CentraleSupélec

Rapport de stage

Auteur :
Arié Benhamou

Encadrant :
Antoine Veillard

Stage de fin d'études effectué dans le cadre de l'obtention du diplôme d'ingénieur Supélec

effectué en collaboration avec

National University of Singapore
Krakatoa

Août 2016

Remerciements

Je voudrais tout d'abord remercier mes encadrants lors de ce stage : Antoine Veillard et Olivier Morère. Leurs précieux conseils m'ont permis d'acquérir de nouvelles compétences et connaissances, aussi bien au niveau théorique que pratique. Je tiens aussi a remercier les stagiaires avec qui j'ai eu l'occasion de travailler, une grande partie d'entre eux venant d'ailleurs de CentraleSupélec.

Table des matières

Remerciements i

Contents ii

List of Figures iv

Abréviations v

Introduction 1

1 Environnement de travail et objectifs 2
- 1.1 Présentation de l'environnement de travail . . . 2
  - 1.1.1 Krakatoa . . . 2
    - Les locaux . . . 2
    - L'équipe . . . 3
  - 1.1.2 National University of Singapore . . . 3
- 1.2 Objectifs du stage . . . 3
  - NUS . . . 4
  - Krakatoa . . . 4

2 Méthodologie utilisée 6
- 2.1 Outils . . . 6
  - 2.1.1 Git . . . 6
  - 2.1.2 Librairies . . . 6
    - Numpy . . . 7
    - Theano . . . 7
    - Tensorflow . . . 7
- 2.2 Collaboration au sein de l'équipe . . . 8
  - Réunions hebdomadaires . . . 8
  - Partage des serveurs de calcul . . . 8

3 Le Deep Learning appliquée à la classification d'images 9
- 3.1 Le problème de la classification d'images . . . 9
- 3.2 Les réseaux de neurones . . . 10
  - 3.2.1 Les neurones computationnels . . . 10
- 3.3 Les réseaux de neurones . . . 11
- 3.4 Les Convolutional Neural Networks . . . 12

4 Différentes architectures testées **14**
4.1 Présentation théorique des ResNet 14
Residual Blocks 15
4.2 Implémentation en Theano, résultats attendus et résultats obtenus 16
CIFAR-10 16
Architecture 16
Résultats obtenus sur CIFAR-10 17
Dataset X 19
4.3 Stochastic ResNet 25
4.4 Tensorflow 26
4.5 Residual Network of Residual Network 30
4.6 Dernières expériences 31

Conclusion **33**

Bibliographie **34**

Table des figures

1.1 Région de Tien Hai, Vietnam, correspondant à la surface recouverte par les images satellite 4

3.1 Exemple de classification d'image 10
3.2 Neurone computationnel 11
3.3 Réseau de neurones composé de trois couches 11
3.4 Entrainement du réseau 12

4.1 Différence de training error et de validation error avec des réseaux composés de 20 et 56 couches 15
4.2 Illustration d'un Residual Block 15
4.3 Dataset CIFAR-10 16
4.4 Résultats pour 20 couches - 91,9% de validation accuracy 18
4.5 Résultats pour 32 couches - 92,5% de validation accuracy 18
4.6 Résultats pour 50 couches - 92,4% de validation accuracy 18
4.7 Deux indicateurs de performance : precision / recall 19
4.8 Courbes représentant la precision et le recall pour la classe 4 20
4.9 Courbes d'apprentissage pour le Dataset X 21
4.10 Localisation des éléments de la classe 2 au sein du cube 22
4.11 Résultats obtenus avec 20 couches - 89,4 % de validation accuracy 23
4.12 Résultats obtenus avec 32 couches - 91,5 % de validation accuracy 23
4.13 Résultats obtenus avec 8 couches 24
4.14 Résultats obtenus avec 20 couches 24
4.15 Exemple d'output récupéré sur Tensorboard : validation accuracy pour un réseau de 50 couches 26
4.16 Architecture globale du réseau 27
4.17 Architecture d'un Residual Block 27
4.18 Architecture d'un Residual Block dans sa version stochastique 28
4.19 Validation accuracy pour un Stochastic Resnet de 50 couches : 93,18% . . 29
4.20 Validation accuracy pour un Stochastic Resnet de 600 couche : 91,12% . . 30
4.21 Residual Network of Residual Network (RoR) de niveau 3 31

Abréviations

NUS	**N**ational **U**niversity of **S**ingapore
CNN	**C**onvolutional **N**eural **N**etwork
ResNet	**R**esidual **Network**
ResBlock	**R**esidual **B**lock
VAE	**V**ariational **A**uto **E**ncoder
GAN	**G**enerative **A**dversorial **N**etwork

Introduction

Ce rapport a pour but de résumer mon expérience lors de mon stage de fin d'étude effectué dans le cadre de l'obtention du diplôme d'ingénieur Supélec délivré par CentraleSupélec. Durant ces 6 mois, j'ai pu mettre en pratique les connaissances acquises pendant mon cursus académique, notamment pendant mon année de spécialisation en Système Intéractifs et Robotiques à CentraleSupélec, campus de Metz.

Ce stage a été effectué en collaboration avec la National University of Singapore et Krakatoa, une start-up française basée à Singapour. En tant que chercheur en Deep Learning, j'ai eu l'occasion de travailler sur différents réseaux de neurones qui constituent l'état de l'art actuel, dans le but de voir à quoi pourraient être appliqués ces réseaux qui sont parmis les plus récents/performants.

Dans une première partie, nous présenterons les objectifs du stage et le contexte dans lequel celui-ci s'est déroulé. Puis, nous aborderons la méthodologie de travail qui a été mise en place ainsi que le travail en lui-même. Enfin, dans une troisième partie, nous présenterons les résultats obtenus sur les différentes architectures testées.

Chapitre 1

Environnement de travail et objectifs

Ce stage a été effectué en collaboration avec la National University of Singapore et Krakatoa. Ce chapitre a donc pour but de présenter l'environnement dans lequel j'ai été amené à travailler tout au long de ces 6 mois.

1.1 Présentation de l'environnement de travail

1.1.1 Krakatoa

Krakatoa une start-up française implantée à Singapour. Antoine Veillard, qui a été mon encadrant, est l'un de ses co-fondateurs. C'est une start-up orientée sur la recherche, principalement dans le domaine du Deep Learning, et qui a pour but d'appliquer les résultats de ces travaux à des application très variées : finance, projets pour le gouvernement singapourien etc.

Les locaux : c'est dans les locaux de cette société que j'ai eu l'occasion de travailler. Ceux-ci se trouvent dans ce que l'on pourrait appeler le technopôle singapourien, lieu dans lequel se regroupent de nombreuses entreprises opérant dans le domaine des nouvelle technologies.

L'équipe : au début de mon stage, elle était constituée d'Antoine Veillard, co-fondateur, Olivier Morère, doctorant, d'un ancien Supélec ainsi que d'un Supélec en césure. L'équipe a évolué tout au long de ces 6 mois jusqu'à atteindre un maximum de 10 stagiaires. Nous reviendrons sur la collaboration au sein de l'équipe dans la partie "Méthodologie de travail".

1.1.2 National University of Singapore

Classée première université asiatique dans le classement *QS World University Ranking*, NUS est composée de nombreux départements, allant de la médecine au droit, en passant par l'informatique.
NUS a été mon employeur officiel lors de ce stage. Mon maître de stage officiel, Stéphane Bressan, Associate Professor à NUS, encadre de nombreux doctorants dans le département informatique de l'université. J'ai eu l'occasion de le rencontrer à plusieurs reprises, lors des réunions hebdomadaires qu'il organise avec ses doctorants. Lors de chacune de ces réunions, quelques étudiants préparent une présentation et présentent l'avancement de leur projet. Chacun est alors au courant des projets de ses camarades et une cohésion est ainsi créée entre les étudiants.
Ces réunions ont été l'occasion de travailler dans un environnement international, ce qui a pu manquer au sein de Krakatoa, entreprise française avec une équipe entièrement francophone. J'ai ainsi pu découvrir les méthodes de travail d'étudiants venant de Singapour, d'Inde ou encore de Chine.

1.2 Objectifs du stage

Les stagiaires travaillant chez Krakatoa travaillent pour la plupart en collaboration avec NUS ou une autre université, comme par exemple Télécom ParisTech. L'objectif de cette collaboration est souvent le suivant : faire de la recherche en Deep Learning pour le compte de l'université dans le but d'obtenir si possible une publication, et appliquer les résultats de la recherche aux intérêts de Krakatoa.

NUS : l'objectif côté NUS était de travailler sur un projet de classification d'images satellite.

Dans le cadre de ce projet, nous disposions d'un dataset composé 7 images satellite prises au dessus du Vietnam et recouvrant une surface de 226 km carrés.

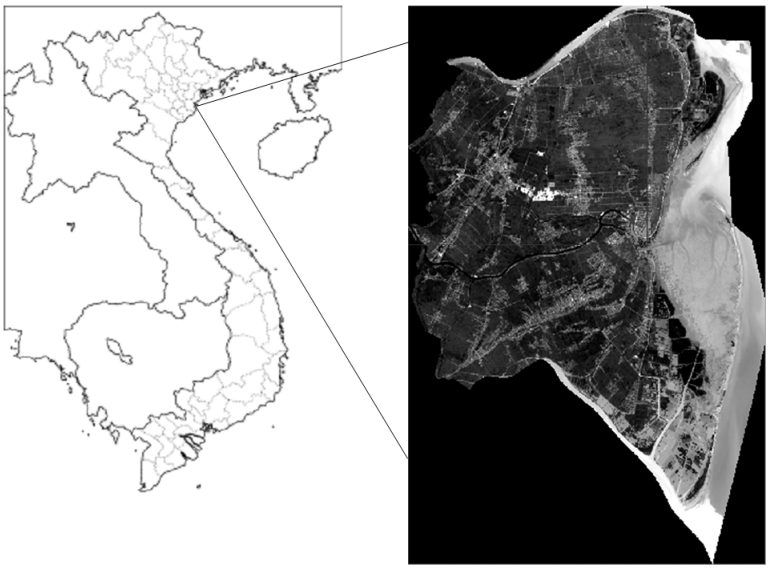

FIGURE 1.1: Région de Tien Hai, Vietnam, correspondant à la surface recouverte par les images satellite

Les pixels de ces images étaient divisés en six classes :

- Aquaculture
- Vasière
- Manglier (une famille d'arbres)
- Rizière
- Constructions
- Eau

Des techniques de classifications autres que le Deep Learning avaient déja été testées dans le passé (maximum likelihood, arbre de décision, support vector machine). Le but était donc d'utiliser les réseaux de neurones afin de voir si nous arrivions bien à de meilleurs résultats.

Krakatoa : du côté de Krakatoa, l'objectif dans le cadre de ce projet était donc de tester différents types de réseaux de neurones constituant l'état de l'art actuel dans le domaine de la classification d'images, et étudier comment ces réseaux pouvaient être

améliorés. Un autre stagiaire étant arrivé courant juillet, il a été décidé que je continuerais de travailler sur la partie théorique tandis que le nouveau stagiaire travaillerait sur la partie applicative ainsi que sur la rédaction de la potentielle publication. J'ai donc été amené à travailler sur différents réseaux de neurones, principalement sur les Residual Networks, introduits pour la première fois dans une publication de Microsoft Research datant du 10 décembre 2015.

Nous reviendrons plus en détails sur les différents réseaux et paramétrages qui ont été utilisés dans le chapitre 3.

Chapitre 2

Méthodologie utilisée

Dans ce chapitre, nous allons présenter les éléments de méthodologie qui ont été utilisés pendant le stage. Il s'agira de parler des outils et librairies utilisés, ainsi que des conditions de collaboration avec les autres membres de l'équipe

2.1 Outils

2.1.1 Git

Dès notre arrivée dans les locaux de l'entreprise, un compte git nous a été créé. N'ayant pas l'habitude d'utiliser cet outil, nous avons reçu une formation nous permettant d'en connaître les bases.
Git est un outil de gestion de versions. Il permet de garder une sauvegarde de ses fichiers sources, tout en mémorisant l'évolution des versions. De plus, il permet un travail en collaboration avec d'autres personnes travaillant sur le même projet grâce à son système de branches. Ainsi, plusieurs personnes pourront travailler en même temps sur un même projet tout en évitant les conflits de version.

2.1.2 Librairies

Le langage exclusivement utilisé lors de ce stage était Python (dans sa version 2.7 pour la grande majorité). Ainsi nous allons présenter les principales librairies python utilisées au cours de ces 6 mois :

Numpy Numpy est une librairie Python très utilisée dans le calcul scientifique. Elle permet, grâce à ses nombreuses fonctions prédéfinies et optimisées, le traitement et le calcul sur de "gros" vecteurs multi-dimensionnels. Elle a été utilisée dans les différentes implémentations que nous avons testées, aussi bien sur Theano que Tensorflow.

Theano Theano est une librairie Python ayant pour but de faciliter la pratique du Deep Learning. Ce projet a été fondé par l'Université de Montréal, où travaille actuellement Yoshua Bengio, un des précurseurs du deep learning.
La prise en main de cette librairie n'a pas été aisée. Pour cause, elle utilise le principe des graphes computationels, qui sont des représentations symboliques de l'architecture du réseau. De plus, cette librairie reste tout de même assez bas niveau et la compréhension du code brut est d'autant plus difficile. Ainsi, la prise en main de cette librairie diffère de tout ce que j'avais eu l'occasion de voir par le passé.
Pour palier à cette difficulté de prise en main dûe au bas niveau de cette librairie, certaines surcouches existent. Une des plus connues, et que j'ai eu l'occasion d'utiliser pendant ce stage, est la surcouche *Lasagne*. Cette surcouche permet de simplifier la syntaxe et donc l'implémentation de réseaux de neurones en utilisant Theano.
Pour l'optimisation de la vitesse de calcul, nous avons utilisé CUDA, un pilote de NVIDIA permettant d'effectuer les calculs sur GPU. Nous avons également utilisé cuDNN, une librairie d'optimisation développée par Nvidia spécialement pour le Deep Learning.

Tensorflow Tensorflow est une autre librairie dominante dans le domaine du Deep Learning. Plus récente que Tensorflow, elle a été créée et est actuellement gérée par Google. Elle est encore en phase de développement, de nombreuses features étant ajoutées régulièrement. De plus, Google a fait le choix de l'open sourcer.
Comme Theano, Tensorflow utilise le système des graphes computationels. L'architecture du réseau est représentée par des noeuds, chaque noeud correspondant à une opération mathématique.
A la différence de Theano, Tensorflow est censé permettre le parallélisme de modèle : à condition de disposer de plusieurs GPU, on peut alors créer des réseaux plus profonds sans risque de manquer de mémoire sur les GPU. De plus, Tensorflow est plus rapide que Theano pour les Convolutional Neural Network (CNN). Enfin, Tensorflow est livré avec la feature TensorBoard, qui permet de surveiller les paramètres d'un réseau de manière

très simple et lisible, ainsi que de visualiser l'architecture du réseau, ce qui n'est pas possible sur Theano (nous verrons cela dans la suite du rapport).
L'inconvénient majeur de Tensorflow est que, dû à sa mise à disposition récente, cette librairie souffre pour le moment d'un certain manque de fonctionnalités. Néanmoins, ce problème devrait être réglé sous peu.

2.2 Collaboration au sein de l'équipe

Réunions hebdomadaires Avec l'augmentation du nombre de stagiaires, un système de réunion hebdomadaire a été instauré. Ces réunions ont permis de suivre l'avancement du projet de chaque stagiaire et de définir les objectifs pour la semaine à venir. Pour ma part, ces réunions étaient individuelles car je n'ai pas eu l'occasion de travailler sur un certain projet en collaboration avec d'autres stagiaires.

Partage des serveurs de calcul Nous disposions pour notre travail d'un accès à un serveur de calcul contenant 4 GPU d'une mémoire de 12 Go chacun. Etant jusquà 5 personnes à travailler sur ce serveur, il nous a fallu nous adapter et trouver des solutions pour que chacun ait accès à un GPU. Cela n'a pas toujours été évident car les tests étaient nombreux, longs (plusieurs jours d'entrainement pour les modèles les plus profonds) et nécessitaient parfois l'utilisation de plusieurs GPU.

Chapitre 3

Le Deep Learning appliquée à la classification d'images

Dans cette partie, nous allons aborder certains aspects de la théorie du Deep Learning afin d'expliciter des concepts nécessaires à la compréhension du travail réalisé.

3.1 Le problème de la classification d'images

La classification d'images est un problème traité depuis des décennies dans le domaine de l'intelligence artificielle. Les techniques ont beaucoup évolué, depuis la fin des années 90 à aujourd'hui.
Ce problème consiste à repérer, dans une images, certains éléments ou caractéristiques tels qu'un animal, un être humain ou encore une voiture.

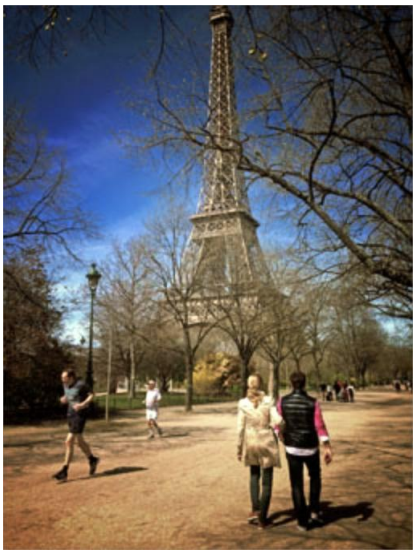

FIGURE 3.1: Exemple de classification d'image

Sur cette photo, nous aimerions par exemple détecter la présence d'êtres humains, des arbres ou de la Tour Eiffel. Mais nous aimerions également détecter des éléments de contexte comme par exemple, le fait que nous nous trouvons dans un parc.

3.2 Les réseaux de neurones

Nous allons à présent aborder les réseaux de neurones : de quoi sont-ils composés, quel est leur intérêt et comment fonctionnent-ils ?

3.2.1 Les neurones computationnels

Un neurone computationnel est la modélisation informatique du neurone bioologique comme nous le comprenons aujourd'hui.

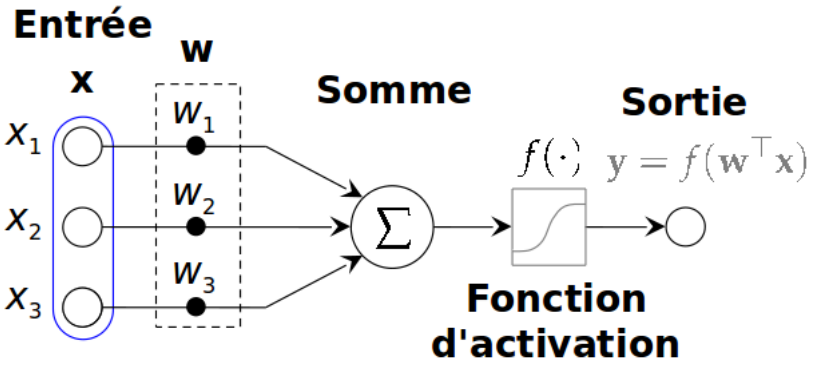

FIGURE 3.2: Neurone computationnel

Sur cette figure, nous avons trois arguments x_1, x_2, x_3, et trois paramètres w_1, w_2, w_3. Le neurone va calculer une combinaison linéaire de ces paramètres, puis en faire la somme. Cette combinaison va ensuite passer à travers une fonction d'activation de laquelle on pourra récupérer la sortie du neurone. En quelques sortes, l'on peut considérer que le neurone fait la synthèse des informations qu'on lui envoie, puis renvoie lui même cette synthèse à d'autres neurones.

3.3 Les réseaux de neurones

Un réseau de neurones va alors être la combinaison de plusieurs neurones, qui vont communiquer entre eux certaines informations afin d'en tirer les conclusions voulues :

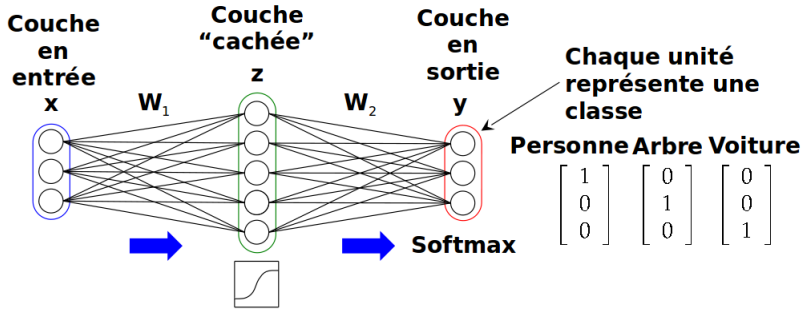

FIGURE 3.3: Réseau de neurones composé de trois couches

Dans cet exemple, le but du réseau va être de détecter si l'élément dans l'image en entrée est une voiture, une personne ou un arbre. La dernière couche, appelée *Softmax*, aura pour rôle de tirer les conclusions des calculs effectués en amont dans le réseau.

Maintenant que nous savons de quoi est composé un réseau de neurones, il va être

question de l'entraînement. L'entraînement d'un réseau de neurones aura pour but de trouver les bon paramètres W_i afin que le réseau soit aussi performant que possible. En apprentissage supervisé, nous allons "nourrir" le réseau avec de nombreuses images labellisées à partir desquelles il va apprendre et s'améliorer.

Lorsque nous allons envoyer une image dans le réseau, celui ci va comparer le résultat en sortie avec le résultat attendu. En utilisant une fonction de perte, le réseau va ensuite utiliser la "backpropagation" : il va faire remonter l'erreur à travers le réseau, et ainsi ajuster les poids afin de minimiser la fonction de perte. Le but de l'entraînement est donc d'ajuster les poids afin d'atteindre le minimum de la fonction de coût, ou du moins un minimum local qui est assez proche du minimum global.

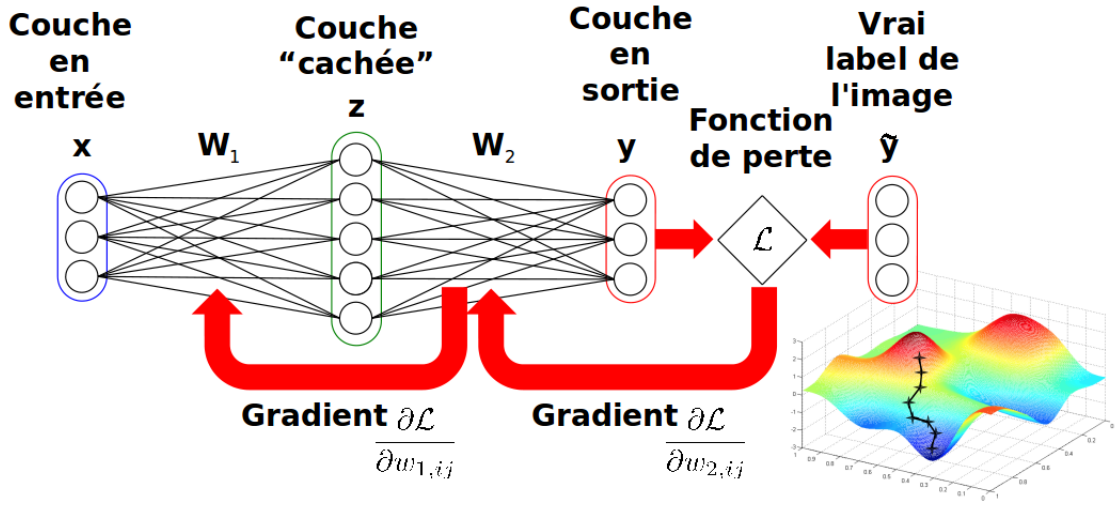

FIGURE 3.4: Entrainement du réseau

En pratique, les réseaux de neurones auront plus d'une couche "cachée", ou "hidden layer". Un des enjeux actuels est d'ailleurs d'augmenter la profondeur de ces réseaux afin d'obtenir de meilleures performances. Nous expliquerons dans la suite quel est l'intérêt des réseaux plus profonds, quelles en sont les limitations et comment essayer de les contrer.

3.4 Les Convolutional Neural Networks

Les Convolutional Neural Networks (CNN) sont, comme leur nom l'indique, des réseaux de neurones composés de couches convolutionnelles. Ils sont très utilisés dans le secteur de la reconnaissance d'images. En effet, ces architectures partent du principe que les

inputs sont des images, ce qui permet d'en exploiter certaines propriétés.
Les différentes couches d'un tel réseau vont avoir pour effet de calculer un produit de convolution entre une certaine couche et des filtres qui seront en réalité des matrices dont les coefficients seront des paramètres que l'on ajustera au fil de l'entrainement, et que l'on appellera les poids du réseau.
Dans les premières couches du réseau, et donc après seulement quelques convolutions, le modèle obtiendra une représentation assez "réaliste" de l'image qu'il voit en entrée. Plus le réseau sera profond, plus le modèle aura une représentation abstraite de l'image qui lui a été donné. C'est ainsi que le modèle va pouvoir déterminer des "patterns" qui lui serviront à reconnaitre le contenu d'une image.
L'objectif d'un CNN va être de calculer, convolution après convolution, des représentations de plus en plus abstraites des images qu'il reçoit. C'est ainsi que le réseau va être capable de déterminer des "patterns" qu'un humain ne pourrait obtenir et qui lui permetterait de classifier les images.
Les CNN représentent depuis maintenant plusieurs années l'état de l'art dans le domaine de la classification d'image. Néanmoins il existe de nombreuses architectures utilisant ces réseaux. Dans la suite de ce rapport, nous allons en présenter certaines que nous avons eu l'occasion de tester au cours de ces 6 mois.

Chapitre 4

Différentes architectures testées

Dans cette partie, nous allons présenter le travail effectué durant ces 6 mois, ainsi que la théorie associée aux recherches effectuées. Nous présenterons les étapes du travail dans l'ordre chronologique et expliquerons au fur et à mesure les raisons qui nous ont motivés à suivre la direction choisie.
Nous rappelons que ces recherches ont été effectuées dans le cadre du projet de classification d'images satellites présenté au début de ce rapport.

4.1 Présentation théorique des ResNet

Les ResNet ont fait leur appartition pour la première fois dans la publication de Microsoft Research : *Deep Residual Learning for Image Recognition*[1]. Dans cette partie nous allons premièrement présenter la théorie qui se trouve derrière cette architecture, puis nous verrons quels résultats il a été possible d'obtenir avec les différentes implémentations utilisées.
A force de constater l'augmentation incessante de la profondeur des réseaux de neurones constituant l'état de l'art actuel dans le secteur de la reconnaissance d'images, l'on pourrait aisément penser que plus un réseau est profond, meilleur il sera. Néanmoins, il a été prouvé empiriquement que ce n'est pas le cas : à partir d'une certaine profondeur, la précision a tendance à saturer et à se dégrader très rapidement.

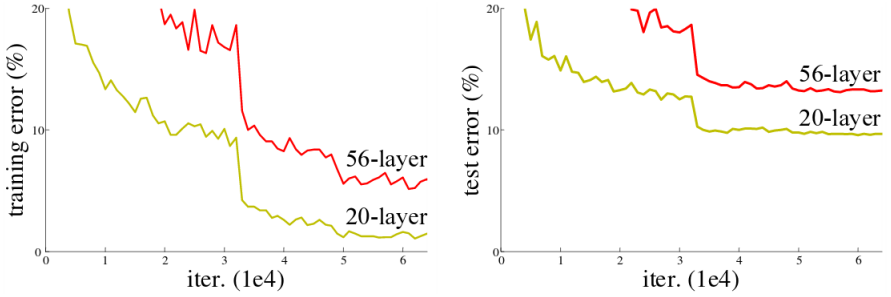

FIGURE 4.1: Différence de training error et de validation error avec des réseaux composés de 20 et 56 couches

Residual Blocks C'est de ce constat qu'est née l'idée du Residual Block, et ainsi, l'idée du Residual Network.

En raison de leur caractère non-linéaire, il est beaucoup plus difficile pour une couche

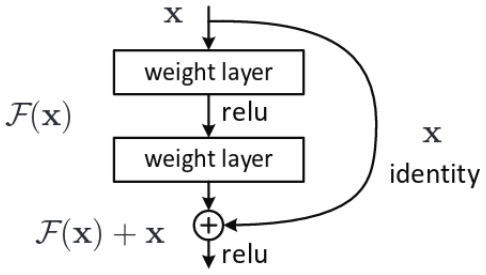

FIGURE 4.2: Illustration d'un Residual Block

du réseau de calculer la fonction identité que de calculer la fonction "zero mapping". En effet, pour calculer cette dernière il suffirait de fixer tout les poids à 0.

Nous allons alors décomposer le réseau en plusieurs Residual Blocks (cf figure 3.2). Fixons x l'entrée du Residual Block, et $H(x)$ la sortie que l'on désire obtenir.

Fixons :

$$H(x) = F(x) + x$$

$F(x)$ sera appelé résidu du block. Nous créons désormais une "shortcut connection", ou "raccourci", entre l'entrée et la sortie du block. Ce raccourci aura pour but de transférer l'entrée à la sortie (il représente donc la fonction identité). Alors, le block n'aura plus à calculer la sortie entière $H(x)$, mais le résidu $F(x)$ qui lui est beaucoup plus simple à calculer.

Le but de ce type de réseau est de réduire l'erreur de calcul sur chaque block et ainsi

réduire les erreurs de calcul qui s'accumuleraient sur l'ensemble du réseau. C'est pourquoi nous pouvons finalement gagner en profondeur tout en continuant à gagner en précision.

4.2 Implémentation en Theano, résultats attendus et résultats obtenus

Après avoir étudié la théorie de ce type de réseau, nous sommes passés à l'implémentation afin d'effectuer les tests nécessaires. Fait courant en recherche, nous sommes repartis d'une implémentation déja existante et disponible sur Github à l'adresse suivante : https://github.com/Lasagne/Recipes/blob/master/papers/deep_residual_learning/Deep_Residual_Learning_CIFAR-10.py En effet, il est fastidieux d'implémenter un réseau de neurones en partant de zero et cela nous ferait commencer les tests plus tard, ce qui n'a donc aucun intérêt.

CIFAR-10 Dans cette section nous allons présenter le dataset sur lequel nous avons commencé notre phase de tests.
CIFAR-10 est un dataset composé de 60000 images réparties en 10 classes : avion, voiture, camion, chat ... Il est divisé en un training set de 50000 images et un validation set de 10000 images. Les images sont de taille 32*32*3 (car au format RGB).

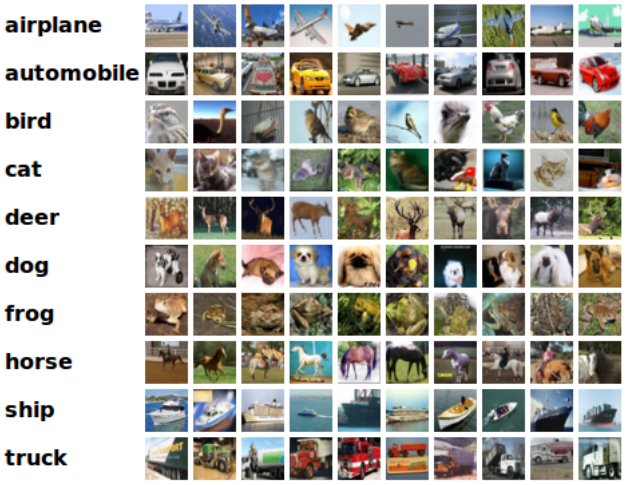

FIGURE 4.3: Dataset CIFAR-10

Architecture Appliqué au dataset CIFAR-10, ce réseau va donc prendre en entrée des images 32*32. L'objectif de l'architecture est de, petit à petit, réduire la spatialité

des images tout en augmentant le nombre de "feature maps" disponibles. Autrement dit, nous allons tenter d'avoir un nombre important de représentations abstraites à partir desquelles le réseau déterminera les "patterns" nécessaires à la classification. Ainsi, au début du réseau nous allons trouver 16 filtres par couche. Nous avons donc un vecteur de taille 32*32*16. Après un nombre n de residual blocks (qui fait partie des paramètres ajustables du système), nous allons diviser la taille des images par 4, tout en multipliant le nombre de filtres par 2. Nous avons alors un vecteur de taille 16*16*32. Toujours après un nombre n de residual blocks, nous allons réitérer l'opération. Alors, nous obtenons un vecteur de taille 8*8*64.
Après n derniers residual blocks, nous allons trouver un *Average Pool Layer* qui va calculer la moyenne de chacune des 64 représentations 8*8. Enfin, un dernier layer, appelé *Softmax*, va prendre le vecteur résultant en entrée et retourner un vecteur dont la taille sera équivalente au nombre de classes du dataset. Pour chaque classe i, ce layer va attribuer à l'image un score entre 0 et 1 qui correspondera à la probabilité pour l'image d'appartenir à la classe i. Le résultat de la classification sera donc logiquement la classe qui maximise cette probabilité.

Résultats obtenus sur CIFAR-10 Pour l'entraînement sur CIFAR-10, nous avons utilisé des techniques classiques d'augmentation de données :

- Le flip horizontal, qui consiste à prendre une image et en extraire son miroir
- Le zero-padding + random crop : le zero-padding consiste à rajouter des 0 sur les bords de l'image. Dans notre cas, nous avons rajouté quatre 0 sur chaque extremité de l'image, de façon à obtenir des images 40*40. Vient ensuite le random crop, qui consiste à extraire au hasard une image de taille 32*32 au sein de l'image 40*40.

Ces techniques d'augmentation de données ont pour but d'amener plus de variétés dans le dataset. Ainsi, le risque d'overfitting est moins élevé, et le réseau apprend mieux.
Après avoir apporté les différentes modifications nécessaires au code (définition des différents paramètres, récupération des outpus désirés etc...), nous avons lancé l'entrainement et avons rapidement obtenu des résultats similaires à ceux obtenus dans le papier Microsoft. Voici par exemple les résultats obtenus avec 20, 32 et 50 couches :

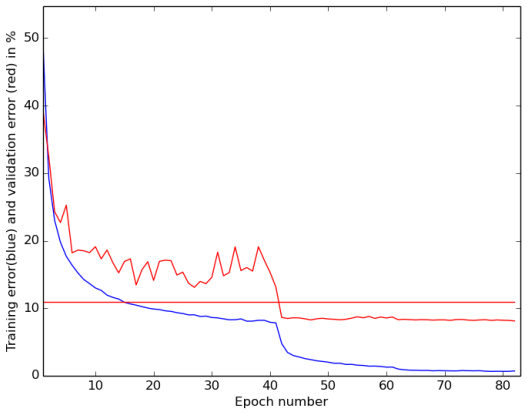

FIGURE 4.4: Résultats pour 20 couches - 91,9% de validation accuracy

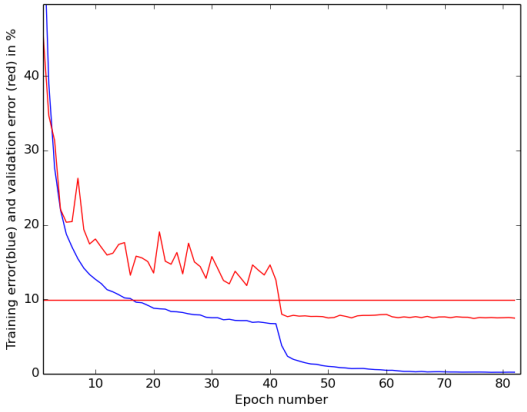

FIGURE 4.5: Résultats pour 32 couches - 92,5% de validation accuracy

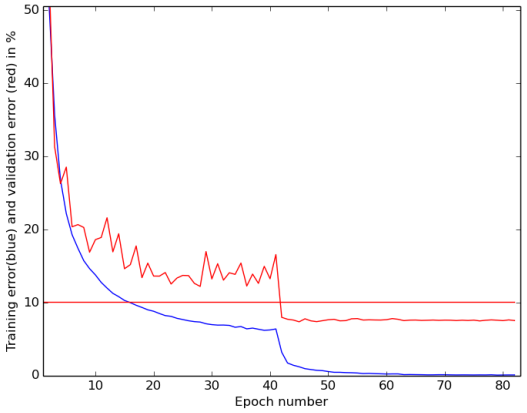

FIGURE 4.6: Résultats pour 50 couches - 92,4% de validation accuracy

Après avoir bien vérifié que notre réseau fonctionnait comme prévu, nous avons décidé de l'appliquer à un autre dataset qui va être présenté dans le paragraphe suivant.

Dataset X Dans le cadre d'un des projets de Krakatoa, nous avons eu l'occasion de tester ces réseaux sur un autre type de dataset. Pour des raisons de confidentialité nous ne pourrons explicitement dévoiler ce dataset, mais nous pouvons considérer les caractéristiques suivante :

- Le dataset est composé de 4000 images, divisées en 4 classes
- Les images sont de type grayscale (nuances de gris), et de taille 150*150
- Les images ont été extraites d'un cube en 3 dimensions, mais on ne connait pas leurs coordonnées dans ce cube.

Nous avons utilisé pour ce dataset des techniques d'augmentation similaires à celles utilisées pour le dataset CIFAR-10.

Les premiers tests sur ce dataset n'ont pas été concluants. Afin de comprendre les raisons de cet échec, nous avons ajouté au code le calcul de deux Key Performance Indicator : la precision et le recall.

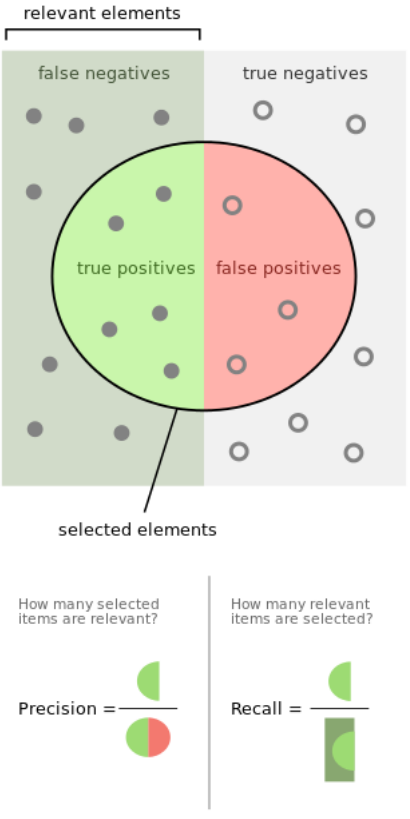

FIGURE 4.7: Deux indicateurs de performance : precision / recall

Pour une classe i, nous divisons les résultats de la classification en 4 catégories : les vrais positifs, les faux positifs, les vrais négatifs et les faux négatifs.

La précision correspond à la proportion d'éléments appartenant vraiment à la classe i parmi ceux que le réseau a attribué à cette classe :

$$Precision = \frac{VraisPositifs}{VraisPositifs + FauxPositifs}$$

Le recall correspond quant à lui à la proportion de vrais positifs parmi le nombre d'éléments appartenant réellement à la classe i :

$$Recall = \frac{VraisPositifs}{VraisPositifs + FauxNegatifs}$$

Ces indicateurs permettent d'étudier les différences de résultat entre les différentes classes. Ainsi, nous avons pu trouver grâce à eux pourquoi le réseau ne fonctionnait pas comme il fonctionnait sur CIFAR-10 : le dataset n'était pas correctement mélangé. Le réseau apprenait majoritairement sur seulement 3 classes, et la validation se faisait principalement sur la 4ème.

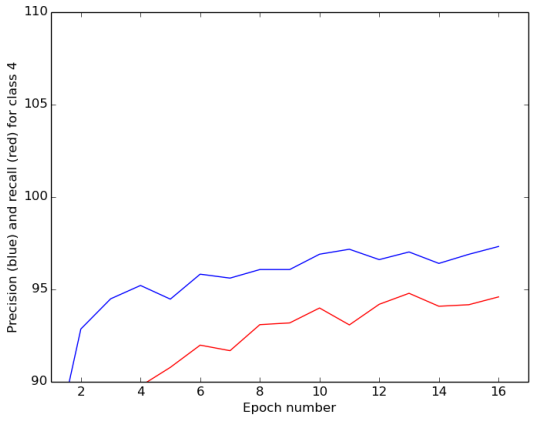

FIGURE 4.8: Courbes représentant la precision et le recall pour la classe 4

Après avoir réglé ce problème, nous avons pu réiterer l'expérience. Avec un réseau de 32 couches, nous avons obtenus les résultats suivants :

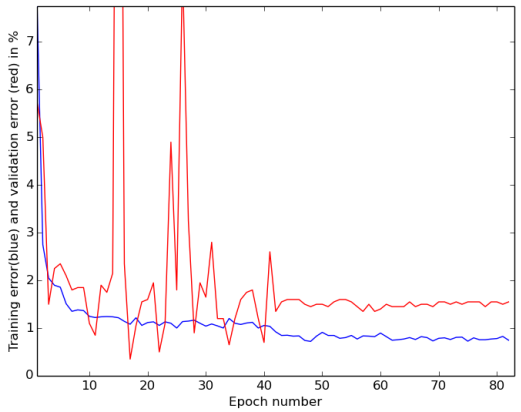

FIGURE 4.9: Courbes d'apprentissage pour le Dataset X

Avec une training error de 99% et une validation error de 98,45%, nous aurions pu penser que le réseau marchait au-delà de nos espérances. Mais la courbe d'apprentissage parait peu conventionnelle. La validité du dataset a donc été remise en cause : a-t-on suffisamment d'image, de diversité etc ..

Afin d'obtenir plus de diversité, nous avons essayé de localiser les images labellisées dans le cube à partir desquelles elles ont été extraites. Ainsi, en trouvant dans quelles régions du cube les images ont été extraites, nous pourrions peut-être extraire plus d'images correspondant à cette classe dans un certain voisinage à déterminer.

Pour ce faire, nous avons implémenté des scripts de recherche d'image à travers le cube. A noter que le cube était de dimension 450*600*900

Le fonctionnement était le suivant : pour chaque image de taille 150*150 parmi les 4000 images labellisées, nous parcourons le cube selon l'axe y. Nous calculons la distance (L2) entre l'image labellisée et l'ensemble des images 150*150 que l'on peut extraire du cube. L'image du cube minimisant cette norme est alors l'image désirée, en théorie.

Après avoir testé cet algorithme, nous nous sommes rendus compte que les images labellisées avaient subi un traitement dont nous ne connaissions pas les caractéristiques. Nous avons alors cherché dans le cube, à la main, une image labellisée choisie, et avons effectué les modifications nécessaires sur l'image extraite du cube afin d'obtenir la même moyenne et le même écart-type que sur l'image labellisée. De cette manière, nous avons récupéré les paramètres nécessaires à la transformation et avons appliqué cette dernière à l'ensemble du cube. Nous avons alors réussi à récuperer les coordonnées de chaque image labellisée au sein du cube.

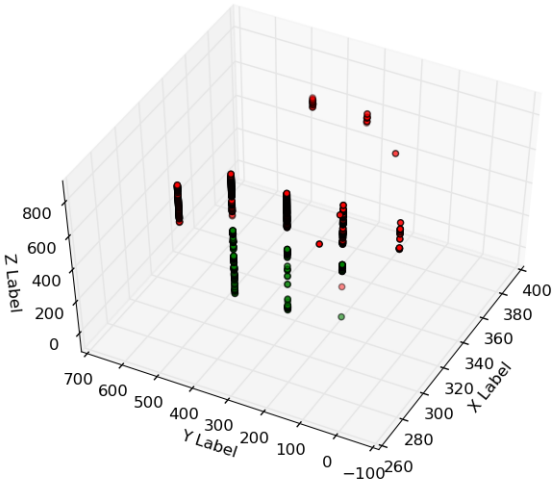

FIGURE 4.10: Localisation des éléments de la classe 2 au sein du cube

Après avoir obtenu ces coordonnées, nous avons décidé de tester la robustesse du dataset et du réseau d'une nouvelle façon : définir le training set et le validation set non plus au hasard, mais en sélectionnant des zones spécifiques. Le problème était en effet que, parmis les images d'une certaine classe, beaucoup se suivaient dans le cube et se ressemblait donc fortement. Si plusieurs images se ressemblant se trouvaient à la fois dans le training set et le validation set, il était donc normal que le modèle ait une bonne validation accuracy. Il n'y avait à proprement parler plus vraiment de traing set et de validation set distincts, uniquement un dataset sur lequel le modèle s'entrainait et était testé en même temps.

Une fois que nous disposions des coordonnées à partir desquelles les images labellisées avaient été extraites, nous pouvions changer nos techniques d'augmentation de donnée :

— A la place du zero padding + random crop, nous avons implémenté une fonction qui, à partir des coordonnées d'une image, en extrait une autre dans un voisinage à définir (en général il s'agissait d'un décalage allant de 20 à 50 pixels)

— Nous avons rajouté une fonction qui multipliait l'amplitude des pixels par un facteur aléatoire, choisi dans un interval déterminé au préalable (en général [0,9 ;1,1]

Nous avons tout d'abord entrainé le modèle avec de nombreux paramètres différents (profondeur, augmentation de données etc..) et avons obtenu les résultats suivants, toujours en définissant le training set et le validation set de manière aléatoire mais en

ajoutant de l'augmentation de données :

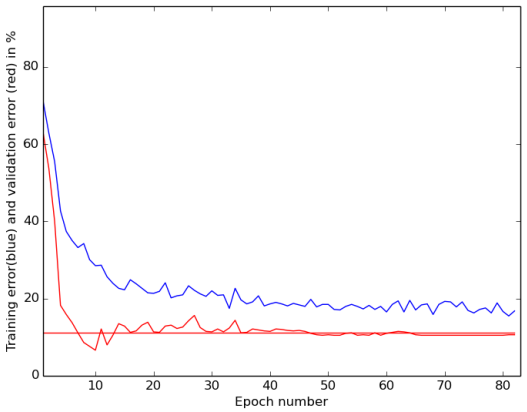

FIGURE 4.11: Résultats obtenus avec 20 couches - 89,4 % de validation accuracy

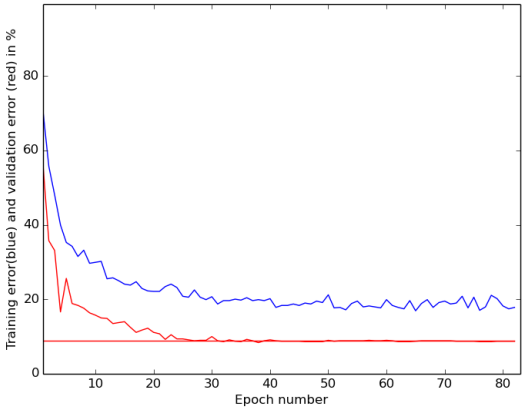

FIGURE 4.12: Résultats obtenus avec 32 couches - 91,5 % de validation accuracy

Sur ces figures, l'on se rend compte que contrairement à d'habitude, la validation accuracy est meilleure que la training accuracy. Cela est probablement du à la forte augmentation de donnée qui a été appliquée.
Nous avons ensuite réitéré ces expériences avec cette fois une séparation déterministe entre le training set et le validation set : nous avons choisi deux zones distinctes afin d'être sûrs que des images se suivant dans le cube et donc quasi similaires ne se retrouvent pas dans les deux sets. Les résultats ont été les suivants :

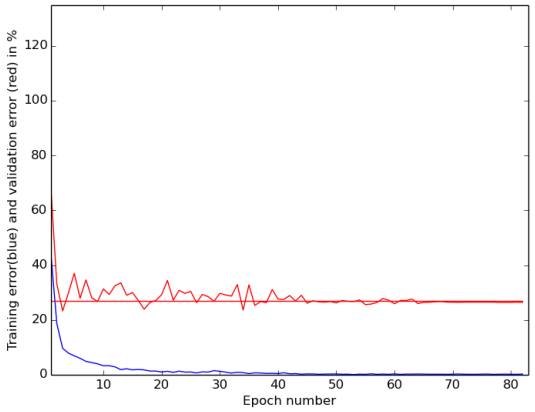

FIGURE 4.13: Résultats obtenus avec 8 couches

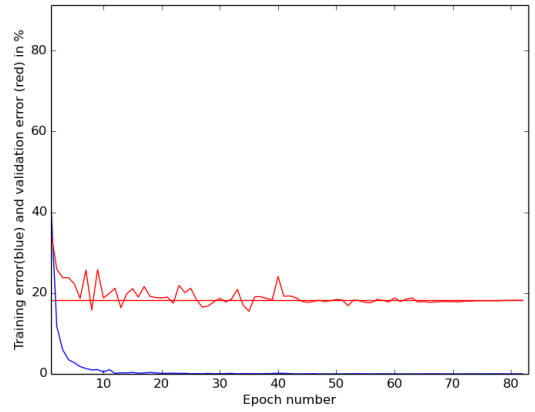

FIGURE 4.14: Résultats obtenus avec 20 couches

Nous voyons très clairement sur ces graphiques que, même avec un nombre minimal de couches, le réseau va très vite sur-apprendre, il overfit. Nous en arrivons donc à nous demander si le dataset est tout simplement exploitable. La décision est donc prise de tester une nouvelle architecture, en la couplant à l'architecture des ResNet : nous allons utiliser une profondeur stochastique dont nous allons présenter l'intérêt dans la partie suivante.

4.3 Stochastic ResNet

Le concept de profondeur stochastique a fait son apparition en mars 2016 dans la publication *Deep Networks with Stochastic Depth* [2]. Le concept est le suivant : dans un réseau, nous allons attribuer à chaque couche une "probabilité de survie". Cette probabilité va être linéairement décroissante, fonction de la profondeur, commençant à 1 pour la première couche et allant jusqu'à une certaine probabilité fixée pour la dernière couche (dans le cadre des expériences, cette probabilité a été fixée à 0,5).
Lors de l'entrainement, et à chaque itération, chaque couche aura une probabilité correspondant à la probabilité qui lui a été attribuée d'être prise en compte. Si elle n'est pas prise en compte, le réseau va simplement sauter cette couche. La sortie calculée par la couche l du réseau est donc :

$$H_l = p_{survie,l} * f_l * H_{l-1} + Id(H_{l-1})$$

avec f_l le résidu calculé par la couche l et H_{l-1} la sortie de la couche l-1 et donc l'entrée de la couche l.

Cette structure va permettre deux améliorations principales :

- Au niveau de la performance : lors de l'entrainement, à chaque itération, le réseau va avoir une strucutre différente. Il est alors plus difficile pour lui d'apprendre par coeur, ou encore d'overfiter. Cela se vérifie dans les performances annoncées dans le papier.
- Au niveau du temps de calcul : le réseau va passer par moins de couches, le calcul sera donc plus rapide. En choisissant une probabilité de survie de 0.5 pour la dernière couche, on montre que le réseau passe par un nombre de couches équivalent à 3/4 du nombre initial de couches. Ainsi, nous ferons une économie de 25% du temps de calcul sur l'entraînement.

Les résultats des expérimentations n'ont pas été concluant. Pour une profondeur de 32 couches, nous avons obtenu les résultats suivants :

- Non stochastique : 92,61% de validation accuracy
- Stochastique : 92,86% de validation accuracy

Comparée aux résultats du papier, cette amélioration n'est pas significative. Elle peut être simplement dûe à un split différent entre le training set et le validation set.

Nous en sommes alors arrivés à la conclusion que pour que le caractère stochastique soit effectif, il fallait que la profondeur soit plus importante. Mais sur Theano, nous étions limités en mémoire. En effet, Theano ne permet pas le parallélisme de modèle. Ainsi, l'architecture entière du réseau, incluant les millions de paramètres qui lui sont associés, doivent tenir en mémoire sur un seul GPU. Nous avons donc décidé de tester une autre librairie, permettant quant à elle le parallélisme de modèle : Tensorflow.

4.4 Tensorflow

De même que pour l'implémentation Theano, nous avons repris une implémentation déja existante de ResNet sur Tensorflow.
Premièrement, nous avons eu l'occasion de nous familiariser avec l'interface graphique Tensorboard, qui permet de récupérer de nombreux paramètres en sortie très facilement.

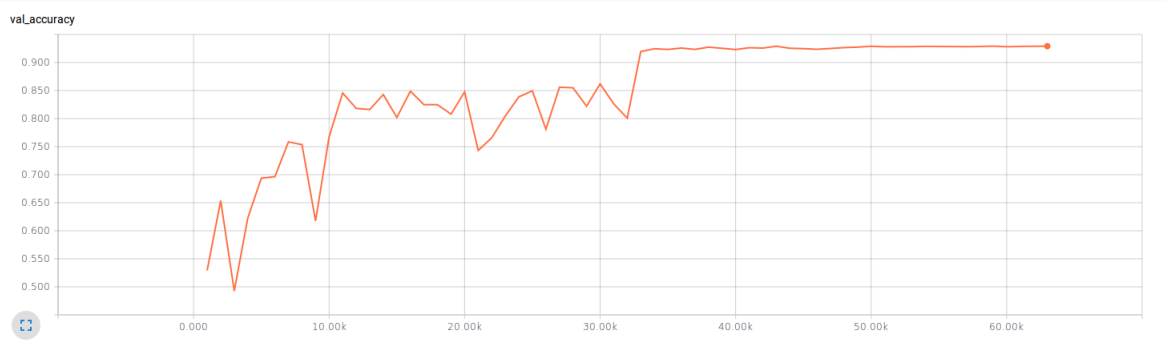

FIGURE 4.15: Exemple d'output récupéré sur Tensorboard : validation accuracy pour un réseau de 50 couches

De plus, cet outil nous permet de visualiser l'achitecture du réseau, et ainsi de vérifier notre implémentation.

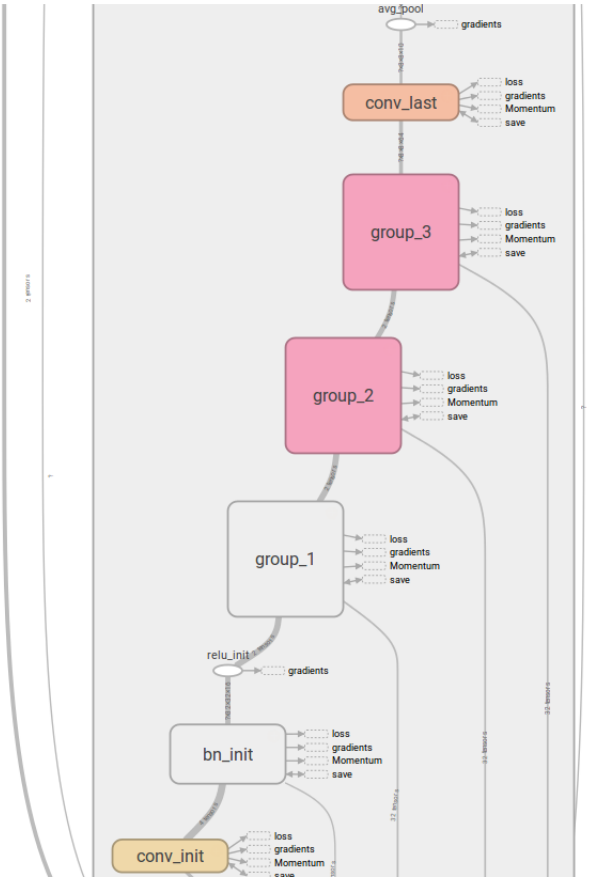

FIGURE 4.16: Architecture globale du réseau

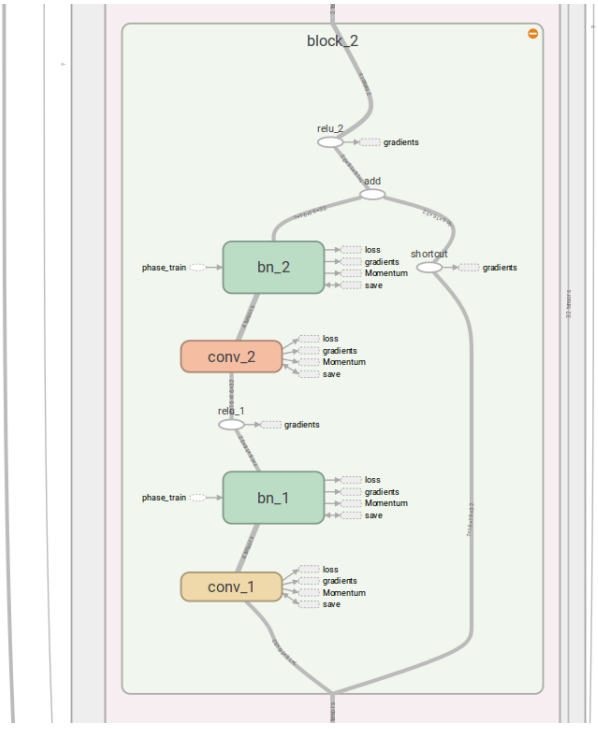

FIGURE 4.17: Architecture d'un Residual Block

Avec une profondeur de 50 couches, et en faisant varier les paramètres, le meilleur résultat obtenu en terme de validation accuracy a été de 92,9%.
Nous avons alors implémenté une fonction permettant au réseau d'adopter une profondeur stochastique.

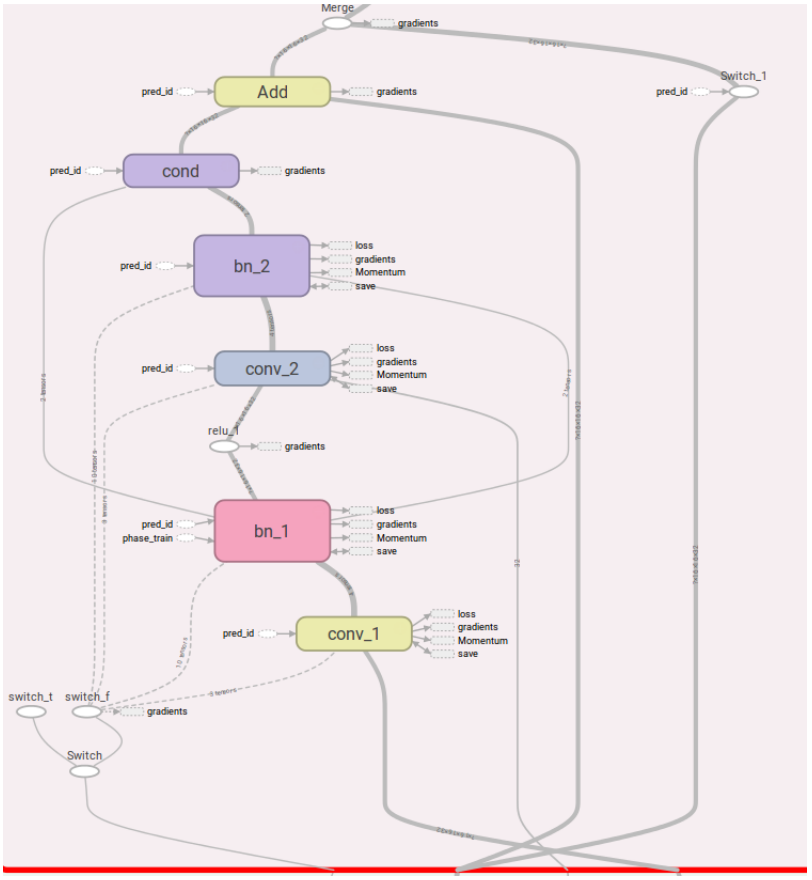

FIGURE 4.18: Architecture d'un Residual Block dans sa version stochastique

Sur cette figure, nous voyons les différents chemin que peuvent suivre les inputs entrant dans le Resblock : si le ResBlock "survit", l'input va passer par le chemin de gauche qui correspond au chemin classique du ResBlock. Si par contre le ResBlock ne "survit" pas, l'input va passer par le chemin de droite, qui est simplement un chemin correspondant à la fonction identité.

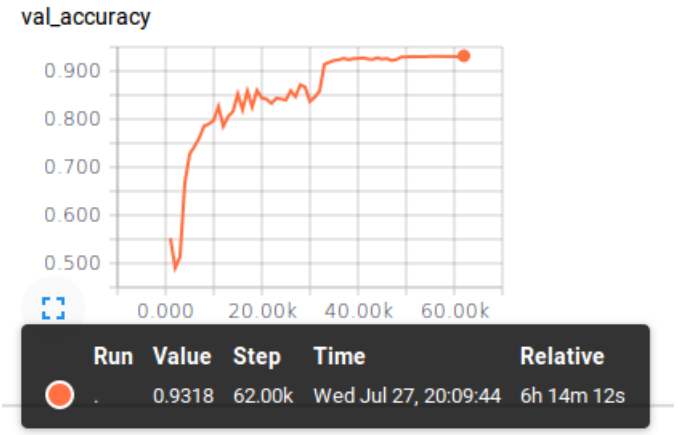

FIGURE 4.19: Validation accuracy pour un Stochastic Resnet de 50 couches : 93,18%

Après avoir réitéré les mêmes expériences que celles effectuées avec l'implémentation Theano, nous avons tenté d'expérimenter des réseaux beaucoup plus profonds. Nous avons alors fait deux constats :

— La mémoire n'est pas gérée de la même façon en Theano et en Tensorflow. Alors que Theano adapte la mémoire qu'il utilise à ses besoins et laisse le reste de la mémoire du GPU disponible, Tensorflow va réserver toute la mémoire disponible sur le GPU, même s'il n'utilisera qu'une partie de celle-ci. Cela a pour but d'optimiser la vitesse de calcul, mais peut également être problématique lorsque la mémoire GPU doit être partagée avec d'autres personnes.

— Sans passer par le multi-GPU, Tensorflow permet d'utiliser des réseaux beaucoup plus profonds. Nous avons pu en effet réaliser une expérience avec un réseau de 600 couches tandis que sur Theano nous étions limités à 100 couches.

En expérimentant sur ce réseau de 600 couches, nous n'avons pas obtenu les résultats escomptés :

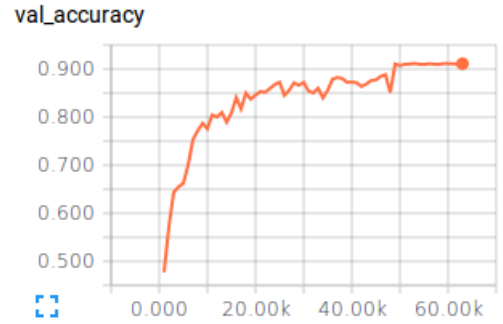

FIGURE 4.20: Validation accuracy pour un Stochastic Resnet de 600 couche : 91,12%

Malgré de nombreux efforts, nous n'avons pu trouver ce qui nous empêchait de trouver des résultats similaires à ceux du papier *Deep Networks with Stochastic Depth* [2]. Nous avons alors décidé, pour le dernier mois de stage, de travailler sur des éléments récents de la littérature que nous allons présenter.

4.5 Residual Network of Residual Network

Notre attention s'est tout d'abord portée vers un article abordant de nouveau le thème des Residual Networks : *Residual Networks of Residual Networks : Multilevel Residual Networks* [3], publié en Août 2016.

Dans cet article, l'auteur propose de généraliser le concept de Residual Block au sein de l'architecture du réseau. Dans l'architecture présentée, nous reprenons le modèle du ResNet, et rajoutons des shortcut connections sur plusieurs niveaux. L'architecture résultant ressemble alors à cela :

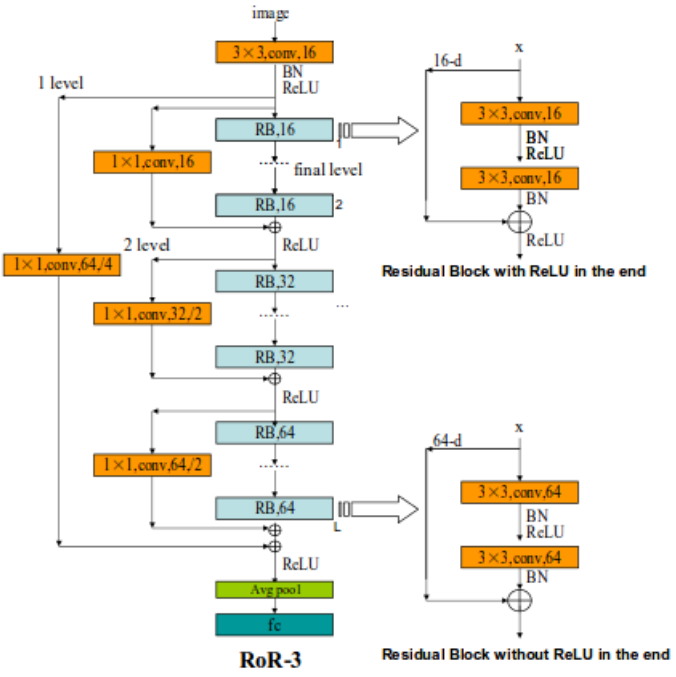

FIGURE 4.21: Residual Network of Residual Network (RoR) de niveau 3

Cette architecture est appellée RoR de niveau 3 car on retrouve des shortcut connections sur 3 niveaux différents :

- Au sein même des ResBlock
- Autour des ResBlock de même taille
- Entre la première et la dernière couche du réseau

L'auteur du papier annonce des résultats qui constituent l'état de l'art actuel sur différents datasets, notamment CIFAR-10 sur lequel ce réseau obtiendrait un score de 96,23% de validation accuracy.

4.6 Dernières expériences

Durant les quelques dernières semaines de stage, nous avons pu aborder une nouvelle structure de réseau sur laquelle un stagiaire a eu l'occasion de travailler : les Variational Auto Encoder - Generative Adversorial Networkles, ou encore VAE-GAN. Ces deux réseaux, mis bout à bout, vont s'entraîner mutuellement.

La structure et le fonctionnement du réseau sont présentés dans le papier : *Autoencoding beyond pixels using a learned similarity metric* [4]. Premièrement, le VAE aura pour rôle d'encoder et de décoder des images. Le résultat sera ensuite envoyé dans le GAN,

qui lui, aura pour rôle de déterminer si l'image est une image d'origine ou si elle a été artificiellement reconstruite.

Cette structure obtient de bon résultats dans le domaine de la génération d'images. Elle peut être appliquée, par exemple, à la génération de visages. Les expériences actuellement menées ont alors pour objectifs d'améliorer encore plus ces performances, notamment au niveau du VAE qui possède une marge d'amélioration plus importante que le GAN.

Conclusion

Ce stage m'a permis d'acquérir une certaine expérience dans le domaine du Deep Learning, mais également de découvrir le milieu de la recherche. Après avoir effectué une année de césure dans des domaines beaucoup moins techniques, je pense maintenant avoir un apperçu concret du monde actuel du machine learning. Les conseils de mes encadrants m'ont permis d'évoluer rapidement. Leur confiance m'a également permis de gagner en autonomie au fil des mois afin de finalement être capable de gérer seul le protocole d'expériences à mener.

Le domaine du machine learning, et spécialement celui du Deep Learning est vaste et prometteur. C'est pourquoi, à la suite de cette année de spécialisation et de ce stage de fin d'études, j'ai décidé de suivre une formation supplémentaire en Data Science et Cyber Sécurité, qui me permettra de compléter mon background technique avant d'entamer si possible une carrière dans le monde de la high-tech.

Bibliographie

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[2] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. *CoRR*, abs/1603.09382, 2016. URL http://arxiv.org/abs/1603.09382.

[3] K. Zhang, M. Sun, T. X. Han, X. Yuan, L. Guo, and T. Liu. Residual Networks of Residual Networks : Multilevel Residual Networks. *ArXiv e-prints*, August 2016.

[4] A. Boesen Lindbo Larsen, S. Kaae Sønderby, H. Larochelle, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. *ArXiv e-prints*, December 2015.