer des représentations compressées plus efficacement. Ces résultats montrent la force des Deep Autoencoders.

3.8 La force des Deep Autoencoders

En plus de réduire l’espace de représentation pour prouver que ce n’est pas finalement ce dernier qui influe sur les résultats, l’utilisation des Autoencoders a montré une amélioration considérable des résultats de recherches d’images.

Pour comprendre pourquoi ceci s’est produit nous allons explorer plus en détail le fonctionnement de l’Autoencoder [Goo et al. 16]. L’idée est, comme expliqué précédemment dans le chapitre 2, que les données sont concentrées autour d’une partie d’un espace d’une dimension (appelée Variété ou Manifold) inférieure à celle de l’espace de représentation actuelle des données. L’objectif est de retrouver la structure de ce cette dimension réduite. Un Autoencoder effectue cela en faisant le compromis entre les deux contraintes suivantes :

- i Apprendre une représentation h d’un exemple en entrée x (h étant généralement d’une dimensionnalité plus inférieure) tel que x peut être récupéré à partir de h grâce à un décodeur, et ceci avec un minimum d’erreur. Si le x' reconstruit à partir x n’appartient pas à l’espace de données en entrée, la reconstruction n’est pas sensée donner de bon résultats.
- ii Le deuxième point à satisfaire est l’ensemble de régularisations qui sont infligés à un Autoencoder qui ont généralement pour but d’éviter le sur-apprentissage et permettre donc d’avoir une représentation plus générale.

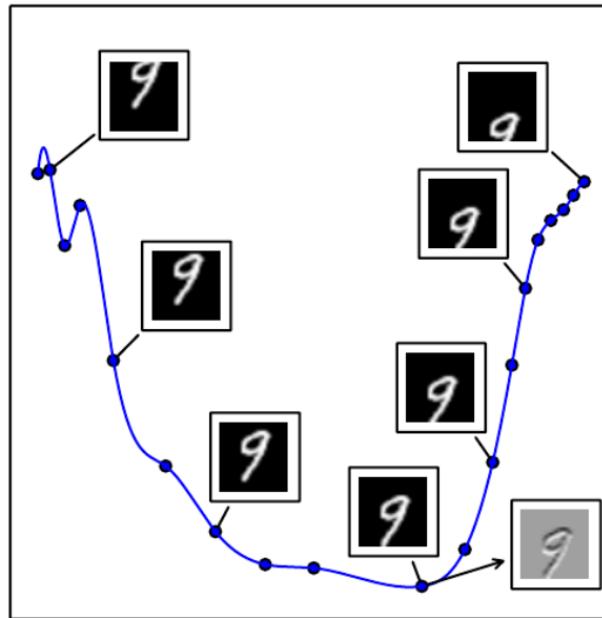


FIGURE 3.15 – Exemple d'une Variété à une dimension [Goo et al. 16] dans un espace de 784 dimensions des images de la base MNIST (28×28 pixels) [LeCun et al. 98]. Une image fut prise et déplacée verticalement de haut en bas. Le taux de déplacement vertical défini des coordonnées qui tracent une courbe. Pour la visualiser, cette dernière a été projetée vers un espace à 2 dimensions en utilisant PCA (Principal Component Analysis).

Nous expliquons donc, dans notre cas, la hausse de niveau de précision dans la comparaison des images par le fait que les représentations sémantiques retenues par le réseau à convolution soient compressées par le codeur, il constitue dorénavant un moyen de résumer les représentations en les combinant d'une façon à en garder ce qu'il y a de plus essentiel.

Ensuite, les contraintes ajouté au Denoising Autoencoder lui ont permis d'apprendre une compression plus générale et plus efficace, d'où ses résultats remarquables.

3.9 Recherche avec descripteur d'image

Comme nous l'avons mentionné au début de ce chapitre, nos contributions aux techniques de recherche d'images par le contenu basée-exemple est de proposer des alternatives aux descripteurs usuels qui étaient utilisés pour calculer la ressemblance des images et les comparer entre elles. Nous avons ensuite montré la puissance des techniques de l'apprentissage profond et ce qu'ils peuvent donner comme résultats. Dans cette partie, nous allons rajouter des descripteurs de texture aux meilleures représentations que nous avons obtenus (4096b et Denoising Autoencoder). Pour les descripteurs de texture, nous avons choisi le Local Binary Pattern [Oja et al. 02] (Motifs Locaux Binaires) et les matrices de co-occurrences GLCM [Har 79] (Grey Level Co-Occurrences Matrix).

3.9.1 4096b avec descripteur

Dans cette approche nous allons rajouter aux 4096 valeurs de la couche sémantique 4096b deux descripteurs, qui sont l'histogramme des motifs uniformes du Local Binary Pattern avec invariabilité aux rotations [Oja et al. 02], et les mesures suivantes des matrices de

co-occurrences [Har 79] : contraste, dissimilarité, homogénéité, énergie, corrélation, second moment angulaire.

Les résultats obtenus sont représentés sur le tableau suivant :

Mesure	1-Préc	5-Préc	10-Préc	20-Préc	40-Préc	60-Préc	Rappel
WANG	0.964	0.949	0.937	0.919	0.880	0.835	0.713
Caltech-101	0.774	0.681	0.627	0.539	-	-	0.405

TABLE 3.8 – Résultats obtenus avec l'ajout des descripteurs de texture.

Les résultats que nous avons obtenus ressemblent beaucoup à ceux des résultats de la couche sémantique 4096b pour la base Caltech-101, et on remarque une légère amélioration de 2% à 3% dans certaines mesures de performances pour la base d'image WANG. On peut dire que l'ajout de descripteur n'a pas un impact important lors de la comparaison des images.

3.9.2 Denoising Autoencoder avec descripteur

Nous allons dans cette approche prendre les mêmes valeurs de l'approche précédente qui sont les 4096 valeurs de la couche sémantique 4096b plus de l'histogramme du Local Binary pattern et les mesures de matrices de co-occurrences, ces valeurs seront données en entrée à une nouvelle architecture de Deep Denoising Autoencoder que nous avons implémenté qui est représentée sur la figure suivante :

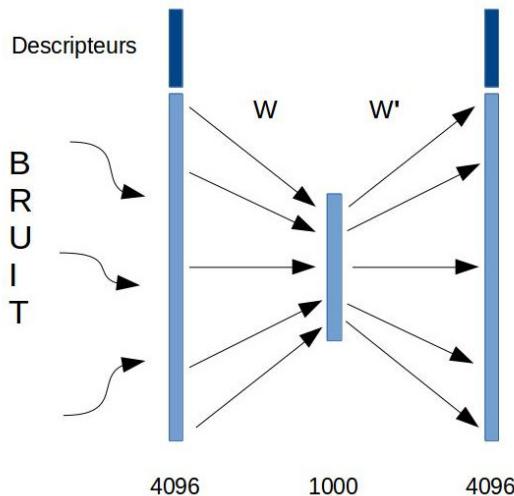


FIGURE 3.16 – Architecture Denoising Autoencoder 4x1x4 avec descripteur.

Cette architecture a une couche d'entrée contenant les valeurs de la couche 4096b plus les valeurs des descripteurs mais avec un taux de corruption de 30%, nous avons dans cette architecture une seule couche cachée qui a une taille de 1000 valeurs, et une couche de sortie qui devra reconstruire les 4096 valeurs plus les valeurs des descripteurs de textures tout en détectant les valeurs corrompues. Les résultats obtenus sont comme suit :

Mesure	1-Préc	5-Préc	10-Préc	20-Préc	40-Préc	60-Préc	Rappel
WANG	0.96	0.952	0.939	0.929	0.905	0.874	0.811
Caltech-101	0.816	0.746	0.707	0.641	-	-	0.512

TABLE 3.9 – Résultats obtenus avec Denoising Autoencoder avec descripteurs de texture.

C'est résultats sont presque similaires aux résultats obtenus avec le Denoising Autoencoder sans descripteurs, nous remarquons par contre une très légère amélioration de 0.2% à 0.3% dans presque toutes les mesures de performances des deux bases d'images.

3.9.3 Résultats

Les deux graphes suivants montrent les résultats entre l'approche 4096b, le Denoising Autoencoder et les deux approches avec les descripteurs :

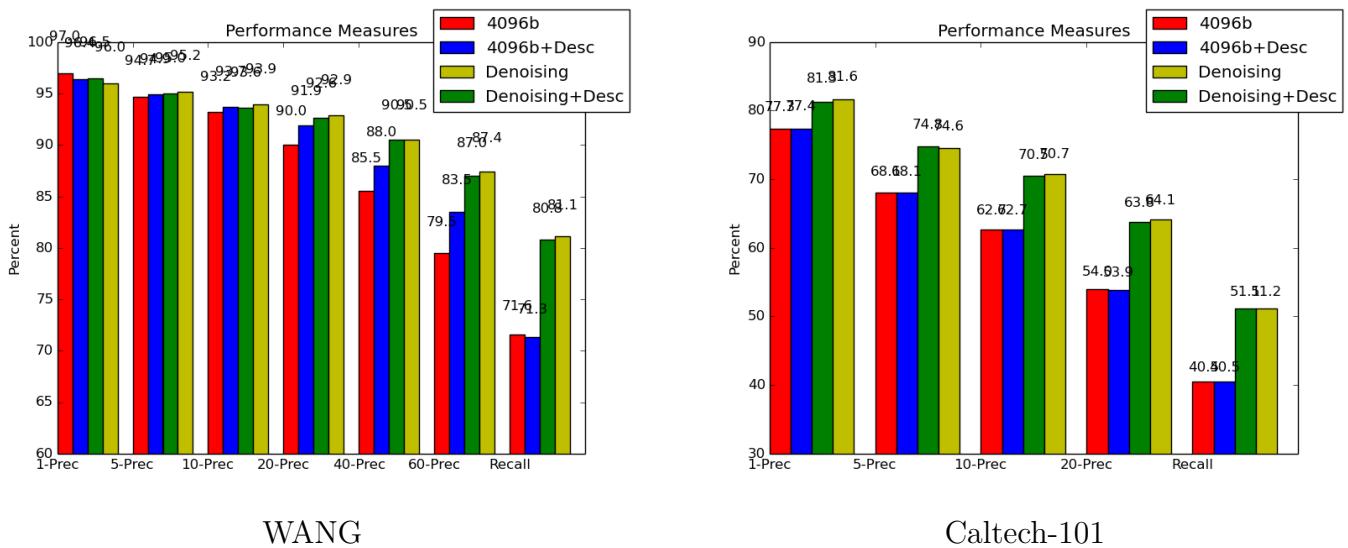


FIGURE 3.17 – Résultats avec descripteurs obtenus sur les deux bases.

Nous remarquons que l'ajout ou non de descripteurs au représentations sémantiques n'affecte pas les résultats d'une manière considérable. Nous pouvons donc dire que nos représentations sémantiques de nos approches offrent de bonnes descriptions des images sans avoir besoins d'ajouter des descripteurs d'images. On pourrait aussi dire que les informations d'une image qu'offrent les descripteurs sont déjà incluses dans la description sémantique de l'image.

3.10 Comparaison avec d'autres systèmes CBIR

Dans cette partie nous allons comparer nos meilleures approches qui sont la description sémantique de la couche 4096b, et la description du Denoising Autoencoder 4x1x4 sans descripteurs. Comme exemple de système de recherche d'image par le contenu, nous avons choisi deux systèmes de recherche qui sont FIRE [Des et al. 08] et LIRE [Lux et al. 13], parce que ces deux systèmes ont utilisé des descripteurs traditionnels pour donner une description des images, et ont présenté les mesures de performances que leurs systèmes ont obtenus sur la base WANG [Li et Wan.,03][Wan et al.,01]. Parmi ces mesures de performances ils

utilisent le taux d'erreur (événement inverse de la 1-Précision) pour comparer leurs différentes approches.

Les résultats des deux systèmes LIRE et FIRE sont représentés dans les deux tableaux suivants :

Descripteur	Taux d'erreur
color histogram	0.169
LF SIFT global search	0.372
LF patches histogram	0.179
LF SIFT histogram	0.256
inv. feature histogram (monomial)	0.192
MPEG7 : scalable color	0.251
LF patches signature	0.243
Gabor histogram	0.305
32x32 image	0.472
MPEG7 : color layout	0.354
Xx32 image	0.559
Tamura texture histogram	0.284
LF SIFT signature	0.351
gray value histogram	0.453
LF patches global	0.429
MPEG7 : edge histogram	0.328
inv. feature histogram (relational)	0.383
Gabor vector	0.655
global texture feature	0.514

TABLE 3.10 – Résultats du système FIRE.

Descripteur	Taux d'erreur
Auto color correlogram	0.171
CEDD	0.178
Color histogram	0.191
FCTH	0.209
Gabor	0.707
JCD	0.177
Joint histogram	0.196
JPEG coefficients histogram	0.215
MPEG-7 color layout	0.309
MPEG-7 scalable color	0.462
MPEG-7 edge histogram	0.401
SIFT BoVW	0.687
Tamura	0.601

TABLE 3.11 – Résultats du système LIRE.

Rappelons que la base WANG contient 1000 images (10 classes des 100 images), contrairement à notre algorithme de recherche d'image par le contenu où nous avons divisé la base WANG en deux parties (la première contenant 20% des images et la deuxième le reste des

80%), les approches des deux systèmes sont un peu différentes. Ils calculent le taux d'erreur pour les 1000 images et non pas 200 requêtes comme dans notre algorithme, pour cela ils prennent chaque image et ils la comparent avec les 999 autres images, si l'image la plus ressemblante appartient à la même classe, le taux d'erreur sera égal à 0 sinon à 1 dans le cas contraire.

Le choix du taux d'erreur comme mesure pour comparer les différents modèles de recherche d'image par le contenu, est dû au fait que quand un utilisateur cherche les images ressemblantes à son image (à partir d'un smartphone par exemple), il s'intéresse généralement au top-5 des images ressemblantes, et principalement au top-1, donc il est plus important de donner les bonnes images pertinentes en premier que de retourner plusieurs images qui ne sont pas toutes forcément pertinentes.

Nous avons donc implémenté un nouvel algorithme qui utilise la même manière de comparaison que les deux systèmes LIRE et FIRE (1 image requête comparée avec les 999 autres images), pour nous permettre de les comparer avec nos approches, les résultats sont comme suit :

Comp	1000	4096a	4096b	AE414	AE42124	Denosing	4096b+Desc	Denoising+Desc
1-Précision								
20%-80%	0.86	0.84	0.97	0.925	0.95	0.965	0.97	0.96
1-999	0.885	0.87	0.964	0.935	0.953	0.966	0.964	0.967
Rappel								
20%-80%	0.442	0.514	0.716	0.637	0.756	0.808	0.717	0.811
1-999	0.413	0.520	0.713	0.641	0.758	0.806	0.713	0.807

TABLE 3.12 – Comparaison des mesures obtenues entre les deux manières de comparaison.

Avant de comparer nos résultats avec les bases LIRE et FIRE, nous allons d'abord comparer nos deux manières de comparaison de la base WANG que nous avons implémenté, nous remarquons que les 1-Précision et Rappels des deux manières de comparaison sont très similaires, avec une différence rare de 3% au maximum. On peut donc dire que les mesures de performance sur une partie aléatoire d'une base d'image, sont approximativement les mêmes que les mesures sur la totalité de la base. Le tableau [Tableau 3.13] ci-dessous donne les valeurs des taux d'erreur avec la manière (1 avec 999) :

Mesure	1000	4096a	4096b	AE414	AE42124	Denosing	4096b+Desc	Denoising+Desc
1-Préc	0.885	0.87	0.964	0.935	0.953	0.966	0.964	0.966
Taux-Err	0.115	0.13	0.036	0.065	0.047	0.034	0.036	0.034

TABLE 3.13 – Les mesures obtenues avec la nouvelle manière de comparaison (1 avec 999).

Nous allons comparer tous nos approches avec les descripteurs de LIRE et FIRE qui ont donné les plus petits taux d'erreur, color histogram pour FIRE et Auto color correlogram pour LIRE, les résultats sont présentés sur le graphe suivant :

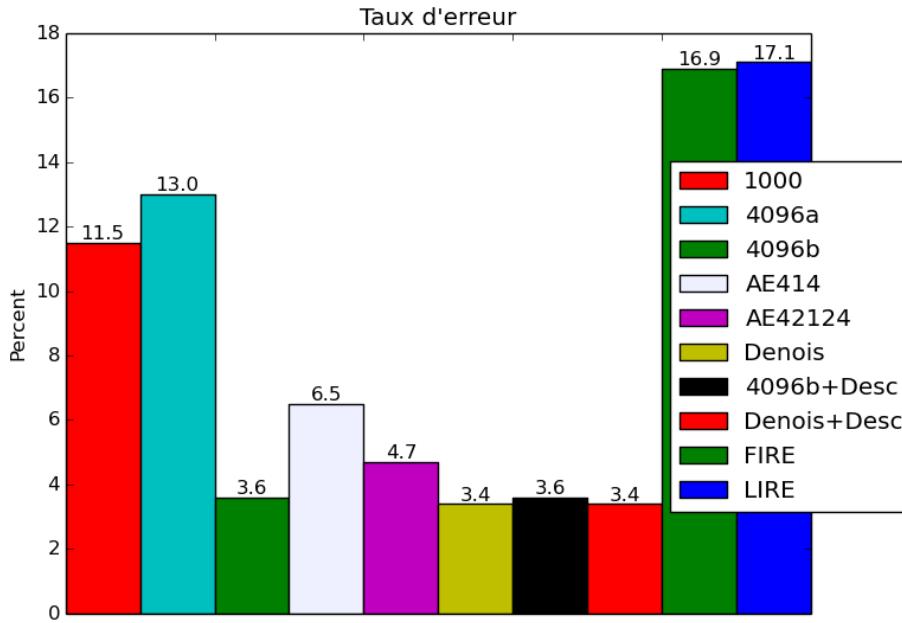


FIGURE 3.18 – Comparaison de nos approches avec FIRE et LIRE.

Il est clair que les taux d'erreur de nos approches sont très largement petits par rapport à ceux des deux systèmes LIRE et FIRE, sauf pour les approches de la couche Soft-max 1000 (par étiquette) et la couche 4096a (sémantique mais non profonde). Cela confirme la puissance de la description sémantique profonde ainsi que la puissance des Deep Autoencoders. Nos techniques utilisées offrent donc une meilleure alternative aux descripteurs d'images traditionnels.

3.11 Conclusion

Nous concluons ce chapitre en constatant que les méthodes d'apprentissage profond donnent effectivement des résultats très satisfaisants quand on les appliquent au problème de recherche d'image par le contenu. Les méthodes d'apprentissage profond que nous avons implémenté dans nos approches sont les réseaux à convolution (ConvNets) et les Deep Autoencoders, nous avons proposé une nouvelle approche pour décrire une image qui est la description sémantique, nous avons montré à quel point cette description est efficace et comment nous avons pu l'améliorer en utilisant des Deep Autoencoders.

Comme les techniques d'apprentissage profond sont nouvelles et très variées, nos approches pourraient être développées et améliorées pour donner de meilleurs résultats en performance et en temps d'exécution. Nous allons dans le prochain chapitre, aborder les détails techniques de nos implémentations, ainsi qu'une visualisation concrète des exemples d'images retrouvées par notre système.

Chapitre 4

Application

4.1 Introduction

Nous allons aborder dans ce chapitre les détails techniques de notre implémentation (langage de programmation, Frameworks, etc.), ensuite nous présenterons l'interface que nous avons conçu pour l'application de notre approche. Enfin, nous examinerons plus en détail certains exemples et résultats de recherches d'images.

4.2 Outils utilisés

La configuration de la machine utilisée est comme suit : Processeur Intel R Core TM i7-2670QM CPU @ 2.20GHz , système d'exploitation GNU/Linux - Ubuntu 14.04 x64.

Nous avons implanté nos approches avec le langage de programmation Python, version 3.0 et à l'aide de différents Framework et bibliothèques.

Pour développer l'interface graphique de notre implémentation, nous avons utilisé PyQt4, un module pour python qui permet de lier la librairie Qt (framework pour le développement d'applications multi-plates-formes).

4.2.1 Le langage Python

Nous avons choisi ce langage pour différentes raisons dont nous citons :

- Une facilité d'écriture et de compréhension du code.
- Son niveau d'abstraction permet de mettre en place des prototypes et de les tester rapidement
- Sa syntaxe organisée, l'indentation obligatoire rend le code trivialement lisible.
- Un grand nombre de bibliothèques puissante.
- Une documentation très variée.
- Une grande communauté qui aide à obtenir de l'aide rapidement.

4.2.2 Theano

Theano est un framework en Python très utilisé, il est l'un des plus ancien aussi et du coup, beaucoup de recherches se sont basées dessus.

Il a été conçu par le laboratoire MILA (Montréal Institute for Learning Algorithms) de l'Université de Montréal. C'est un projet Open Source que de nombreux autres chercheurs de différents laboratoires et institutions ont fini par rejoindre (Google Deepmind, New York University, NVIDIA Corporate, Meiji University Tokyo ..etc) [Theano 16].

Ce framework permet de définir des expressions mathématiques symboliques en les représentant par des graphes orientés acycliques. Ses derniers sont constitués de deux types de noeuds : **Variable** qui représente les données et **Appliquer** qui représente les opérations mathématiques. Cette représentation permet une manipulation simple des expressions et fonctions mathématiques dont, entre autre, la génération automatique de leurs dérivées.

Par exemple, soit un "scalar" x :

```
» x = theano.tensor.dscalar('x')
```

ensuite on définit une expression, la fonction carré par exemple :

```
» y = x ** 2
```

On peut aussi visualiser le graphe généré qui résume toute la fonction, avec la ligne suivante :

```
» theano.printing.pydotprint(y, outfile="yGraph.png", var_with_name_simple=True)
```

On aura comme résultat le graphe de la figure [Figure 4.1] ci-dessous :

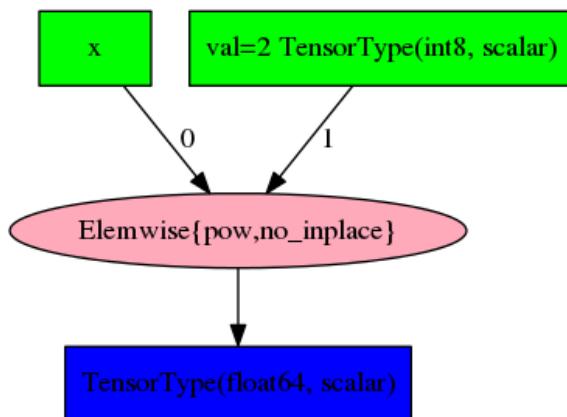


FIGURE 4.1 – Visualisation du graphe d'une fonction

Les nœuds en verts sont les arguments de l'opérateur puissance (pow) qui est en bleu, et le résultat est dans le nœud en rose. Rappelons que lors de la phase d'apprentissage des réseaux de neurones (perceptrons multicouches et autres), il est nécessaire lors de la rétro-propagation du gradient de calculer des dérivés en fonctions de plusieurs paramètres. Theano nous facilite cela puisque les dérivations s'effectuent simplement en appliquant la fonction *grad*. Donc, la dérivée de la fonction y que nous avons défini, par rapport à x peut être calculer comme suit :

```
» gy = T.grad(y, x)
```

gy contient dorénavant la dérivée de y par rapport à x , qui n'est rien d'autre que $2 * x$.

En essayant d'afficher le graphe de gy nous obtenons la figure [Figure 4.2] ci-dessous :

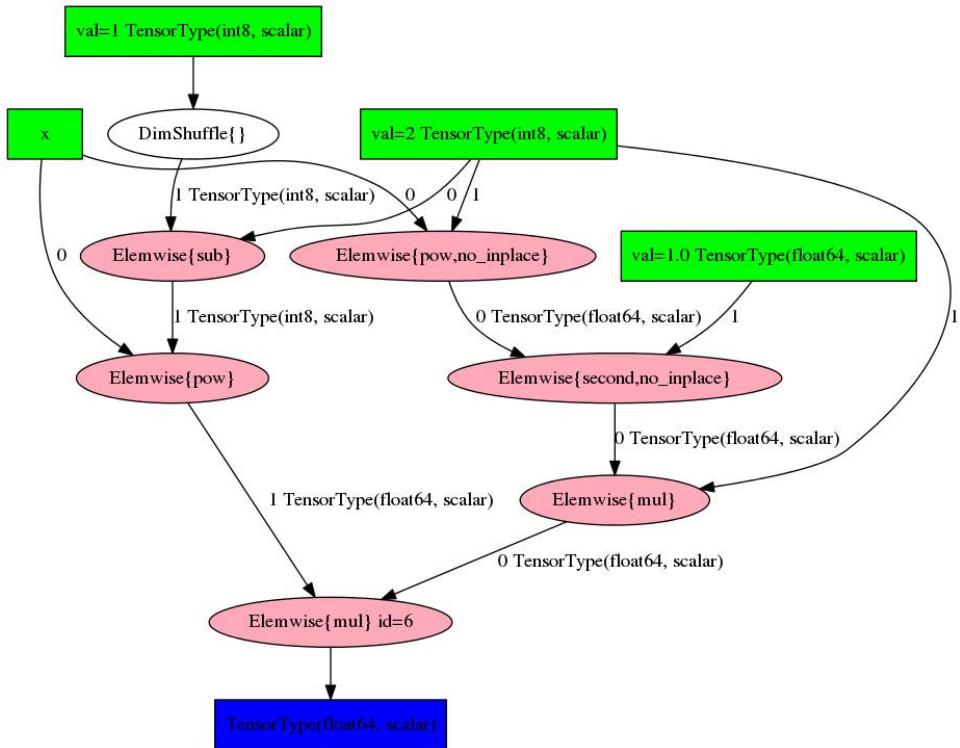


FIGURE 4.2 – Visualisation du graphe de **gy** avant optimisation.

Ce graphe n'est pas évident à comprendre alors qu'il s'agit seulement de la fonction $gy = 2 * x$. Compiler le graphe de **gy** à l'aide de la fonction "function" de Theano pour le rendre exécutable s'effectue comme suit :

```
» f = theano.function([x], gy)
```

Cette opération optimise le calcul de **gy**, et le graphe de **f** devient comme le montre la figure ci-dessous :

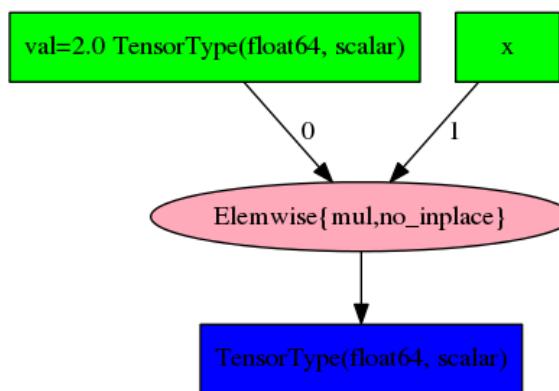


FIGURE 4.3 – Visualisation du graphe de **gy** après optimisation.

Ceci montre à quel point Theano est puissant dans sa gestion des fonctions et de leurs dérivées, ainsi que leurs optimisation en quelques lignes de codes.

Une notion très importante qui existe dans Theano est celle des *variables symboliques*. Elle peuvent être considérées comme des tampons (buffers) qui contiennent des valeurs qui peuvent être partagées entre plusieurs fonctions de Theano.

Un autre point fort de Theano est l'optimisation pour l'utilisation de matrices multidimensionnelles, mais aussi la facilité de compiler et exécuter les opérations sur des GPU au lieu des CPU.

Theano fut suivi par d'autres frameworks qui ont repris les mêmes principes, les plus connus sont Tensorflow de Google, Blocks, Lasagne.

4.2.3 Autres Frameworks

Theano n'étant pas évident à utiliser au début, pour remédier à ce problème, il existe d'autres frameworks qui ajoutent des couches d'abstraction à Theano pour faciliter son utilisation, comme Blocks, Lasagne et Keras. Le modèle VGG-CNN-S du réseau à convolution que nous avons utilisé dans notre algorithme est implémenté en Lasagne après avoir été converti de Caffe (Caffe étant un autre Framework en C++ développé par le laboratoire Berkeley Vision and Learning Center).

Pour implémenté les autoencoders de nos approches, nous avons utilisé Blocks [Mer et al. 15], c'est un framework de theano qui facilite la définition de modèles et leur modifications. Il introduit des outils pour l'apprentissage, la visualisation et la sérialisation. Enfin, le framework Fuel [Mer et al. 15] permet de formater les données d'apprentissage d'une façon qui facilite leurs manipulation, surtout quand les tailles de ses dernières deviennent importantes.

Dans notre cas, au lieu de le définir couche par couche, l'Autoencoder 4x1x4 est défini simplement par :

```
» ae = MLP([Tanh(), Tanh()], [4096, 1000, 4096], weights_init=IsotropicGaussian(0.01),  
biases_init=Constant(0))
```

Puis, il est initialisé par :

```
» ae.initialize()
```

Ces deux frameworks sont développés aussi par le laboratoire MILA, ils forment une base solide pour le développement d'approches basées sur l'apprentissage profond.

4.3 Interface de l'application

Après notre étude comparative dans le chapitre précédent entre les différentes approches que nous avons réalisé, nous avons décider d'implémenter dans l'interface graphique la meilleure approche (celle qui a donné les meilleures mesures de performance), qui est celle du Denoising Autoencoder 4x1x4.

L'interface que nous avons développé pour l'implémentation de notre approche permet de sélectionner une base d'images qui servira de recueil pour les résultats des requêtes, comme le montre la figure [Figure 4.2].

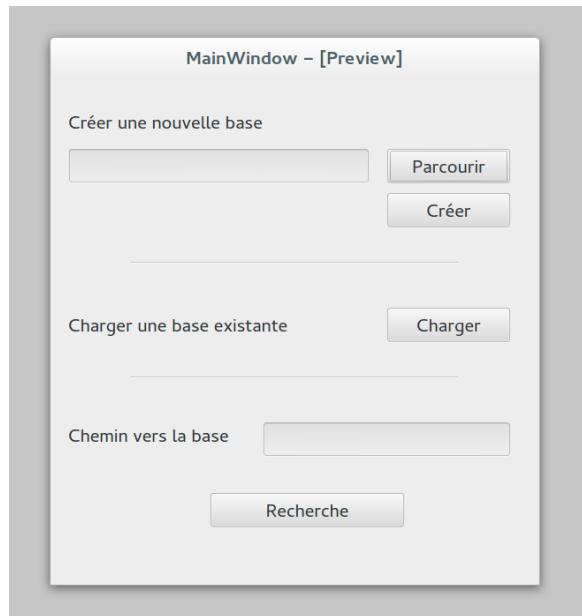


FIGURE 4.4 – Menu principal

Nous pouvons choisir de créer une nouvelle base, pour cela on doit indiquer le lien vers le dossier contenant la base d'images, puis appuyer sur le bouton *Créer*. Le programme dans ce cas va créer la description de chaque image en la faisant passer par le réseau à convolution, ensuite extraire de la description sémantique de la couche 4096b et finalement créer la description compressée en utilisant le Denoising Autoencoder 4x1x4. C'est cette description qui sera enregistrée en tant que modèle pour la recherche d'images. On peut aussi charger une base déjà existante en appuyant sur le bouton *Charger*.

En cliquant sur le bouton "Recherche", une nouvelle fenêtre s'ouvre qui permet de sélectionner une image qui fera office de requête [Figure 4.3]. Une fois l'image sélectionnée, le programme va créer sa représentation sémantique compressée. Le bouton "Chercher" lance la recherche et affiche les cinq images jugées les plus ressemblantes (top-5).

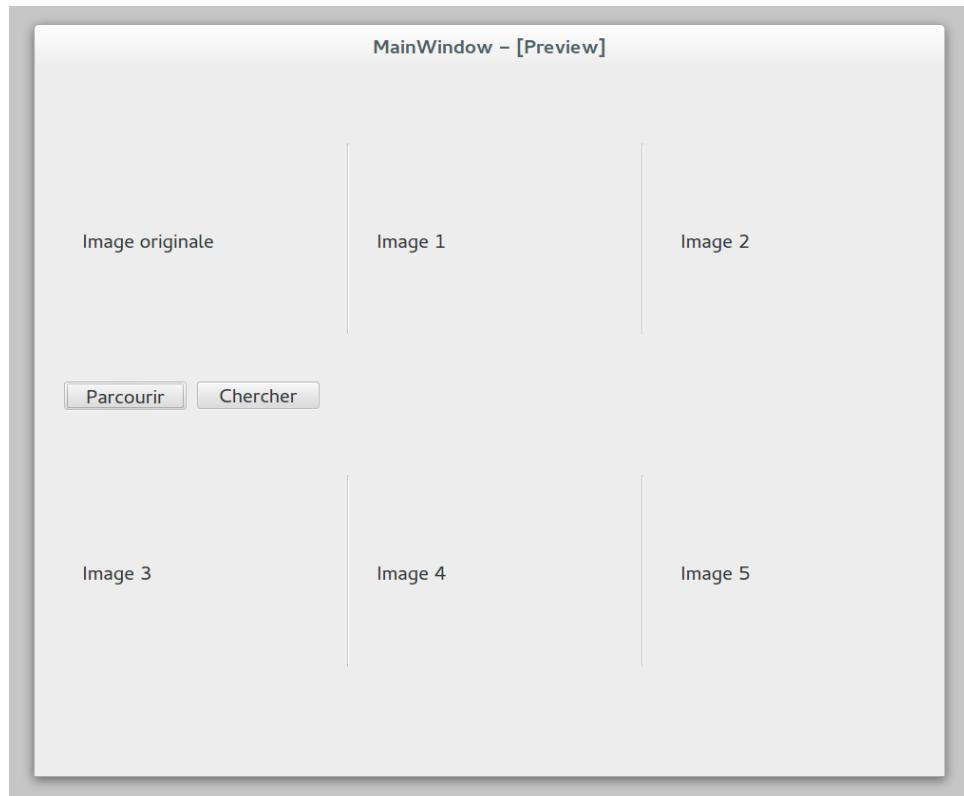


FIGURE 4.5 – Interface de recherche d’images.

4.4 Tests

Nous avons pris quelques images pour effectuer des testes sur les bases de donne WANG et AUTRE.

4.5 Conclusion

Nous concluons ce chapitre.

Conclusion générale

Nous avons, durant notre travail, fait une recherche approfondie sur les techniques d'intelligence artificielle utilisées par les plus grands laboratoires de recherches dans le monde entier (Facebook AI Research, Google DeepMind, Montréal Institute for Learning Algorithms et d'autre), et plus spécialement celles qui sont appliquées dans le domaine du traitement d'image. C'est ainsi que nous avons découvert cette nouvelle tendance qui est l'apprentissage profond (Deep Learning), vers laquelle pratiquement toutes les recherches scientifiques qui traitent de grandes masses de données essayent d'exporter leur recherche. Ceci étant dû à la puissance de ses techniques et aux résultats phénoménaux qu'elles continuent à produire, et qui semblent facilement surpasser les techniques classiques.

De notre part, nous avons proposé des approches dans le problème de recherche d'image par le contenu (CBIR) où nous avons essayé, dans ce travail, de faire apprendre à la machine à créer des représentations sémantiques d'une image, dans le but de pouvoir la comparer avec d'autres images.

Nous avons essayé différentes approches, et avons démontré que la différence de précision entre les couches des réseaux à convolutions, n'était pas dû à la compression des espaces de représentation (de 4096 valeurs à seulement 1000 valeurs) mais plutôt au type de l'information que contiennent les représentations, et à quelle profondeur le traitement se fait.

Nous avons aussi montré que l'ajout de certains descripteurs fabriqués à la main (description de texture, couleur, forme) n'avait finalement pas autant d'impact sur les descriptions sémantiques des réseaux à convolutions. Cela est logique, car il est très peu probable que le cerveau humain effectue ce genre d'opérations mathématiques formelles. Comme l'apprentissage profond tente d'imiter le fonctionnement du cerveau, ces descripteurs traditionnels s'avéreront très peu utiles.

Une amélioration de la structure de nos descriptions et du temps de recherche peut-être permise grâce à l'apprentissage d'une représentation binaire, au lieu d'une représentation par une liste de nombres réels. Elle permettrait la construction d'un arbre de hachage qui optimisera le temps de recherche et de récupération d'images similaires. Une méthode inspirée par le travail de [Kri et al. 11] qui ont fait passer leurs images (représentations en pixels) dans un autoencoder à base de réseau de croyances (DBN), et les ont compressé en une représentation binaire. Nous pensons qu'une approche similaire mais qui prendrait, au lieu des pixels brutes de l'image, l'information sémantique apprise par le réseau à convolution pourrait donner de meilleurs résultats.

Bibliographie

[Shr et al. 13] Meenakshi Shruti Pal, Sushil Kumar Garg, International Journal of Advanced Research in Computer Engineering and Technology (IJARCET) Volume 2, Issue 6, June 2013

[Oja et al. 02] T. Ojala, M. Pietikinen, T. Menp, Multiresolution Gray-Scale and Rotation Invariant Texture Classification with Local Binary Patterns. IEEE Transactions on Pattern Analysis and Machine Intelligence 24 971-987, 2002.

[Har 79] R. Haralick, Statistical and structural approaches to texture, Proc. Of IEEE, vol. 67, no. 5, pp. 786-804, May 1979.

[Low 99] David G. Lowe, Object recognition from local scale-invariant features , Proceedings of the International Conference on Computer Vision, vol. 2, 1999, p. 1150-1157.

[Sin et al. 15] Singh, Anshuman Vikram, "Content-Based Image Retrieval using Deep Learning" (2015). Thesis. Rochester Institute of Technology. Accessed from <http://scholar-works.rit.edu/theses/88286/2015>

[Ma et al. 99] Wei-Ying Ma, B.S. Manjunath, NeTra : A toolbox for navigating large image databases, Journal Multimedia Systems - Special issue on video content based retrieval archive Volume 7 Issue 3, May 1999 Pages 184 - 198

[Car et al. 99] Chad Carson, Megan Thomas, Serge Belongie, Joseph M. Hellerstein and Jitendra Malik, Blobworld : a System for Region-Based Image Indexing and Retrieval (long version) EECS Department, University of California, Berkeley, 1999

[Yod 08] Josiah Yoder Content-Based Image Retrieval from large Medical Image Databases, Avinash C. Kak, The Purdue Robot Vision Laboratory, School of Electrical and Computer Engineering at Purdue University. 2008

[Nib et al. 93] W. Niblack, R. Barber, W. Equitz, M. D. Flickner, E. H. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, G. Taubin, QBIC project : querying images by content, using color, texture, and shape, in : W. Niblack (Ed.), Storage and Retrieval for Image and Video Databases, Vol. 1908 of SPIE Proceedings, 1993, pp. 173-187

[Fli et al. 95] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, Peter Yanker, "Query by Image and Video Content : The QBIC System", Computer, vol.28, no. 9, pp. 23-32, September 1995, doi :10.1109/2.410146

[Lux et al. 08] Mathias Lux, Savvas A. Chatzichristofis. LIRE : Lucene Image Retrieval – An Extensible Java CBIR Library. In proceedings of the 16th ACM International Conference on Multimedia, pp. 1085-1088, Vancouver, Canada, 2008

[Lux et al. 13] Mathias Lux, Oge Marques Visual Information Retrieval using Java and LIRE, Morgan Claypool Publishers, 2013

[Des et al. 08] Thomas Deselaers, Daniel Keysers, Hermann Ney Features for Image Retrieval : An Experimental Comparison. Information Retrieval. 2008. Vol. 11. Issue 2. Springer. pp. 77-107.

[Roe et al. 92] Anna W. Roe, Sarah L. Pallaqb Young H. Kwon, and Mriganka Sur, Visual projections routed to the auditory pathway in ferrets : receptive fields of visual neurons in primary auditory cortex, The Journal of Neuroscience, September 1992, 12(g) : 3651-3664

[Mit et al. 97] Tom Mitchell, McGraw Hill, Machine Learning Textbook 1997.

[Goo et al. 16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, 2016, Deep Learning, Book in preparation for MIT Press, <http://www.deeplearningbook.org>

[Mar et al. 89] H. Martens and T. Naes, Multivariate Calibration, John Wiley, Sons Inc.,

New York, 1989.

[Moo 65] Gordon E. Moore, Cramming More Components Onto Integrated Circuits, Electronics, vol. 38, 19 avril 1965

[Ben 14] Yoshua Bengio. Deep Learning. Machine Learning Summer School, Iceland 2014.

[Hub et al. 68] Hubel, D. and Wiesel, T. (1968). Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology (London)*, 195, 215–243

[LeCun et al. 98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, 86(11) :2278-2324, November 1998.

[Kri et al. 12] Krizhevsky, Sutskever, and Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012.

[CS231n 16] CS231n : Convolutional Neural Networks for Visual Recognition (January - March, 2016), Fe-Fei Li, Andrej Karpathy, Justin Johnson, Stanford University.

[Site1] <http://deeplearning4j.org/deepautoencoder.html>

[Site2] Li Jiang, Musings on Deep Learning <https://medium.com/global-silicon-valley/machine-learning-yesterday-today-tomorrow-3d3023c7b519#.u0qtths18>

[Rus et al.,15] 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. (* = equal contribution) ImageNet Large Scale Visual Recognition Challenge. IJCV, 2015. Volume 115, Issue 3 , pp 211-252

[Li et Wan.,03] Jia Li, James Z. Wang, "Automatic linguistic indexing of pictures by a statistical modeling approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 9, pp. 1075-1088, 2003.

[Wan et al.,01] James Z. Wang, Jia Li, Gio Wiederhold, SIMPLIcity : Semantics-sensitive Integrated Matching for Picture Libraries, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol 23, no.9, pp. 947-963, 2001.

[Fei et al. 04] L. Fei-Fei, R. Fergus and P. Perona. Learning generative visual models from few training examples : an incremental Bayesian approach tested on 101 object categories. *IEEE. CVPR 2004, Workshop on Generative-Model Based Vision*. 2004

[Cha et al.14] K Chatfield, K Simonyan, A Vedaldi, A Zisserman - Return of the devil in the details : Delving deep into convolutional nets, British Machine Vision Conference, 2014.

[LeCun et al. 89] LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D. : Backpropagation applied to handwritten zip code recognition. *Neural Comput.* 1(4), 541–551 (1989)

[Zei et al. 14] M.D. Zeiler, R. Fergus Visualizing and Understanding Convolutional Networks, *ECCV 2014* (Honourable Mention for Best Paper Award)

[Gru et at. 06] M. Grubinger, P. Clough, and H. Muller and T. Deselaers, "The IAPR TC-12 Benchmark : A New Evaluation Resource" for Visual Information Systems," Proc. of the Intl. Workshop OntoImage'2006 Language Resources for CBIR, 2006.

[Theano 16] Theano Development Team, Theano : A Python framework for fast computation of mathematical expressions, arXiv e-prints, 2016.

[Mer et al. 15] Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio, "Blocks and Fuel : Frameworks for deep learning," arXiv preprint arXiv :1506.00619 [cs.LG], 2015.

[Kri et al. 11] Alex Krizhevsky Geoffrey E. Hinton, Using Very Deep Autoencoders for Content-Based Image Retrieval, Conference : ESANN 2011, 19th European Symposium on Artificial Neural Networks, Bruges.
