Réseaux de neurones convolutifs pour la segmentation sémantique et l'apprentissage d'invariants de couleur

Damien Fourure

N° d'ordre NNT : 2017LYSES056

THESE DE DOCTORAT DE L'UNIVERSITE DE LYON

opérée au sein de
l'**Université Jean Monnet de Saint-Étienne**
et en collaboration avec le **LIRIS**

Ecole Doctorale N°488
Sciences, Ingénierie, Santé

Spécialité de doctorat : Informatique

Soutenue publiquement le **12/12/2017**, par :

Damien FOURURE

Réseaux de neurones convolutifs pour la segmentation sémantique et l'apprentissage d'invariants de couleur

---

Devant le jury composé de :

Thierry CHATEAU	*Professeur, Université Blaise Pascal, Clermont-Ferrand*	Président
Patrick PEREZ	*Distinguished Scientist, PhD, Technicolor, Rennes*	Rapporteur
Nicolas THOME	*Professeur CNAM, Paris*	Rapporteur
Mikaela KELLER	*Maître de Conférences, INRIA Lille Nord Europe*	Examinatrice
Jakob VERBEEK	*Maître de Conférences HDR, INRIA Rhone-Alpes, Grenoble*	Examinateur
Alain TREMEAU	*Professeur, Université Jean Monnet, Saint-Etienne*	Directeur
Christian WOLF	*Maître de Conférences HDR, INSA de Lyon*	Directeur
Rémi EMONET	*Maître de Conférences, Université Jean Monnet, Saint-Etienne*	Co-encadrant
Elisa FROMONT	*Professeur, Université Rennes 1*	Invitée
Damien MUSELET	*Maître de Conférences, Université Jean Monnet, Saint-Etienne*	Invité

Fourure Damien

Réseaux de neurones convolutifs pour la segmentation sémantique et l'apprentissage d'invariants de couleur

Rapporteurs : Patrick Perez et Nicolas Thome

Examinateurs : Thierry Château, Mikaela Keller et Jakob Verbeek

Superviseurs : Alain Trémeau, Christian Wolf

Co-encadrants : Rémi Emonet, Élisa Fromont et Damien Muselet

Université Jean Monnet

Laboratoire Hubert Curien

Informatique, télécommunications et image

Data Intelligence

18 Rue du Professeur Benoît Lauras

42023 Saint-Étienne

Remerciement

Il va dans l'ordre naturel des choses, lors de l'écriture du manuscrit de thèse, de terminer par les remerciements et l'on pourrait croire, qu'après l'écriture de plus d'une centaine de pages, que ça ne soit qu'une formalité. Pourtant, je sens que cette partie sera probablement la plus difficile à écrire pour une raison simple : au cours de mes trois années de thèse, de nombreuses personnes m'ont accompagné, soutenus et même supporté et je ne veux en oublier aucune dans mes remerciements.

Pour commencer, je remercie mes encadrants, Alain, Christian, Damien, Elisa et Rémi, qui au cours de ces trois années, ont été une seconde famille. Ils ont participé à mon éducation scientifique et m'ont permis de grandir en tant que chercheur. Tout comme une famille, il y a eu des hauts et des bas, mais avec le recul je réalise que ça a toujours été pour mon bien. Un grand merci à vous. Je remercie également mes rapporteurs Patrick Pérez et Nicolas Thome pour le temps qu'ils m'ont accordé et les retours qu'ils m'ont faits. Je suis également reconnaissant envers Thierry Chateau, Mikaela Keller et Jakob Verbeek qui ont accepté d'être membre de mon jury de thèse et d'évaluer mes travaux.

Je tiens aussi à remercier ma famille, ma mère, mon père, ma sœur (et pti lulu) et mon frère et toutes les "pièces ajoutées" (comme on les appelles) qui ont cru en moi et surtout qui m'ont supporté dans mes nombreux moments de fatigue. Merci à tous d'avoir été présent dans les moments difficiles. Je remercie aussi la famille dites "éloignée", c'est-à-dire mes oncles, tantes, cousins et cousine, que je n'ai pas souvent l'occasion de voir, mais qui m'ont poussé à donner mon maximum. J'ai également une pensée pour les Fourure de Saint-Etienne (Jean-Christophe, France, Quentin et les autres) dont la présence m'a permis de me sentir moins éloigné de la famille. Je n'oublie pas non plus mon grand-père et ma grand-mère, qui sont probablement les personnes les plus heureuses au monde que je fasse une thèse. J'ai finalement aussi une pensée pour mon autre grand-mère, décédée au cours de ma thèse.

Enfin, il y a la famille que l'on choisit, à savoir les amis, tous très présents au cours de ces trois dernières années et envers qui j'ai toujours pu me tourner pour trouver du réconfort, ou juste une oreille attentive à mes nombreuses plaintes. Un grand merci à la team Agathe, Alexandre, Anaïs, Guillaume, Jean, Radu et Veronica avec

qui les sorties, soirées et même voyages ont toujours été de grande libération. Je vous remercie également de m'avoir rendu addict à Overwatch. Je remercie aussi à Maxime (et Elisabeth) dont la patience et la détermination à me conseiller ne sont pas pour rien dans la finalisation de ce manuscrit. Big check aussi à Aline qui a beau avoir fui à Perpignan, mais qui reste présente (je l'entends rire d'ici). Merci aussi à tous les autres, Clémence, Rémi, Benoit, Basile, Eva, Maeva, Dadou (et tous ceux que j'oublie probablement) d'avoir crus en moi.

Durant ces années de thèse j'ai aussi rencontré beaucoup de personnes super, au début des collègues, mais qui sont très vite devenus des amis. Je remercie particulièrement Emilie et Cécile qui m'ont permis de me défouler en me "suggérant" de me mettre au Kung-fu, Valentina qui m'aura supporté jusqu'au bout (mais se sera quand même enfui lors de ma rédaction), Baptiste qui était toujours présent pour aller boire une bière, Michaël et Irina qui ont fini trop tôt et dont la présence manquait au labo et Rémi et Claire sur qui je pouvais compter pour passer des bons moments et avoir des discussions très intéressantes sur tout et n'importe quoi. Merci également à Kevin qui a kidnappé Civet et ne m'autorise pas de droit de visite, Jordan et ses bons conseils sur les entreprises et notre petit nouveau Léo à qui je n'ai qu'une chose à dire : non je ne suis pas malade. Il y a aussi des gens super que j'ai rencontré en conférence. Je pense particulièrement à Bérengère et Romain grâce à qui j'ai passé un moment exceptionnel à Clermont-Ferrand et qui m'ont fait rencontrer Marine, que je remercie également d'avoir été présente dans une période difficile de ma vie. J'ai aussi une pensée très chaleureuse pour Isabelle qui m'a tenue compagnie et soutenue lors de la rédaction, pendant que tout le monde était en vacances. Et finalement un grand merci à mon colocataire Tanguy et sa copine Sophie pour tous les bons moments passés en votre compagnie.

Finalement, merci à tous les autres qui de près ou de loin ont participé à cette grande aventure qu'auront été mes trois années de thèse.

Table des matières

1 Introduction

1.1 Aperçu

La vision par ordinateur peut être vue comme une branche de l'intelligence artificielle qui consiste à doter les machines d'un système de perception "visuelle". Si les caméras et appareils photographiques permettent de capturer des images, ces dernières ne sont que des tableaux de pixels (points de couleur) et ne possèdent aucune information de plus haut niveau. La perception "visuelle" peut être considérée comme étant le processus permettant d'extraire une connaissance à partir des informations portées par la lumière et donc, en vision par ordinateur, à partir des pixels d'une image.

Un problème central en vision par ordinateur est le problème de la reconnaissance d'objets, qui consiste à percevoir les différents objets d'une scène. Les humains possèdent une capacité remarquable pour traiter les informations reçues, puisque d'un simple coup d'oeil nous sommes capables de reconnaître les différents éléments d'une scène. Par exemple lorsque nous conduisons nous reconnaissons naturellement la route, les autres voitures, les piétons etc. alors que cette tâche est extrêmement compliquée pour un ordinateur.

L'humain possède aussi une capacité de généralisation hors du commun. À partir de nos connaissances nous sommes capables de généraliser des concepts de manière à reconnaître la fonction d'objets que l'on voit pour la première fois. Un exemple très souvent cité dans le domaine de la vision, permettant d'illustrer cette capacité de généralisation, est celui des chaises. Les chaises sont des objets possédant une très grande diversité de formes, il en existe de toutes les formes et malgré cela nous sommes capables de reconnaître une chaise au premier regard, même si cette dernière possède une forme originale. Nos connaissances, apprises à partir des chaises que l'on a déjà vues et utilisées au cours de notre vie, nous permettent d'extraire des concepts communs et de généraliser la notion de chaise. Cette généralisation est un processus naturel et inconscient chez l'être humain, mais constitue un challenge en informatique et est à la base des techniques d'apprentissage automatique.

L'apprentissage supervisé, qui est étudié dans cette thèse, consiste à fournir de nombreux exemples à un algorithme et d'entraîner ce dernier de manière à ce qu'il

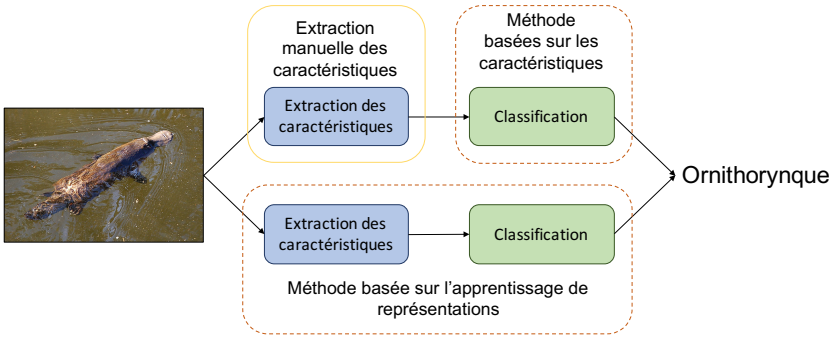


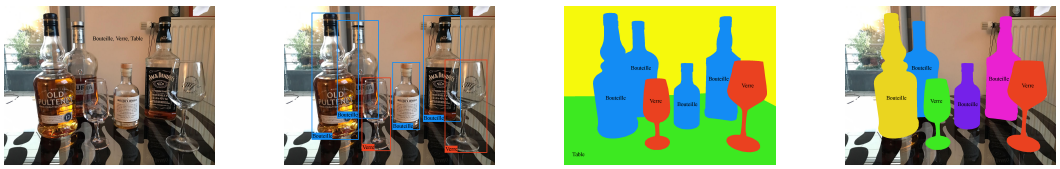
Fig. 1.1.: Illustration de la différence entre les méthodes basées sur les caractéristiques (chemin du haut) et les méthodes basées sur l'apprentissage de représentations (en bas). Les rectangles en pointillés montrent les parties entraînées par les algorithmes.

apprenne à reconnaître des caractéristiques communes permettant de généraliser les concepts de base et de reconnaître les éléments d'une scène. Il existe deux types de méthodes : les méthodes basées sur les caractéristiques calculées manuellement et les méthodes basées sur l'apprentissage de représentations (voir figure 1.1). Les méthodes basées sur les caractéristiques nécessitent l'extraction préalable de caractéristiques. Un expert du domaine doit décider des caractéristiques importantes (par exemple, pour les images, des caractéristiques de couleur ou de texture). Il extrait ensuite ces caractéristiques des données brutes (l'image) et les donne à un algorithme d'apprentissage qui apprend à classer les différents éléments en fonction de ces caractéristiques. Cette méthode présente l'inconvénient de reposer sur une extraction manuelle de caractéristiques.

Le second type de méthodes, principalement guidé par l'apprentissage profond, fournit à l'algorithme les données brutes (sans transformation préalable) et permet d'extraire automatiquement les caractéristiques nécessaires et de classer les éléments en fonction de ces caractéristiques. Cette technique présente l'avantage d'apprendre conjointement l'extraction des caractéristiques et la classification de ces dernières, permettant, à un algorithme, une totale liberté de décision. Les réseaux de neurones, qui constituent la principale approche des méthodes basées sur l'apprentissage des représentations, sont composés de multiples niveaux de représentations, obtenus par composition d'opérateurs simples qui permettent de transformer les données d'un niveau à l'autre, allant d'une représentation simple à une représentation plus complexe.

La tâche de reconnaissance d'objets à partir d'images (ou de vidéos) peut être séparée en quatre domaines avec un niveau de complexité croissant :

— La *classification d'images* qui vise à détecter la présence ou non d'objets dans une scène, indépendamment de leur position.



(a) Classification d'images (b) Détection d'objets (c) Segmentation sémantique (d) Segmentation sémantique d'instance

Fig. 1.2.: Comparaison des différentes tâches de reconnaissance d'images. De la tâche la moins précise à la plus précise : Classification d'images, Détection d'objets, Segmentation sémantique, Segmentation sémantique d'instance.

— La *détection d'objets* le plus souvent à l'aide de boîtes englobantes, qui vise à donner la position des objets en prédisant leurs boîtes englobantes.

— La *segmentation sémantique*, dont l'objectif est d'estimer la classe sémantique des objets d'intérêt présents dans une image.

— La *segmentation d'instances*, qui cherche en plus à distinguer les différentes instances des objets présents dans l'image.

Malgré les succès des premiers réseaux dans les années 1990, l'apprentissage profond et en particulier les réseaux de neurones convolutifs n'étaient pas très appréciés de la communauté scientifique principalement à cause du manque de preuves théoriques. Néanmoins, ces dernières années, des machines plus puissantes ainsi qu'un nombre croissant de données d'entraînement disponibles ont permis de faire des avancées extraordinaires dans le développement de nouveaux algorithmes. En 2012, l'équipe d'Alex Krizhevksy a obtenu des résultats impressionnants sur la compétition ILSVRC [Rus+15], dite "ImageNet", qui consiste à reconnaître les objets présents sur une image et de les ranger dans une des 1000 catégories recensées. Ce succès des réseaux de neurones convolutifs, leur a permis d'acquérir une popularité qui n'a cessé d'augmenter, à la fois dans la communauté scientifique, mais aussi auprès du grand public et de l'industrie.

Aujourd'hui, l'apprentissage profond et particulièrement les réseaux de neurones sont un domaine de recherche extrêmement actif qui permet de faire des avancées extraordinaires dans le domaine de la vision par ordinateur. Les algorithmes d'aujourd'hui sont, entre autres, capables de reconnaître les différents objets présents sur une image, de jouer à des jeux vidéo stratégiques très complexes tels que le jeu appelé Starcraft, ou encore de générer des images de plus en plus réalistes.

Cette thèse s'inscrit dans le cadre du projet ANR SoLStiCe (Similarités entre données localement structurées pour la vision par ordinateur) [1] porté par le LaHC (Laboratoire Hubert-Curien) à Saint-Étienne et en collaboration avec le LIRIS (Laboratoire

1. ANR-13-BS02-0002

d'InfoRmatique en Image et Systèmes d'information) à Lyon. Le projet SoLSTiCe avait pour objectif de créer de nouveaux modèles et outils afin de représenter et manipuler les images et vidéos. L'objectif était de permettre la reconnaissance de similarités, d'objets ou d'activités, à partir d'images ou de vidéos. Le projet visait à explorer différents types de données localement structurées (LSD) qui puissent être combinées à des structures discrètes (comme les chaînes, les arbres, les cartes combinatoires ou, plus généralement, les graphes) afin de mieux modéliser, puis d'exploiter, les relations locales entres les attributs. Trois problèmes principaux ont été étudiés : l'extraction de caractéristiques, la mesure de similarité et la fouille de données. Les techniques d'apprentissage profond ont notamment été utilisées afin de répondre à deux tâches du projet :

— L'extraction de caractéristiques, tels que, par exemple, des caractéristiques couleurs permettant d'obtenir une invariance des descripteurs images aux changements de condition d'illumination.

— L'apprentissage de caractéristiques permettant de reconnaître et d'extraire des motifs pertinents.

La tâche de segmentation sémantique répond particulièrement aux problématiques soulevées par le projet SoLSTiCe et a donc été l'objet principal de nos recherches. Cette dernière ne peut être résolue par les approches classiques de classification d'image en raison au grand nombre d'éléments à annoter et de la difficulté de la tâche d'annotation quand les classes sont "mal définies". Par conséquent, il est nécessaire de développer de nouvelles architectures spécialisées afin de produire des annotations denses et robustes.

1.2 Objectifs

Dans cette thèse nous nous sommes intéressés aux réseaux de neurones convolutifs, qui ont démontré leur efficacité pour la tâche de reconnaissance d'objets et avons étudié en détail les différents composants des réseaux ainsi que les verrous scientifiques qui ont mené aux architectures d'aujourd'hui, dans le but d'améliorer leurs performances sur la tâche de segmentation sémantique.

Nous avons cherché à comprendre comment il est possible de modifier les opérateurs internes des réseaux afin de les spécialiser à une tâche spécifique. Nous avons commencé par nous intéresser à la tâche de constance chromatique, qui vise à estimer la couleur des illuminants d'une scène afin de supprimer leurs influences d'une image. Cette tâche permet généralement de prétraiter les données afin de les rendre invariantes aux illuminants, ce qui permet aux algorithmes utilisés par la suite, d'extraire des caractéristiques non liées à l'éclairage des scènes. La majorité des méthodes de l'état de l'art étant basées sur les propriétés statistiques des images,

nous avons voulu étudier une approche basée sur l'apprentissage et plus précisément, sur l'apprentissage profond. L'objectif était d'apporter une approche nouvelle pour la résolution de la tâche de la constance chromatique, tout en améliorant nos connaissances et compréhensions des réseaux de neurones. Nous nous sommes notamment intéressés à l'opérateur de sous-échantillonnage présent dans les réseaux de neurones et nous avons cherché à montrer que ce dernier peut être modifié afin, par exemple, de se spécialiser à la tâche de constance chromatique.

Nous avons ensuite étudié la tâche de la segmentation sémantique, qui constitue l'objectif principal de cette thèse, où nous avons fait face à un manque de données annotées. Nous nous sommes donc intéressés la fonction de perte des réseaux afin de montrer qu'il est possible de l'adapter de manière à utiliser plusieurs bases de données d'entraînement simultanément. Cette fonction est particulièrement adaptée pour la tâche de segmentation sémantique, mais nous allons également montrer qu'elle est aussi utilisable sur des tâches de régression, telle que l'estimation de la pose de la main.

Pour finir, nous avons cherché à développer une nouvelle architecture de réseaux, conçue spécialement pour la segmentation sémantique, qui généralise les structures actuelles et qui permet aux réseaux de fournir des prédictions en haute résolution tout en conservant une précision dans la classification.

1.3 Contributions de la thèse

Les différentes contributions de cette thèse peuvent donc être résumées de la manière suivante :

— **Réseaux de neurones avec sous-échantillonnage mixte pour la constance chromatique.** (chapitre 3).

La constance chromatique est la capacité du système visuel humain à percevoir des couleurs constantes, quels que soient les changements de couleur de l'illuminant (éclairage). En vision par ordinateur, l'approche principale consiste à estimer la couleur de l'illuminant puis à supprimer son impact sur la couleur des objets. De nombreux algorithmes de traitement d'images ont été proposés afin de résoudre ce problème de manière automatique. Cependant, la plupart de ces approches reposent sur des hypothèses empiriques fortes, comme par exemple, l'hypothèse que la réflectance moyenne d'une scène est grise. D'autres approches plus perfectionnées peuvent très bien fonctionner sur certaines bases de données, mais mal s'adapter sur d'autres. Pour notre première contribution, nous avons étudié comment les approches basées sur les réseaux de neurones peuvent être utilisées pour traiter le problème de la constance chromatique.

Nous avons proposé une nouvelle architecture de réseau utilisant un opérateur de sous-échantillonnage, appelé sous-échantillonnage mixte qui est basé sur les approches classiques. Nous proposons aussi des méthodes d'augmentation artificielle des données afin de résoudre les problèmes d'entraînement liés au faible nombre de données d'apprentissage. Finalement, nous montrons que notre méthode permet d'obtenir des résultats proches de l'état de l'art et cela sur plusieurs bases de données dédiées à la constance chromatique.

Ces travaux ont fait l'objet d'une publication à la conférence ICIP (International Conference on Image Processing) en 2016 [Fou+16a].

— **Apprentissage multi-tâches, multi-domaines : application à la segmentation sémantique et à l'estimation de la pose de la main.** (chapitre 5).

Nous présentons une approche qui permet d'utiliser plusieurs bases de données annotées, définies pour différentes tâches (par exemple, plusieurs tâches de classification utilisant différents ensembles d'étiquettes) afin d'améliorer la généralisation obtenue sur chacune des bases de données. Nous proposons une nouvelle fonction de perte sélective qui peut être intégrée dans des réseaux de neurones profonds afin d'exploiter les données d'entraînement provenant de plusieurs bases de données annotées avec des ensembles d'étiquettes liés mais éventuellement différents. Nous montrons que l'approche d'inversion du gradient développée pour l'adaptation de domaine peut être utilisée dans cette configuration afin de gérer les décalages de domaines entre les différentes bases. Nous proposons également une approche d'auto-contexte qui permet de modéliser les corrélations existant entre les tâches. Des expérimentations approfondies sur deux types d'applications (segmentation sémantique et estimation de la pose des mains) montrent la pertinence de notre approche dans des contextes différents.

Ces travaux ont été en partie publiés dans les actes de la conférence française RFIA (Conférence Nationale en Reconnaissance des Formes et Intelligence Artificielle) [Fou+16b] et publiés dans les actes de la conférence internationale S+SSPR (International Workshops on Structural and Syntactic Pattern Recognition and Statistical Techniques in Pattern Recognition) en 2016 [Fou+16c]. Une étude plus approfondie, ainsi que les résultats sur une nouvelle tâche (l'estimation de la pose de la main), ont par la suite été publiés dans le journal Neurocomputing [Fou+17a].

— **GridNet : Un réseau conv-déconv en grille résiduelle pour la segmentation sémantique** (chapitre 6).

Dans notre dernière contribution, nous présentons GridNet, une nouvelle architecture de réseaux de neurones convolutifs conçue pour la segmentation sémantique d'image. Les réseaux de neurones classiques sont implémentés comme un seul chemin de l'entrée à la sortie avec des opérateurs de sous-échantillonnage appliqués le long du chemin afin de réduire la taille des

cartes caractéristiques et d'augmenter le champ réceptif du réseau pour la prédiction finale. Cependant, pour la segmentation sémantique d'image, où la tâche consiste à fournir une classe sémantique à chaque pixel d'une image, la réduction des cartes de caractéristiques est néfaste car elle conduit à une perte de résolution dans la prédiction de sortie. Pour s'attaquer à ce problème, notre GridNet suit un modèle de grille permettant à plusieurs chemins interconnectés de fonctionner à différentes résolutions. Nous montrons que notre réseau généralise de nombreux réseaux bien connus tels que les réseaux conv-deconv, résiduels ou U-Net. GridNet est entraîné à partir de zéro et obtient des résultats compétitifs sur l'ensemble de données Cityscapes.

Ces travaux ont été publié dans la conférence internationale BMVC (British Machine Vision Conference) 2017 [Fou+17b].

1.4 Organisation de la thèse

Les chapitres de la thèse sont organisés de la manière suivante :

— **chapitre 2 : Les réseaux de neurones convolutifs pour la vision par ordinateur**. Dans ce chapitre nous présentons les réseaux de neurones. Nous détaillons les différents éléments de construction ainsi que les opérateurs utilisés, puis nous expliquons comment entraîner un réseau en utilisant un algorithme de descente de gradient. Finalement, nous détaillons les différentes architectures qui ont été utilisées au fil des années en expliquant les évolutions et améliorations qu'elles ont entraînées.

— **chapitre 3 : Sous-échantillonnage mixte appliqué à la constance chromatique**. Dans ce chapitre nous introduisons la tâche de constance chromatique et présentons notre approche utilisant des réseaux de neurones. Nous commençons par donner un état de l'art des méthodes antérieures à notre contribution et expliquons comment ces dernières peuvent être définies sous la forme de réseau de neurones. Puis nous introduisons un opérateur de sous-échantillonnage basé sur les méthodes classiques et montrons que l'utilisation de ce dernier dans des réseaux de neurones permet d'obtenir des résultats proches de l'état de l'art. Finalement nous faisons un état de l'art des méthodes postérieures à notre contribution, permettant de répondre au problème de constance chromatique à l'aide de réseaux de neurones convolutifs.

— **chapitre 4 : Segmentation sémantique d'images**. Dans ce chapitre nous commençons par présenter la tâche de segmentation sémantique. Nous présentons ses objectifs ainsi que les difficultés engendrées. Une définition des différentes métriques ainsi que les fonctions de coûts couramment utilisées

pour la classification sont également détaillées dans ce chapitre. Nous présentons ensuite un ensemble de base de données créées afin de résoudre le problème de la segmentation sémantique et permettant d'évaluer les différentes méthodes développées. Finalement nous donnons un état de l'art des différentes architectures conçues pour cette tâche en détaillant leurs spécificités et leurs défauts.

— **chapitre 5 : Une fonction de coût sélective**. Dans ce chapitre nous présentons notre contribution sur les fonctions de pertes des réseaux. Nous commençons par positionner le problème d'utilisation de plusieurs bases de données en montrant qu'il s'agit d'un problème multi-tâches et multi-domaines. Ensuite nous détaillons les fonctions de perte classiques ainsi que les contraintes posées lors de l'utilisation de plusieurs bases simultanément. Ces limitations nous conduisent à développer une fonction de perte sélective adaptée à l'utilisation de multiples bases de données. Nous améliorons ensuite cette fonction afin de prendre en compte les corrélations existantes entre les étiquettes des différents ensembles d'étiquettes. Finalement, après avoir montré l'efficacité de notre fonction sur la tâche de segmentation sémantique (particulièrement adaptée à ce problème), nous montrons qu'elle est aussi intéressante sur des tâches de régression, telle que la tâche d'estimation de la pose de la main, où l'entraînement d'un réseau sur deux bases permet d'améliorer les résultats individuels obtenus.

— **chapitre 6 : GridNet, une architecture spécialisée pour la segmentation sémantique**. Dans ce chapitre nous étudions une architecture de réseaux de neurones convolutifs conçue spécialement pour la segmentation sémantique. Nous commençons par un rappel des architectures qui ont inspiré nos travaux. Puis nous présentons une nouvelle architecture, appelée GridNet, organisée en forme de grille de calculs. Afin d'aider l'entraînement de la grille nous introduisons aussi un opérateur de dropout appelé dropout par bloc. Finalement, nous validons notre architecture en montrant les résultats obtenus sur la base de données Cityscapes.

— **chapitre 7 : Conclusion et Perspectives** Le dernier chapitre conclut les travaux développés dans cette thèse et propose des futures directions de recherche en apprentissage profond.

2

Les réseaux de neurones convolutifs pour la vision par ordinateur

Dans le domaine de l'apprentissage automatique, les différentes méthodes, autres que les réseaux de neurones, reposent sur des algorithmes de classification utilisant des modèles théoriques et permettant de calculer des optimums mathématiquement justes. Ces algorithmes prennent généralement en entrée des caractéristiques extraites par des experts, dont le rôle est d'estimer quelles sont les propriétés les plus intéressantes à donner à un algorithme (telles que les caractéristiques de couleurs ou de textures d'une image).

En comparaison des autres méthodes, les réseaux de neurones, qui constituent ce que l'on appelle l'apprentissage profond (*Deep Learning* en anglais), présentent plusieurs particularités. Premièrement, bien que les résultats empiriques aient prouvé leur efficacité, les réseaux de neurones ne possèdent mathématiquement aucune garantie de convergence et l'optimum calculé n'a aucune garantie d'être l'optimum global de la fonction objectif. Ensuite, à la différence d'autres méthodes, les réseaux de neurones utilisent les données brutes en entrée et sont entraînés à extraire eux-mêmes les caractéristiques nécessaires à la tâche étudiée. Cela leur donne un avantage considérable puisque classification et extraction de caractéristiques sont liées, mais implique un phénomène de boîte noire. En effet, bien que l'on comprenne de mieux en mieux les représentations internes apprises par les réseaux, il est difficile de savoir quelles sont les caractéristiques qu'ils utilisent pour prendre une décision. Cela est problématique pour les utilisateurs qui ont souvent besoin de savoir sur quels éléments repose une décision (lorsque d'un réseau fournit une prédiction, il est difficile de répondre à la question "pourquoi cette prédiction ?").

Toutes ces particularités réunies font des réseaux de neurones une branche à part dans le domaine de l'apprentissage automatique et présentent de nombreuses questions et de verrous à soulever, faisant de l'apprentissage profond un domaine très étudié aujourd'hui par la communauté scientifique.

Dans ce chapitre nous verrons comment construire un réseau de neurones. Nous donnerons la composition et le fonctionnement des différents opérateurs utilisés

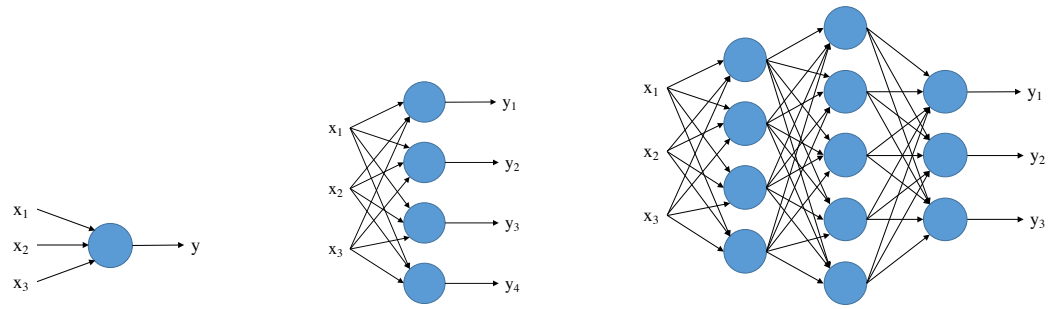
dans les structures des réseaux, tout en détaillant leur(s) fonctionnement(s) et leur utilité. Nous verrons ensuite le fonctionnement des algorithmes permettant l'entraînement des réseaux. Finalement, nous ferons un historique des différentes architectures développées au fil du temps en expliquant les verrous et les solutions apportées.

2.1 Présentation des réseaux de neurones

En apprentissage automatique, les réseaux de neurones correspondent à une classe de modèles composés de plusieurs couches de calculs formant un réseau profond. Ces couches sont composées de plusieurs opérateurs de base, appelés perceptron, inspirés des processus biologiques des connexions entre les neurones du cerveau, assemblés entre eux. Dans cette section, nous verrons ce qu'est un perceptron et comment les assembler pour créer des réseaux profonds à plusieurs couches. Nous ajouterons d'autres opérateurs en expliquant leur principe et leur utilité.

Dans ce chapitre et dans la suite du manuscrit, les nombres sont indiqués par des lettres en minuscules (par exemple x), les vecteurs par des lettres minuscules en gras ($\mathbf{x}$) et les matrices par des majuscules en gras ($\mathbf{X}$).

2.1.1 Perceptron multi-couches



(a) Perceptron

(b) Une couche entièrement connectée

(c) Réseaux de perceptrons multi-couches

Fig. 2.1.: Différentes étapes de construction d'un réseau de perceptrons multi-couches. À gauche, l'unité de base des réseaux de neurones : un perceptron. Au milieu, plusieurs perceptrons sont utilisés en parallèle pour former une couche entièrement connectée. À droite, plusieurs couches sont utilisées les unes à la suite des autres pour former un réseau de perceptrons multi-couches

Le perceptron est l'unité de base des réseaux de neurones. Il s'agit d'une fonction, prenant un vecteur d'entrée $\mathbf{x}$ et qui produit une sortie y en fonction d'un vecteur de

paramètres (souvent appelé poids) $\mathbf{w}$. La sortie y correspond à une somme pondérée des entrées $\mathbf{x}$ et des poids $\mathbf{w}$ à laquelle est ajouté un biais b (voir figure 2.1a).

Def. 2.1
Perceptron

$$\Theta(\mathbf{x}, \mathbf{w}) = h(\sum_i w_i \times x_i + b) = y \tag{2.1}$$

Pour simplifier les notations mathématiques, le biais b est généralement inclus dans les paramètres $\mathbf{w}$ en considérant $w_0 = b$ et $x_0 = 1$. L'équation 2.1 fait aussi apparaitre une fonction notée $h()$. Cette dernière, appelée fonction d'activation est une fonction non linéaire. Nous détaillerons son utilité ainsi que les différentes fonctions usuellement utilisées dans la section suivante (partie 2.1.2). La fonction modélisée par un perceptron est une fonction linéaire simple. Elle permet de modéliser des fonctions de base tel que, par exemple, des "**ou**" ou des "**et**" logique.

Un perceptron ne donne qu'une seule valeur sortie y, or souvent on peut vouloir des prédictions plus complexes composées de plusieurs valeurs de sortie, regroupées sous la forme d'un vecteur $\mathbf{y}$. Par exemple, une tâche pourrait consister à prédire des coordonnées x, y dans une image d'entrée. On peut aussi faire de la classification multi-classes d'objets en prédisant un vecteur de probabilités $\mathbf{y} = [y_1, y_2, ..., y_n]$, où chaque valeur de sortie y_i correspond à la probabilité d'appartenance à une classe d'indice i. Afin d'obtenir plusieurs sorties, il est nécessaire d'utiliser plusieurs perceptrons de manière parallèle, regroupés pour former ce que l'on appelle une couche de perceptrons (voir figure 2.1b). Une couche est dite entièrement connectée quand tous les perceptrons sont connectés à la totalité des entrées. Dans ce cas, les poids des perceptrons sont regroupés sous forme de matrice $\mathbf{W}$ et l'évaluation d'une couche de perceptrons revient à calculer le produit matriciel entre les entrées $\mathbf{x}$ et les poids $\mathbf{W}$.

Def. 2.2
Couche de perceptrons

$$\Theta(\mathbf{x}, \mathbf{W}) = h(\mathbf{W}\mathbf{x}) = \mathbf{y} \tag{2.2}$$

Bien qu'aujourd'hui la majorité des réseaux utilisent des couches entièrement connectées, il est possible, afin de réduire le nombre de paramètres, de ne pas connecter les perceptrons à toutes les entrées. Cela revient à fixer certaines valeurs de la matrice $\mathbf{W}$ à 0.

Finalement, lorsque l'on sait construire des couches de perceptrons, il devient naturel de les enchainer les unes après les autres afin de former un réseau plus complexe appelé réseau de perceptrons multi-couches (voir figure 2.1c). Mathématiquement, cela revient à combiner les fonctions formées par les différentes couches, la sortie d'une couche étant utilisée en entrée de la suivante.

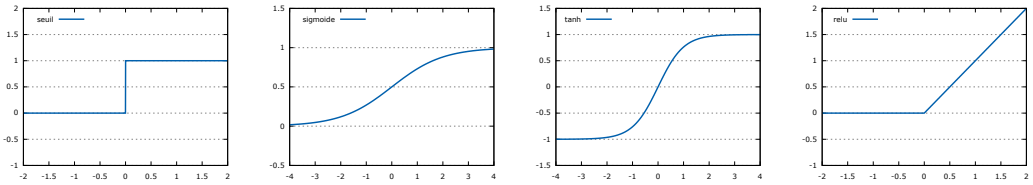
Def. 2.3
Perceptron multi-couches

$$(\Theta_1 \circ \Theta_2)(\mathbf{x}) = \Theta_1(\Theta_2(\mathbf{x})) = h_1(\mathbf{W}_1 h_2(\mathbf{W}_2 \mathbf{x})) \tag{2.3}$$

La construction d'un réseau de perceptrons multi-couches (aussi appelé réseau de neurones) correspond donc à la combinaison de plusieurs fonctions simples (les perceptrons) afin de former une fonction plus complexe. Intuitivement, la complexité de la fonction modélisée par un réseau de neurones dépend du nombre de couches de perceptron ainsi que du nombre de perceptrons par couche. Il a été montré qu'un réseau de neurones peut théoriquement approximer n'importe quelle fonction si ce dernier possède un nombre de couches et de neurones suffisamment élevé.

Le nombre d'entrées d'un perceptron multi-couches est déterminé par le type des entrées et le nombre de sorties par la tâche étudiée. Les couches au centre du réseau sont appelées couches cachées, car elles permettent d'apprendre des représentations intermédiaires propre au réseau dont il est difficile d'avoir connaissance.

2.1.2 Fonctions d'activations



(a) Fonction de seuil **(b)** Sigmoïde **(c)** Tanh **(d)** ReLU

Fig. 2.2.: Les fonctions d'activations les plus connues. De gauche à droite : la fonction de seuil (historique mais inutilisable dans les réseaux d'aujourd'hui), Sigmoïde, Tanh et ReLU (Rectified Linear Unit).

Les perceptrons, vus dans la partie précédente, modélisent des fonctions linéaires. Or, mathématiquement, la combinaison de plusieurs fonctions linéaires reste une fonction linéaire. Par conséquent, quel que soit le nombre de perceptrons et de couches de perceptrons utilisés, si l'on n'introduit pas de la non-linéarité dans les calculs, la fonction objectif calculée par le réseau de neurones restera une fonction linéaire. C'est pourquoi l'on utilise une fonction (notée h dans les formules précédentes) appelée *fonction d'activation* ou *fonction non linéaire*, dont le rôle est d'introduire une non linéarité dans les calculs.

Historiquement, les premiers perceptrons utilisaient la fonction de seuil :

Def. 2.4
Seuil

$$h(x) = \begin{cases} 1, & si\ x \geq 0 \\ 0, & sinon \end{cases} \tag{2.4}$$

Cette dernière permettait d'obtenir une prédiction binaire de type "oui" ou "non". Néanmoins, de par son gradient nul en tous points, elle ne permet pas d'entraîner le réseau par l'algorithme de descente de gradient que nous verrons dans la partie 2.2.1. Il a donc fallu la remplacer par des fonctions dont les gradients sont non nuls.

Les fonctions Sigmoïde et Tanh ont un comportement semblable à la fonction de seuil, mais avec un gradient non nul et une transition plus lisse (voir figure 2.2).

Def. 2.5
Sigmoïde

$$h(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

Def. 2.6
Tanh

$$h(x) = \frac{1 - e^{-2z}}{1 + e^{-2z}} \tag{2.6}$$

La principale différence entre ces deux fonctions étant que la fonction Sigmoïde ramène les valeurs d'entrées entre 0 et 1, alors que la fonction Tanh les ramène entre -1 et 1.

Bien que ces dernières aient été très utilisées, en 2011 l'introduction d'une nouvelle fonction nommée ReLU (Rectified Linear Unit) [Glo+11] les a quasiment remplacées car, elle permet un entraînement plus rapide des réseaux de neurones.

Def. 2.7
ReLU

$$h(x) = \max(0, x) \tag{2.7}$$

Il est intéressant de noter que le gradient de cette fonction est nul sur la moitié de son domaine (pour $x \in [-\infty, 0]$) ce qui pourrait poser un problème pour l'algorithme de descente de gradient (voir partie 2.2.1), or ce n'est pas le cas. En effet, le gradient n'étant pas nul sur l'autre moitié du domaine ($x \in [0, \infty]$) l'algorithme de descente de gradient est efficace quand les valeurs d'entrée sont > 0. De plus, de par sa grande plage de valeurs à 0 cette fonction introduit de la parcimonie dans les activations (de nombreuses valeurs sont à 0) permettant à des neurones de se spécialiser sur certaines tâches. Par exemple, si l'on cherche à reconnaitre des voitures, des vélos et des bateaux, on peut imaginer qu'une partie des neurones d'un réseau se spécialiseront dans la détection de roues (permettant de reconnaître les voitures et les vélos). Il parait donc normal que ces neurones ne soient pas activés lorsqu'une image de bateau (ne contenant donc pas de roue) est donnée en entrée du réseau. Enfin, le gradient nul de la fonction peut être vu comme une prise de

position dans la décision finale et donc dans l'erreur obtenue. En effet, dans notre exemple, si nos neurones responsables de détecter la présence de roues ne sont pas activés à la vue d'un bateau, il parait logique qu'ils ne soient pas pénalisés pour l'erreur faite en sortie par le réseau sur cette image.

Néanmoins, cela veut dire que les paramètres des neurones ne sont mis à jour uniquement lorsque ces derniers sont activés, ce qui peut ralentir l'entrainement des réseaux. C'est pourquoi des variantes de la fonction ReLU ont été créées, introduisant un gradient non nul pour des entrées $x \in [-\infty, 0]$. C'est le cas par exemple de la fonction Leakly ReLU de Maas *et al.* [Maa+13] qui introduit un coefficient a très faible pondérant l'entrée lorsque $x < 0$: $h(x) = max(0, x) + a \times min(0, x)$ ou encore PReLU de He *et al.* [He+15] (pour Parametric Rectified Linear Unit) qui généralise la fonction Leakly ReLU en faisant du paramètre a un paramètre entraînable du réseau.

2.1.3 Convolutions

Si l'on veut utiliser des images en entrée d'un réseau de perceptrons multi-couches, il est nécessaire de créer une valeur d'entrée par pixel de l'image. Or le nombre de pixels, dans une image, peut être très grand (allant de 784 pour des images de taille 28×28 à 160000 pour des images de taille 400×400). Cela implique que la première couche du réseau utilise de nombreux perceptrons, ce qui entraîne une augmentation importante du nombre de paramètres du réseau. De plus, en faisant cela nous n'exploitons pas les informations spatiales des pixels (quel pixel se situe où dans l'image) et surtout le réseau ne possède pas les informations de localisation (il ne sait pas quels sont les voisins d'un pixel).

La prise en compte du voisinage d'un pixel est une notion connue de la communauté de vision par ordinateur et est depuis longtemps exploitée notamment par les algorithmes reposant sur l'extraction manuelle de caractéristiques. Par exemple, les descripteurs les plus utilisés aujourd'hui sont les descripteurs SIFT (*Scale-Invariant Feature Transform*) [Low99] et HOG (*Histogram of Oriented Gradients*) [DT05] qui permettent des analyses locales des caractéristiques d'une image. Ces opérateurs, qui introduisent une invariance à la translation, peuvent être généralisés par un opérateur mathématique, appelé opérateur de convolution. Il est donc naturel d'exploiter ces opérateurs de convolutions dans les réseaux de neurones.

L'opérateur de convolution (ou produit de convolution) est une fonction mathématique prenant en paramètre deux fonctions f et g et dont le résultat, noté $f * g$, est égal en tous points x à l'intégrale de la fonction f centrée en x multipliée par l'intégrale de la fonction g centrée en 0. Dans le cas général, où les fonctions f et

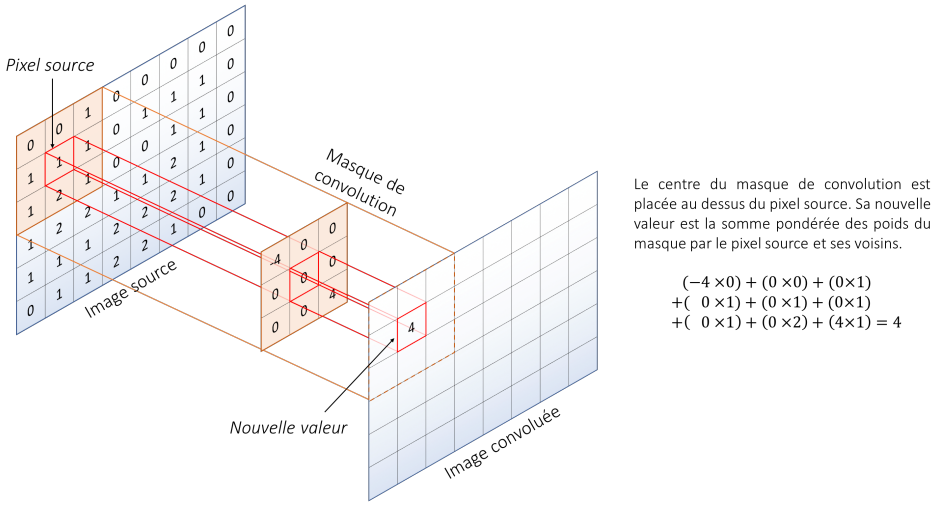


Fig. 2.3.: Illustration d'un opérateur de convolution. L'image source est convoluée avec un masque de convolution (ici de taille 3×3). Il est important de noter que, pour l'illustration, le masque de convolution a subi une rotation verticale et horizontale.

g sont des fonctions continues et définies sur un domaine infini (de $-\infty$ à ∞), le produit de convolution nécessite l'utilisation de l'opérateur d'intégrale. Néanmoins, dans le cas de traitement d'image, la fonction f correspond à une image et est donc discrète et définie sur un domaine fini. La fonction g, appelée masque de convolution est, elle aussi, discrète et est définie sur un domaine de taille S.

Def. 2.8 *Produit de convolution*

Le produit de convolution de la fonction discrète $f \in \mathbb{N}$ par le masque de convolution $g \in \mathbb{N}$ de taille S (et de demi taille $K = \frac{S-1}{2}$) est donné par :

$$(f * g)(n) = \sum_{n'=-K}^{K} f(n - n') \times g(n') \tag{2.8}$$

L'équation 2.8 donne la formule du produit de convolution de la fonction f par la fonction g toutes deux définies sur un espace à une dimension. Or, dans le cas des images, les fonctions sont généralement définies sur un espace à deux dimensions. En effet, l'image d'entrée est une fonction $f \in \mathbb{N}^2$, dont les deux dimensions H, W correspondent à la hauteur et largeur de l'image et qui, à toute paire d'entier $x, y \in \mathbb{N}$, fait correspondre la valeur du pixel à la position (x, y). Le masque de convolution g est donc une fonction elle aussi définie sur un espace à deux dimensions.

Def. 2.9 *Convolution 2D*

Le produit de convolution de l'image I par le masque de convolution g de taille $S \times S$ (et de demi taille $K \times K$ avec $K = \frac{S-1}{2}$) est donnée par :

$$(f * g)(n, m) = \sum_{n'=-K}^{K} \sum_{m'=-K}^{K} f(n - n', m - m') \times g(n', m') \tag{2.9}$$

La figure 2.3 illustre le fonctionnement de l'opérateur de convolution.

Dans les réseaux de neurones convolutifs, les opérateurs de convolutions prennent en entrée les différents canaux des images quand ils sont sur la première couche du réseau ou les différentes cartes de caractéristiques quand ils sont au milieu du réseau. Par conséquent, les résultats d'un opérateur de convolution correspondent à une carte de caractéristique qui combine les cartes de caractéristiques (ou canaux de l'images) reçu en entrée.

Def. 2.10 *Couche de convolution*

Dans un réseau de neurones convolutif, la $j^{ème}$ carte de caractéristiques d'une couche de convolution est obtenue en combinant les N cartes de caractéristiques d'entrées par N masques de convolution. Si l'on note f_i et g_i les $i^{ème}$ cartes de caractéristiques d'entrée et masque de convolution de la couche de convolution et b_j le biais associé à la carte de caractéristiques de sortie j alors, la carte de caractéristiques j est obtenue de la manière suivante :

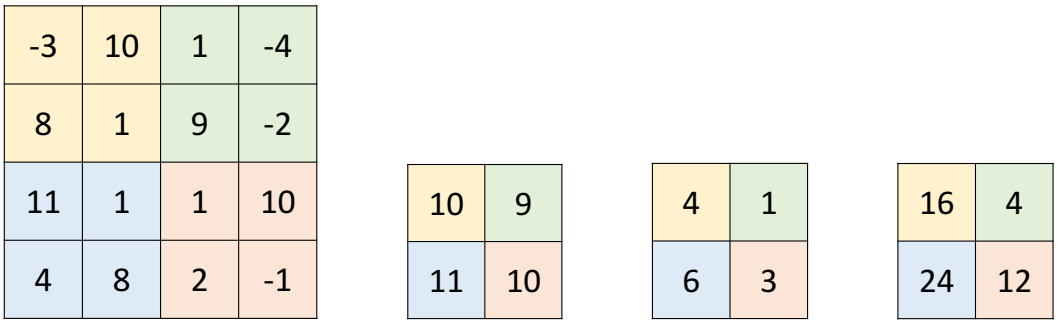
$$(f * g)_j(n, m) = b_j + \sum_{i=1}^{N} \sum_{n'=-K}^{K} \sum_{m'=-K}^{K} f_i(n - n', m - m') \times g_{ij}(n', m') \qquad (2.10)$$

Ainsi, une couche de convolution correspond à une combinaison de plusieurs convolutions appliquées sur les cartes de caractéristiques d'entrées. Cela permet aux couches successives d'extraire des caractéristiques de plus en plus complexes. Par exemple, les premières couches de convolution peuvent extraire des traits ou des arrondis avec différentes orientations dans l'image de base. Ensuite, les couches suivantes peuvent être entraînées à extraire des combinaisons de ces caractéristiques afin, par exemple, de détecter les yeux, le nez et la bouche. Plus profondément, les couches suivantes peuvent enfin être utilisées pour reconnaître la présence simultanée d'un nez, d'une bouche et des yeux afin de reconnaitre un visage. Ce qui permet d'analyser à plus haut niveau comment est structurée d'un point de vue topologique une image. La partie convolutive d'un réseau de neurones convolutif est donc considérée comme un extracteur de caractéristiques hiérarchiques, avec des caractéristiques dont la complexité augmente au fil des couches.

2.1.4 Opérateurs de sous-échantillonnage

Dans un réseau de neurones convolutif, les couches de convolutions sont alternées avec des couches de sous-échantillonnage (appelé *pooling* en anglais). Ces dernières réduisent la taille des cartes de caractéristiques, permettant ainsi de réduire l'espace mémoire ainsi que le nombre de neurones à l'entrée des parties entièrement connectées tout en limitant le nombre de paramètres des réseaux. Nous verrons aussi plus

tard (dans le chapitre 4.2.1) qu'elles ont aussi un impact important sur ce qu'on appelle le champ réceptif d'un réseau de neurones.



(a) Entrées **(b)** Maximum pooling **(c)** Average pooling **(d)** Sum pooling

Fig. 2.4.: Exemples de sous-échantillonnages. De gauche à droite : l'entrée, les résultats avec un opérateur de sous-échantillonnage par valeur maximale (maximum pooling), de sous-échantillonnage par valeur moyenne (average pooling) et de sous-échantillonnage par somme (sum pooling). Dans cet exemple la taille S des fenêtres ainsi que le pas P utilisés sont tous deux de 2.

Les opérateurs de sous-échantillonnages fonctionnent à la manière des opérateurs de convolution. Ce sont des fenêtres glissantes d'une taille donnée $S \times S$ agrégeant les valeurs d'entrées. La réduction des tailles des cartes de caractéristiques est obtenue en modifiant le pas $P \times P$ avec lequel la fenêtre "glisse" sur les cartes de caractéristiques d'entrées.

Bien que les différents paramètres soient personnalisables, dans une très grande majorité des cas, la taille de la fenêtre de sous-échantillonnage ainsi que les pas sont de 2×2, ce qui permet de réduire les cartes de caractéristiques par 4 (réduction de moitié de la hauteur et de la largueur).

Plusieurs stratégies d'agrégation des valeurs sont possibles. Dans la figure 2.4 nous donnons trois exemples d'agrégation des valeurs d'entrées :

— Un sous-échantillonnage par valeur maximale (figure 2.4b), appelé *maximum pooling*, qui consiste à récupérer la valeur maximale dans la fenêtre d'observation.

— Un sous-échantillonnage par valeur moyenne (figure 2.4c), appelé *average pooling*, qui consiste à calculer la moyenne des valeurs dans la fenêtre d'observation.

— Un sous-échantillonnage par somme des valeurs (figure 2.4d), appelé *sum pooling*, qui consiste à calculer la somme des valeurs dans la fenêtre d'observation.

De nombreuses autres stratégies sont possibles, tel que, par exemple, le sous-échantillonnage stochastique (*stochastic pooling* en anglais) proposé par Zeiler et Fergus [ZF13] et qui introduit du non déterminisme dans l'opérateur d'agrégation.

Néanmoins, le sous-échantillonnage très majoritairement utilisé aujourd'hui est celui par valeur maximale avec une fenêtre de taille 2×2 et un pas de 2×2. Une autre stratégie de plus en plus populaire consiste à ne pas utiliser d'opérateurs de sous-échantillonnage, mais à modifier directement le pas des opérateurs de convolution. Intuitivement cela revient à ne produire qu'une valeur sur deux lors de l'application des convolutions, réduisant ainsi la taille des cartes de caractéristiques tout en limitant les calculs et la mémoire utilisée par les opérateurs de sous-échantillonnage.

2.2 Entraînement des réseaux

On a vu dans les sections précédentes comment construire un réseau de neurones convolutif. On s'intéresse maintenant à l'entraînement des réseaux, c'est-à-dire à la manière de modifier les poids des réseaux de manière à ce que la fonction qu'ils modélisent permette de répondre à une tâche désirée.

Notons la fonction modélisée par un réseau de neurones (ou autre algorithme) $\Theta(x, \theta) = \hat{y}$. Cette dernière prend une entrée x (par exemple une image) et des paramètres θ (noté aussi w dans le cas des réseaux de neurones), correspondant aux poids des différentes couches du réseau, et produit une sortie $\hat{y}$ modélisant la prédiction du réseau (par exemple un vecteur de probabilités). En apprentissage automatique supervisé, l'entraînement d'un réseau de neurones se fait en utilisant des bases d'entraînements $\mathcal{B}$. Ces dernières contiennent des données d'entraînement x associées à des annotations (appelée vérités terrain) y correspondant à la sortie désirée pour l'entrée x. Une fonction, appelée fonction d'erreur (ou fonction de perte) $J(x_i, y_i, \theta) = \epsilon_i$ mesure la différence entre les estimations $\hat{y}_i = \Theta(x_i, \theta)$ et les vérités terrain y_i. L'objectif en apprentissage automatique, est de trouver les paramètres θ minimisant la somme des erreurs empiriques observées sur les exemples d'entraînement de la base de données $\mathcal{B}$. C'est-à-dire, de trouver les paramètres θ qui minimisent un risque empirique R tel que $R[\theta, \mathcal{B}] = \frac{1}{N} \sum_i^N J(x_i, y_i, \theta)$. Pour des soucis de notation nous n'inclurons pas les entrées et vérités terrains dans les formules $J(x_i, y_i, \theta)$ utilisées par la suite (sauf si cela est nécessaire).

La fonction d'erreur $J(x_i, y_i, \theta)$ mesurant l'erreur entre une prédiction $\hat{y}$ et une vérité terrain y dépend de la tâche étudiée. Dans ce manuscrit, une fonction de coût pour la tâche de constance chromatique est détaillée dans la partie 3.1.4, la fonction de coût la plus utilisée pour la tâche de classification d'image et de segmentation sémantique est détaillée dans la partie 4.1.2 et finalement une fonction de coût sélective, que nous avons développée pour résoudre des problèmes multi-tâches est détaillée dans le chapitre 5.

Dans cette section, nous nous intéressons à l'entraînement d'un réseau de neurones, c'est-à-dire aux techniques utilisées pour modifier les paramètres d'un réseau de manière à obtenir les poids optimaux qui minimisent le risque empirique (la somme des erreurs) obtenu sur les données d'entraînement.

2.2.1 Descente de gradient

La descente de gradient est un algorithme itératif connu et simple d'utilisation qui permet de minimiser la fonction $J(\theta)$, paramétrée par des poids θ d'un réseau de neurones. Minimiser la fonction (c'est-à-dire trouver les paramètres θ qui minimisent l'erreur empirique) se fait en calculant le gradient $\nabla_\theta J(\theta)$ et en mettant à jour les paramètres θ dans la direction opposée de ce dernier. Intuitivement, le gradient $\nabla_\theta J(\theta)$ donne la direction de la pente de la fonction J au point θ, par conséquent, se diriger dans la direction opposée du gradient revient à descendre cette pente. Un hyper-paramètre η, appelé pas d'apprentissage, détermine la longueur du pas à faire dans la direction du gradient. Une fois les paramètres mis à jour, on calcule le nouveau gradient et on se dirige à nouveau dans la direction opposée. On minimise ainsi la fonction en avançant pas par pas dans la direction de la pente.

Il existe trois variantes connues qui diffèrent en fonction du nombre de données utilisées pour calculer l'erreur empirique de la fonction objectif $J(\theta)$ et donc de son gradient. La première, appelée descente de gradient par batch utilise toutes les données de la base d'entraînement $\mathcal{B}$ pour calculer l'erreur globale. Ainsi, le gradient $\nabla_\theta J(\theta)$ donne la direction exacte de la pente permettant de minimiser l'erreur globale.

Def. 2.11
Descente de gradient par batch

$$\theta = \theta - \eta \times \nabla_\theta J(\theta, \mathcal{B}) \tag{2.11}$$

Si cette dernière présente l'avantage de calculer le gradient exacte (et donc la pente exacte) pour minimiser l'erreur sur l'ensemble des données d'entraînement, elle est très lourde à mettre en place. En effet, cela implique que, pour chaque pas η il est nécessaire d'évaluer l'ensemble des données d'entraînement afin d'obtenir l'erreur de chacun des exemples. Or si la base de données $\mathcal{B}$ contient énormément de données cela peut être très long.

Une deuxième variante consiste donc à ne calculer l'erreur et donc le gradient que d'une donnée $x_i \in \mathcal{B}$ à la fois.

Def. 2.12
Descente de gradient stochastique

$$\theta = \theta - \eta \times \nabla_\theta J(\theta, x_i, y_i) \tag{2.12}$$

Cette variante, appelée descente de gradient stochastique, car les exemples d'entrainement sont choisis de manière aléatoire, permet de mettre à jour les poids après chaque évaluation et est donc beaucoup plus rapide pour faire évoluer les paramètres du réseau, mais avec le défaut que le gradient calculé à chaque étape ne correspond pas au véritable gradient du risque empirique. Néanmoins, minimiser les erreurs des exemples un-à-un permet une convergence moyenne vers un minimum local. Intuitivement, il est possible de faire un pas dans la mauvaise direction, mais la moyenne des pas effectués ira dans la bonne direction. De plus, ce calcul stochastique du gradient possède des propriétés de régularisation intéressantes. En effet, un exemple d'entraînement compliqué (ou faux) dans la base de données n'aura une influence sur le gradient que lorsque ce dernier sera considéré, ce qui entraînera un pas dans une mauvaise direction, mais qui sera corrigé par les exemples suivants. Cela a pour effet de lisser la fonction objectif et donc permet une meilleure généralisation.

Néanmoins, même si la direction globale des gradients permet de se diriger vers un optimum, les gradients individuels de chacun des exemples présentent de grosses variations, entraînant souvent l'algorithme à changer de directions. Par conséquent, une troisième variante de l'algorithme de descente de gradient présente un juste milieu entre la descente de gradient par batch et la descente de gradient stochastique. L'idée est de calculer le gradient pour un sous-ensemble de n données, tirées aléatoirement dans la base de données $\mathcal{B}$. Ce sous-ensemble est appelé *mini-batch* en conséquence l'algorithme correspondant est appelé *descente de gradient par mini-batch*.

Def. 2.13
Descente de gradient par mini-batch

$$\theta = \theta - \eta \times \nabla_\theta J(\theta, x_{(i;i+n)}, y_{(i;i+n)}) \tag{2.13}$$

L'utilisation d'un mini-batch pour calculer l'erreur permet d'obtenir un gradient plus précis (plus proche du véritable gradient de la fonction objectif) tout en gardant l'avantage de ne pas évaluer l'ensemble des données pour chaque pas de l'algorithme.

C'est pourquoi la descente de gradient par mini-batch est aujourd'hui devenue la norme pour l'entraînement des réseaux de neurones. Néanmoins, elle garde le défaut (moins prononcé) de faire des pas dans la mauvaise direction, ralentissant ainsi l'entraînement des réseaux. Pour pallier ce problème, plusieurs améliorations ont été développées, permettant à l'algorithme de converger (en théorie) plus vite vers un minimum local de la fonction $J(\theta)$. Ces algorithmes, appelés optimiseurs, consistent à estimer la direction optimale du gradient basée sur le gradient obtenu à chaque étape avec l'erreur faite sur le mini-batch.

Une première amélioration possible qui permet d'estimer une direction globale est d'ajouter une inertie dans le gradient (*momentum* en anglais). Cette technique,

détaillée par Qian [Qia99], consiste à combiner le gradient obtenu à l'étape t avec l'estimation obtenue à l'étape $t-1$. Pour cela, un hyper-paramètre γ est utilisé. La direction ν_t à suivre à l'étape t est calculée en combinant la direction obtenue à l'étape précédente v_{t-1} avec le gradient $\nabla_\theta J(\theta)$ de la manière suivante : $\nu_t = \gamma \times \nu_{t-1} + \eta \times \nabla_\theta J(\theta)$. Les paramètres θ sont ensuite mis à jour de manière classique : $\theta = \theta - \nu_t$. L'inertie ajoutée au calcul du gradient permet d'éviter les changements de directions trop importants et donc d'éviter aux paramètres de trop zigzaguer.

Basés sur cette idée d'inertie de nombreux autres optimiseurs présentent des différences plus ou moins importantes. On peut par exemple présenter l'optimiseur NAG (*Nesterov accelerated gradient*) [Nes83] qui calcule le gradient après avoir avancé d'un pas dans la direction précédente, Adagrad [Duc+11], qui adapte le pas d'apprentissage η pour chacun des paramètres indépendamment, Adadelta [Zei12] qui est une amélioration d'Adagrad et qui permet d'éviter des changements trop brutaux dans les pas d'apprentissage des paramètres ou RMSprop [Dau+15] développé en parallèle d'Adadelta et qui est un cas particulier d'Adadelta.

L'optimiseur Adam (*Adaptive Moment Estimation*) [KB14] va encore plus loin en calculant non seulement les moyennes m_t des paramètres, mais aussi leurs variances v_t. Formellement, si l'on considère $g_{t,i}$ comme étant le gradient de la fonction J au temps t par rapport au paramètre θ_i alors la première étape consiste à calculer la moyenne et la variance de ce gradient à travers le temps. Ces dernières sont ensuite normalisées afin d'éviter les valeurs trop proches de zéro, puis les paramètres sont mis à jour (voir algorithme 1).

Algorithm 1: Mise à jour des poids en utilisant l'algorithme Adam [KB14].

for t *allant de* 0 *à* ∞ **do**
  $m_t \longleftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$ ▷ Calculs de la moyenne (avec inertie) des gradients à travers le temps
  $v_t \longleftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$ ▷ Calculs de la variance (avec inertie) des gradients à travers le temps
  $\hat{m}_t \longleftarrow \frac{m_t}{1-\beta_1^t}$ ▷ Normalisation de la moyenne
  $\hat{v}_t \longleftarrow \frac{v_t}{1-\beta_2^t}$ ▷ Normalisation de la variance
  $\theta_{t+1} \longleftarrow \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$ ▷ Mise à jour des paramètres
  if *Convergé* **then**
    Fin
  end
end

L'optimiseur Adam présente le défaut d'ajouter trois hyper-paramètres β_1, β_2 et ϵ ce qui rend l'algorithme plus dur d'utilisation puisqu'il faut trouver les bonnes

valeurs. Les auteurs conseillent néanmoins d'utiliser les valeurs $\beta_1 = 0.9$, $\beta_2 = 0.999$ et $\epsilon = 10^{-8}$.

Il existe encore de nombreux optimiseurs parmi lesquels on peut mentionner AdaMax [KB14] ou Nadam [Doz16] qui essayent d'améliorer la stabilité d'Adam.

Le problème avec tous ces optimiseurs, est qu'il est souvent difficile de les comparer car ils ont des comportements différents en fonction des données utilisées. Par conséquent, il n'existe pas de règles précises sur l'utilisation d'un optimiseur plutôt d'un autre, néanmoins, il est possible d'en privilégier certains par rapport à l'objectif recherché. Par exemple, si l'objectif est de tester ou d'optimiser une nouvelle architecture, il peut être intéressant d'utiliser un optimiseur ayant peu d'hyper-paramètres. Ainsi une bonne optimisation de l'optimiseur sera plus facile et il sera plus aisé de vérifier le fonctionnement de l'architecture testée. Au contraire, si l'objectif est d'entraîner une architecture ayant déjà fait ses preuves, il peut être intéressant d'utiliser un optimiseur plus complexe permettant ainsi une finesse plus précise dans les hyper-paramètres et donc une optimisation plus précise.

2.2.2 Rétro-propagation

Nous avons vu, dans la section précédente, les différents algorithmes de descente de gradient ainsi que le fonctionnement des optimiseurs. Néanmoins, la descente de gradient repose, comme son nom l'indique, sur le gradient de la fonction objective (c'est-à-dire le gradient du réseau de neurones et de la fonction de coût associée). Dans cette partie nous expliquerons comment calculer le gradient des paramètres d'un réseau de neurones permettant ainsi d'être optimisé grâce à l'algorithme de descente de gradient.

Nous savons que les réseaux de neurones sont composés de couches d'opérateurs et sont donc des compositions de fonction. Afin d'obtenir l'erreur faite par le réseau sur un mini-batch (voir section précédente) nous présentons donc les entrées à la première couche du réseau qui calcule les activations (cartes de caractéristiques) et transmet ces données à la couche suivante qui recommence le processus jusqu'à la sortie du réseau. Cette étape, où les données traversent les différentes couches du réseau, de l'entrée à la sortie, est appelée propagation ("*forward*" en anglais). À partir des sorties du réseau, une fonction de perte permet de calculer l'erreur faite par rapport aux valeurs cibles.

Une fois que l'on a obtenu l'erreur, on cherche à calculer le gradient de cette dernière par rapport aux paramètres du réseau. Un réseau étant une composition de fonctions,

on utilise le théorème de dérivation des fonctions composées aussi appelé règle de dérivation en chaîne (ou *chain rule* en anglais).

Def. 2.14 *Chain rule* *Si g est une fonction dérivable au point x et f est une fonction dérivable au point $g(x)$ alors $(f \circ g)$ est dérivable au point x et :*

$$(f \circ g)'(x) = f'(g(x)) \times g'(x) \tag{2.14}$$

On peut réécrire cette formule en utilisant la notation des dérivées de Leibniz :

$$(f \circ g)'(x) = \frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g(x))}{\partial g(x)} \times \frac{\partial g(x)}{\partial x} \tag{2.15}$$

Ce théorème nous donne la possibilité d'exprimer le gradient des paramètres d'un réseau de neurones en fonction des gradients des différents opérateurs composant le réseau.

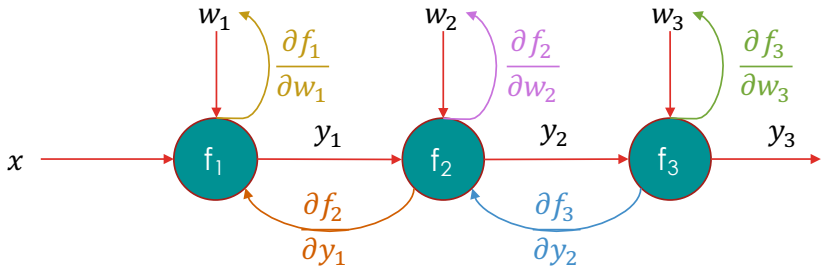


Fig. 2.5.: Exemple du calcul de gradient sur un réseau composé de trois opérateurs f_1, f_2, f_3 dépendant des paramètres w_1, w_2 et w_2.

Afin d'expliquer comment calculer le gradient des paramètres, nous utiliserons un exemple dont le réseau est schématisé dans la figure 2.5. Notre exemple est un réseau composé de trois couches de calculs f_1, f_2 et f_3 dépendant des paramètres respectifs w_1, w_2 et w_3. Le réseau prend une entrée x qui est donnée à l'opérateur f_1 pour produire une sortie y_1 donnée à l'opérateur f_2 pour obtenir une sortie y_2 qui est finalement donnée à l'opérateur f_3 pour obtenir la sortie finale y_3.

Pour rappel, l'objectif est de calculer le gradient des paramètres par rapport à y_3 et donc $\nabla_{w_1,w_2,w_3} y_3 = [\frac{\partial y_3}{\partial w_1}, \frac{\partial y_3}{\partial w_2}, \frac{\partial y_3}{\partial w_3}]$.

Le gradient de y_3 par rapport à w_3 est facile à calculer, car la fonction f_3 (qui produit la sortie y_3) dépend directement de w_3. On a donc :

$$\begin{aligned} \nabla_{w_3} y_3 &= \frac{\partial y_3}{\partial w_3} \\ &= \frac{\partial f_3(y_2, w_3)}{\partial w_3} \end{aligned} \tag{2.16}$$

Le gradient de y_3 par rapport à w_2 est moins évident à calculer puisque y_3 est obtenue par composition de la fonction f_2 et f_3 (on a $(f_2 \circ f_3)(y1) = y_3$). Il est donc nécessaire d'utiliser le théorème de dérivation des fonctions composées vu ci-dessus. On obtient alors :

$$\begin{aligned}
\nabla_{w_2} y_3 &= \frac{\partial y_3}{\partial w_2} \\
&= \frac{\partial f_3(y_2, w_3)}{\partial w_2} \\
&= \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial y_2}{w_2} \\
&= \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial f_2(y_1, w_2)}{\partial w_2}
\end{aligned} \tag{2.17}$$

Autrement dit, pour calculer le gradient de y_3 par rapport à w_2 il faut calculer le gradient de f_3 par rapport à son entrée y_2 puis le gradient de f_2 par rapport à ses paramètres w_2.

De la même manière on peut calculer le gradient de w_1 par rapport à y_3 en utilisant plusieurs fois le théorème de dérivation des fonctions :

$$\begin{aligned}
\nabla_{w_1} y_3 &= \frac{\partial y_3}{\partial w_1} \\
&= \frac{\partial f_3(y_2, w_3)}{\partial w_1} \\
&= \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial y_2}{\partial w_1} \\
&= \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial f_2(y_1, w_2)}{\partial w_1} \\
&= \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial f_2(y_1, w_2)}{\partial y_1} \times \frac{\partial y_1}{\partial w_1} \\
&= \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial f_2(y_1, w_2)}{\partial y_1} \times \frac{\partial f_1(x, w_1)}{\partial w_1}
\end{aligned} \tag{2.18}$$

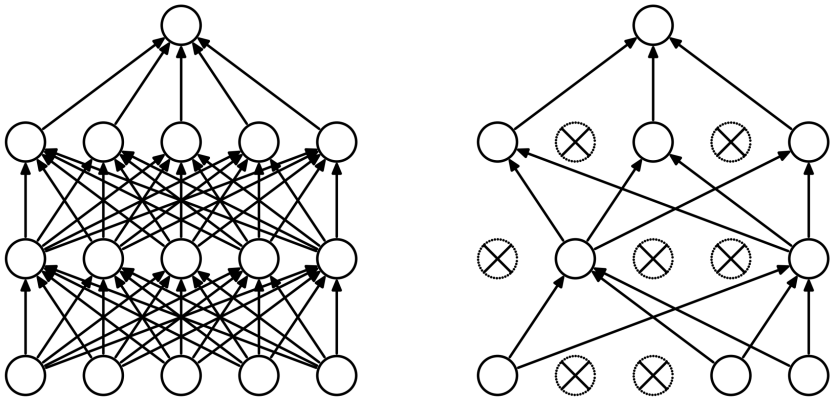
Ainsi, pour calculer le gradient de w_1 par rapport à y_3 on a besoin du gradient de f_3 par rapport à y_2, du gradient de f_2 par rapport à y_1 et du gradient de f_1 par rapport à x. Or, le gradient de f_3 par rapport à y_2 a déjà été calculé pour obtenir $\nabla_{w_2} y_3$ il n'est donc pas nécessaire de le recalculer.

De manière générale, pour chaque opérateur (ou couche) $f(x, w) = y$ présent dans un réseau de neurones il faut pouvoir calculer le gradient de la couche par rapport à ses paramètres $\frac{\partial f(x,w)}{w}$ ainsi que le gradient de la couche par rapport à ses entrées $\frac{\partial f(x,w)}{x}$ afin de le transmettre aux couches précédentes. Cette transmission du gradient (qui remonte le réseau au travers des différentes couches) est appelée rétro-propagation du gradient (ou *backpropagation* en anglais).

Le calcul du gradient des paramètres d'un réseau se fait donc en deux étapes, la propagation (*forward pass*) des données dans le réseau, de l'entrée jusqu'à la fonction de perte puis la rétro-propagation (*backward pass*) des gradients de la fonction de pertes jusqu'à l'entrée du réseau.

2.2.3 Dropout

Lorsque les réseaux de neurones deviennent trop grands (avec trop de neurones et donc trop de paramètres) il devient difficile de les entraîner sans subir un effet de sur-apprentissage. Lorsqu'un réseau sur-apprend (apprend à reconnaitre par cœur les données d'entrée) ses neurones se spécialisent pour reconnaitre les images de la base de données d'entraînement (et perdent donc en performance de généralisation). Intuitivement, on peut considérer que chaque neurone reconnait une image précise de l'ensemble d'entraînement et que ce dernier, quand il reconnait son image, est l'unique responsable de la prédiction finale. Afin de corriger ce problème, Srivastava *et al*. ont développé une technique, appelée *dropout* [Sri+14], empêchant les neurones d'être seuls responsables dans une prise de décision. Cette technique, a montré des résultats exceptionnels lors de l'entraînement des réseaux.



(a) Standard Neural Net (b) After applying dropout.

Fig. 2.6.: Illustration de l'opérateur de Dropout (tirée du papier [Sri+14]). À droite le réseau entièrement connecté. À gauche, des neurones ont été aléatoirement désactivés.

Le principe du dropout est extrêmement simple. En phase d'entraînement, à chaque évaluation d'un nouvel exemple d'entrée, des neurones ont une certaine probabilité p d'être désactivé c'est-à-dire que leurs sorties y a une probabilité p d'être mise à 0. Plus formellement, si l'on considère $r = Bernoulli(p)$, une variable aléatoire tirée selon une distribution de Bernoulli et dont la valeur est égale à 1 avec une probabilité p et 0 sinon, alors la valeur d'un perceptron devient $y = r \times h(\sum w_i \times x_i + b)$. Dans la littérature, la probabilité de désactivation d'un neurone la plus communément utilisée est de 50%. En phase de test, les neurones sont activés en permanence ($r = 1$). Néanmoins, il est important que les neurones fournissent la même énergie

"moyenne" que durant l'entraînement. Par conséquent, la valeur de sortie de chaque neurone est multipliée par la valeur de la probabilité p : $y = p \times h(\sum w_i \times x_i + b)$.

Il y a plusieurs façons intéressantes d'expliquer pourquoi le dropout améliore l'apprentissage et prévient le sur-apprentissage. Premièrement, comme nous l'avons expliqué, un neurone ne peut pas être seul responsable dans la prédiction finale du réseau. En effet, les neurones étant régulièrement désactivés lors de l'entraînement (avec une probabilité p) la prédiction finale du réseau ne peut reposer sur uniquement l'un d'entre eux, les neurones doivent chacun prendre part aux prédictions en "collaborant". Une autre vision, tout aussi intéressante, est d'observer le réseau dans sa globalité. Lors de la désactivation aléatoire des neurones, on construit en fait un nouveau réseau inclus dans le réseau d'origine (voir figure 2.6). Entraîner un réseau avec du dropout revient donc, à chaque passe dans le réseau, à entraîner un réseau différent appartenant à l'ensemble des réseaux inclus dans le réseau d'origine. Le dropout peut donc être vu comme l'entraînement simultané de nombreux réseaux de neurones partageant des paramètres. Il a été démontré qu'en phase de test, le dropout correspond à la moyenne géométrique d'un nombre exponentiel de modèles faiblement entraînés [BS14]. Il a été plus particulièrement démontré que le dropout correspond à un terme de régularisation L2 supplémentaire dans la fonction de coût [BS14].

2.2.4 Normalisation par Batch

Une autre technique, appelée normalisation par batch, qui permet d'empêcher le sur-apprentissage et d'augmenter l'efficacité des réseaux de neurones a été introduite récemment par Ioffe et Szegedy [IS15b].

Lors de l'entraînement d'un algorithme sur une base, il est courant de normaliser (centrer et réduire) les données d'entrée afin que ces dernières aient une moyenne centrée en zéro et une variance à 1, permettant ainsi aux données d'être comparables à travers les caractéristiques. Ensuite, lorsque les données traversent un réseau de neurones les poids (ou paramètres) du réseau transforment les valeurs pouvant créer des valeurs intermédiaires décentrées et/ou trop importantes. Ce phénomène est appelé "internal covariate shift". Afin de résoudre cela, Ioffe et Szegedy exploitent les mini-batch vus dans la partie 2.2.1 en normalisant les données contenues dans ces derniers. Concrètement, leur module, qui peut être placé n'importe où dans le réseau, calcule la moyenne et l'écart type des valeurs obtenues lors de l'évaluation d'un mini-batch. Les valeurs sont ensuite normalisées en soustrayant la moyenne et divisant par l'écart type. Les différentes étapes sont décrites dans l'algorithme 2.

Algorithm 2: Calcul de la normalisation par batch décris par Ioffe et Szegedy [IS15b].

Data: Les valeurs de x à travers un mini-batch $\mathcal{B} = \{x_1, ..., x_n\}$;
Les paramètres γ et β
Result: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

$\mu_{\mathcal{B}} = \frac{1}{n}\sum_{i=1}^{n} x_i$ ▷ Calculs des moyennes du mini-batch

$\sigma^2_{\mathcal{B}} = \frac{1}{n}\sum_{i=1} n(x_i - \mu_{\mathcal{B}})^2$ ▷ Calculs de la variance du mini-batch

$\hat{x_i} = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$ ▷ Normalisation des activations

$y_i = \gamma\hat{x_i} + \beta = BN_{\gamma,\beta}(x_i)$ ▷ Mise à l'échelle et décalage des valeurs

Cette technique présente plusieurs avantages. Premièrement, elle normalise les activations ce qui permet au réseau d'obtenir une meilleure stabilité puisque ces dernières présentent de moins grandes variations. Le second avantage (et probablement le plus important) est que cette technique permet d'éviter le sur-apprentissage. En effet, de par la construction aléatoire des mini-batch, une même entrée sera normalisée différemment à chaque passage dans le réseau (puisqu'elle sera mélangée à différentes données). La normalisation étant différente, les activations résultantes seront différentes, créant, aux yeux du réseau, de "nouvelles" données.

2.3 Les premiers succès des réseaux de neurones

Nous avons vu précédemment les différents éléments de construction des réseaux de neurones convolutifs, ainsi que l'algorithme de descente de gradient qui permet leurs entraînement. Dans cette partie nous passeront en revue les différentes architectures de réseau les plus populaires, ayant permis d'importantes avancées en termes de reconnaissance d'images. Ces architectures doivent leur popularité à la base de données ImageNet [Den+09] liée à la compétition ILSVRC ("ImageNet Large Scale Visual Recognition Challenge").

2.3.1 La base de données ImageNet

Le succès des premiers réseaux convolutifs est liée à une base de données appelée ImageNet [Den+09]. Cette base contient un très grand nombre d'images (plus de 14 millions) annotées afin d'indiquer la présence d'objet sur ces dernières. Les classes utilisées pour annoter les objets sont tirées de la base lexicale WordNet [Fel05], présentant ainsi 1000 catégories différentes d'objets. En plus de la présence d'objets dans l'image, des vérités terrains ont été ajoutées sur un sous-ensemble d'images.

Ainsi, plus $1,2$ millions d'images possèdent des annotations indiquant les positions (boites englobantes) des objets. En 2015, des vidéos ont aussi été ajoutées aux données présentes dans la base permettant ainsi une tâche de détection d'objet à partir de vidéos.

Associée à la base de données, une compétition annuelle (finie depuis 2017) nommée ILSVRC ("ImageNet Large Scale Visual Recognition Challenge") [Rus+15] met en compétition les meilleurs algorithmes en classification d'image. Cette compétition propose plusieurs challenges, évoluant au fil des années, et allant de la détection d'objets à la détection d'objet à partir de vidéos. Néanmoins, les deux tâches présentes chaque année (et donc les plus populaires) sont la détection d'objets et la localisation d'objet.

La détection d'objets consiste, étant donnée une image, à prédire la classe de l'objet présent sur cette dernière. Dû au grand nombre de catégories d'objets présents (1000 classes différentes) la précision des algorithmes repose sur une précision du top 5. Cela veut dire que les algorithmes fournissent les 5 classes les plus probables indiquant la présence d'un objet ou non et une prédiction est considérée comme juste si la classe cible est présente parmi ces 5 prédictions. La localisation d'objets consiste à prédire la position des objets présents dans les images. Les algorithmes sont donc entraînés à prédire des boites englobantes (voir introduction chapitre 4).

Le nombre extrêmement élevé ainsi que la diversité des images de cette base en font un incontournable pour l'entraînement de réseaux convolutifs qui nécessitent beaucoup de données d'apprentissage.

2.3.2 Transfert de connaissances

La base de données ImageNet vu dans la partie précédente n'explique pas seule la popularité des réseaux ayant remporté la compétition. Un autre aspect important est la notion de transfert de connaissances, avec l'idée que les connaissances apprises par les réseaux lors de l'entraînement peuvent être utilisées afin d'améliorer l'entraînement sur d'autres tâches.

Nous savons que les réseaux de neurones convolutifs sont composés de plusieurs couches de convolutions qui s'enchainent les unes à la suite des autres. Cette structure en couches des réseaux part de l'idée d'une construction hiérarchique des objets. L'exemple souvent cité est celui de la reconnaissance de visage qui est réalisée en commençant par reconnaitre les traits et arrondis, puis ces éléments de base sont assemblés pour construire les yeux, la bouche et le nez qui sont eux-mêmes assemblés pour construire un visage complet. Grâce à Zeiler *et al.* [ZF14], qui ont

développé une technique permettant de visualiser quels sont éléments d'une image qui activent les neurones d'un réseau de neurones, cette construction hiérarchique a été corroborée par l'expérimentation. Ainsi, les premières couches de convolution des réseaux apprennent des caractéristiques générales qui, au fur à mesure que l'on descend dans les couches plus profondes, sont assemblées en éléments de plus de plus complexes et spécialisés à la tâche étudiée.

Or si les caractéristiques des premières couches sont générales, cela veut dire qu'elles peuvent être réutilisées afin d'entraîner un réseau sur une autre tâche. L'idée du transfert de connaissance est de récupérer ces caractéristiques apprises sur une grande base de données (telle que la base ImageNet vue dans la partie précédente) possédant beaucoup de diversité afin d'entraîner un réseau sur une nouvelle tâche dont le nombre de données disponibles est plus faible. Concrètement, un réseau dit "de base" est entraîné sur une base de données. La sortie du réseau est remplacée par une sortie adaptée à la nouvelle tâche (ayant, par exemple, un nombre de classes sémantiques différents), puis le réseau est entraîné à nouveau sur cette nouvelle tâche. Cela revient à initialiser le réseau avec les poids appris sur la première base de données. Cette technique, appelée "*finetuning*" en anglais, a permis à plusieurs équipes ([Ahm+08 ; Oqu+14]) d'obtenir des résultats meilleurs que ceux obtenus dans l'état-de-l'art (de l'époque). Yosinski *et al.* [Yos+14] ont fait une étude poussée montrant que le transfert de connaissance est toujours meilleur qu'une initialisation aléatoire des poids du réseau.

Un autre type de transfert, appelée distillation des connaissances sur les données a été étudié par Hinton *et al.* [Hin+15]. L'idée est d'utiliser un très grand réseau de neurones déjà entraîné afin de guider l'entraînement d'un autre réseau, plus petit, sur la même tâche. L'intérêt étant qu'un plus petit réseau est plus rapide et moins gourmand en ressource mémoire. Pour cela, le petit réseau est entraîné, étant donnée une image, à prédire les mêmes probabilités que celle obtenues par le grand réseau. En effet, en plus de la classe prédite, le vecteur de probabilité en sortie du réseau contient des informations intéressantes apprises par le réseau permettant de guider l'apprentissage du petit réseau. Par exemple, étant donnée une image de chat, le grand réseau pourra avoir une grande probabilité pour la classe chat, mais aussi une probabilité non nulle pour la classe chien alors que la classe voiture sera proche de zéro. On peut traduire cela par "la classe chat est plus proche de la classe chien que de la classe voiture".

Ces techniques de transfert de connaissances ont largement participé à la popularité des réseaux ayant remporté la compétition ILSVRC (vue dans la partie 2.3.1) puisqu'elles ont montré qu'il été possible de réutiliser ces derniers afin d'améliorer les performances obtenues avec d'autres tâches.

2.3.3 Des réseaux de plus en plus grands et profonds

Nous avons vu comment construire un réseau de neurones ainsi que les différents éléments permettant de l'entraîner efficacement. Ces éléments ont permis de construire des architectures de plus en plus grandes et de plus en plus profondes (possédant plus de couches d'opérations). Dans cette partie, nous ferons un historique chronologique des réseaux de neurones les plus populaires aujourd'hui en expliquant leurs évolutions architecturales.

LeNet5

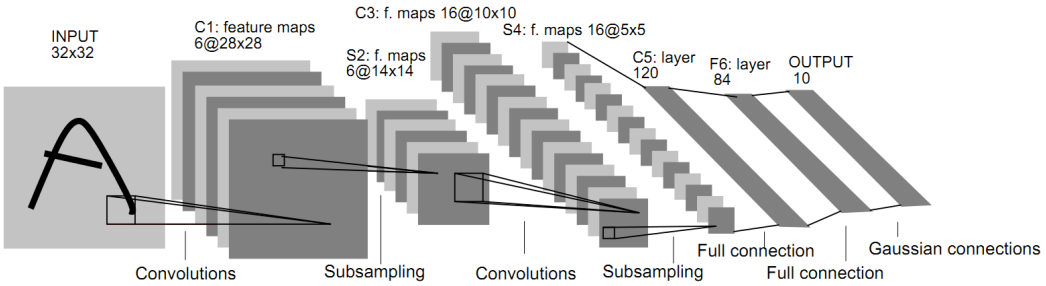


Fig. 2.7.: Schéma du réseau LeNet5, conçu par LeCun *et al.* [LeC+98] pour la reconnaissance de chiffre manuscrit.

Pour notre étude, nous partirons du réseau de neurones convolutifs créé par LeCun en 1990 [LeC+98], ce réseau étant le plus connu de la communauté et parfois considéré (à tort) comme le premier réseau de neurones. Ce réseau, nommée LeNet5 et dont le schéma de l'architecture est donné dans la figure 2.7, est conçu pour la reconnaissance de chiffres manuscrits. Ce dernier prend en entrée des images en niveau de gris (un seul canal de couleur) de taille 32×32 et produit un vecteur de sortie de taille 10 permettant de classer les images d'entrée dans une des 10 catégories possibles (les chiffres de 0 à 9).

Le réseau est composé de deux couches de convolution de masque de taille 5×5 et de sous échantillonnages de taille 2×2. La partie convolutive est suivie d'une partie de type réseau de perceptrons multi-couches composée de deux couches entièrement connectées ainsi que d'une couche de sortie dont les connexions suivent une probabilité gaussienne.

La taille extrêmement réduite du réseau est due à la fois aux limitations matérielles de l'époque, mais aussi aux faibles connaissances de l'époque qui rendaient difficile l'apprentissage d'un réseau trop complexe (avec trop de poids). C'est d'ailleurs ce dernier argument qui justifie l'utilisation d'une couche avec des connexions

gaussiennes puisque cette dernière permet de réduire le nombre de paramètres du réseau.

AlexNet

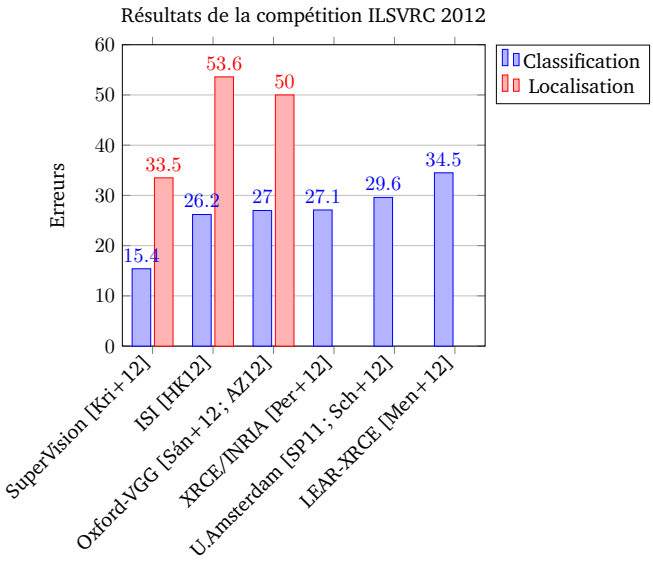


Fig. 2.8.: Résultats de la compétition ILSVRC 2012 [Rus+15] obtenus par les cinq meilleures méthodes.

Après la création du réseau LeNet5 en 1990, il aura fallu attendre 2012 pour que les réseaux de neurones convolutifs deviennent populaires. Cette popularité a été atteinte grâce au réseau de neurones convolutifs appelé AlexNet, développé par Alex Krizhevsky *et al.* [Kri+12]. Ce réseau leur a permis de remporter la compétition ILSVRC 2012 [Rus+15] (vue dans la partie 2.3.1) avec une précision de 84.6% (soit une erreur de $15,4\%$) quand les concurrents les plus proches obtenaient des précisions aux alentours des 73.8% (voir tableau 2.8).

L'architecture présentée par Krizhevsky *et al.* suit le même schéma que LeNet5 mais est nettement plus grande (avec plus de couches) et utilise des avancées plus récentes telles que la fonction d'activation ReLU (vue dans la partie 2.1.2) et l'opérateur de dropout (vue dans la partie 2.2.3).

La figure 2.9 montre le schéma du réseau AlexNet. Ce dernier est composé de cinq couches de convolution toutes suivies d'opérateurs de sous-échantillonnages de type maximum-pooling et de fonctions d'activation de type ReLU. La partie convolutive du réseau est suivie de deux couches entièrement connectées (perceptron multi-couches). Afin de limiter le sur-apprentissage, ils utilisent la technique de dropout

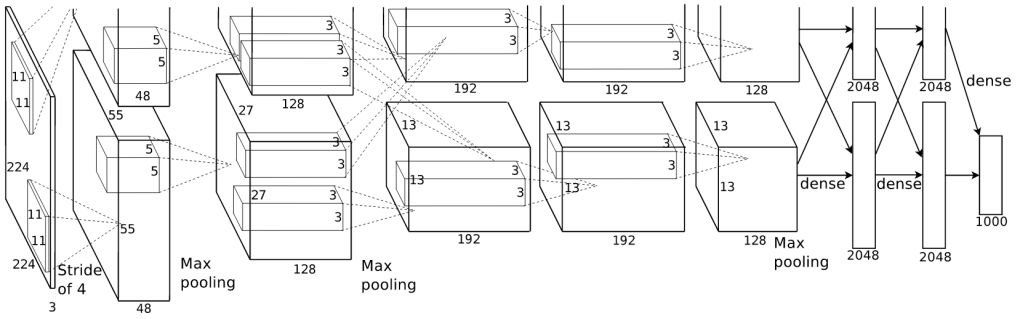


Fig. 2.9.: Schéma du réseau AlexNet. Figure extraite du papier d'origine [Kri+12].

dans les couches entièrement connectées avec une probabilité d'annulation de 50% (voir partie 2.2.3).

Une particularité intéressante due à la limitation mémoire des cartes graphiques de l'époque (GPU) est que le réseau est composé de deux chemins ne partageant des informations que lors de la deuxième couche de convolution et dans la partie entièrement connectée (voir figure 2.9). Cela permet d'entraîner le réseau sur deux GPU en parallèle tout en limitant les transferts d'informations entre eux, optimisant ainsi les temps de calculs.

On peut expliquer le bon fonctionnement de ce réseau (et donc de son succès) grâce à trois facteurs :

— L'utilisation d'opérateurs efficaces telles que les fonctions ReLU et l'opérateur de dropout.

— Les compétences techniques d'Alex Krizhevsky qui ont permis d'implémenter le réseau sur carte graphique, permettant d'accélérer le temps d'entraînement du réseau et donc de l'entraîner plus longtemps et plus efficacement.

— Le nombre important de données utilisées pour entraîner le réseau.

Ces trois facteurs réunis ont permis une avancée importante en reconnaissance d'images et ont contribué à la popularité actuelle des réseaux de neurones convolutifs.

VGG 16

En 2013, l'équipe Visual Geometry Group (VGG) de l'université d'Oxford a étudié l'architecture d'AlexNet afin de l'optimiser, proposant ainsi plusieurs modèles dont les architectures varient quant au nombre de couches utilisées. Parmi les modèles qu'ils ont étudiés, ils ont extrait une architecture appelée VGG-16 composée de 16

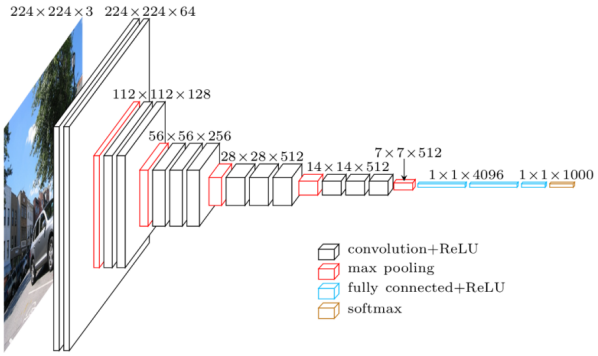


Fig. 2.10.: Schémas du réseau VGG16. Image tirée de la présentation de Matthieu Cord.

couches, avec laquelle ils ont gagné la compétition ILSVRC 2014 pour la tâche de localisation d'objets avec une précision de 92.7%.

Très proche du réseau AlexNet, leur architecture a la particularité d'utiliser plusieurs couches de convolutions successives sans opérateur de sous-échantillonnage (voir figure 2.10). De plus, la taille des masques de convolution est réduite à 3×3. De cette manière, le réseau possède moins de paramètres d'apprentissage mais plus de fonctions de non linéarité (puisque plus de couches de convolution). Cette particularité rend la fonction de décision du réseau plus discriminative tout en rendant l'apprentissage du réseau plus facile.

GoogleNet

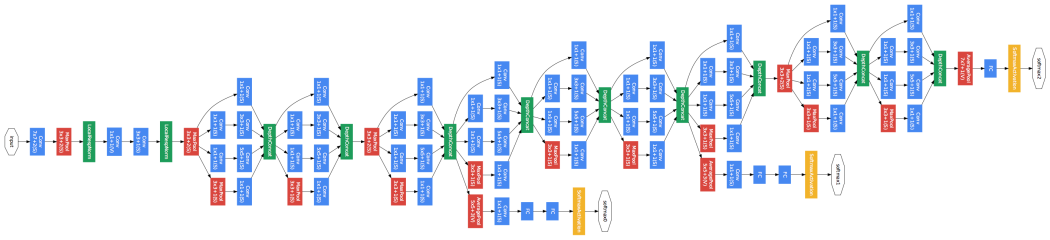


Fig. 2.11.: Schéma du complet réseau GoogleNet. La figure est extraite du papier [Sze+15a]

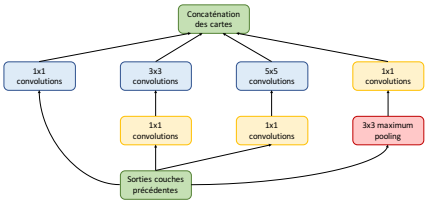


Fig. 2.12.: Module d'inception du réseau GoogleNet [Sze+15a]. Le module est composé de quatre chemins en parallèle, utilisant des convolutions avec des masques de tailles différentes.

GoogleNet est un réseau introduit par Szegedy *et al.* [Sze+15a], qui a gagné la compétition ILSVRC-2014 pour la tâche de classification avec une précision du TOP-5 de 93.3%. Leur architecture introduit un type de bloc appelé "inception module" (voir figure 2.12). Ces modules sont parfois considérés comme un réseau dans un réseau ("Network in Network" NiN) [Lin+13]. Ils sont constitués de quatre chemins en parallèle : un chemin avec une opération de sous-échantillonnage, un chemin avec une convolution large (avec des filtres 5×5), un chemin avec une convolution plus petite (avec des filtres 3×3) et un chemin avec un opérateur d'identité. Tous les chemins possèdent aussi une couche convolution de masque de taille 1×1 permettant de réduire le nombre de cartes de caractéristiques. Les cartes de caractéristiques obtenues par chacun des chemins sont ensuite concaténées et utilisées en entrée du module suivant. Le réseau étant particulièrement profond, des fonctions de coût intermédiaires sont ajoutées au milieu du réseau, permettant au gradient de remonter plus facilement (voir schéma du réseau 2.11)

2.3.4 Introduction des connexions résiduelles

Ajouter de plus en plus de couches dans un réseau permet de modéliser des fonctions plus complexes et donc possédant une capacité de discrimination plus importante. Les résultats empiriques sur les réseaux ont montré que cette profondeur améliore systématiquement les performances. Néanmoins, augmenter la profondeur d'un réseau fait apparaitre des limitations importantes. Premièrement, un réseau plus profond implique une modélisation d'une fonction plus complexe, ce qui entraine une perte de généralisation. Intuitivement, plus une fonction est expressive, plus elle peut apprendre par cœur les données d'entraînement et donc perdre en efficacité lorsque de nouvelles données (de test) sont utilisées. Deuxièmement, un réseau trop profond peut entraîner une augmentation de l'erreur et ce même sur les données d'entraînements. Une profondeur trop importante des réseaux entraîne des difficultés pour trouver un bon minimum de la fonction objectif. En effet, lorsque le réseau possède trop de couches de calculs, le gradient rencontre des difficultés pour remonter jusqu'aux premières couches du réseau. Ce phénomène, très connu aujourd'hui, est appelé problème de disparition du gradient (*vanishing gradient* en anglais). Il se caractérise par un gradient de plus en plus faible (avec des valeurs de plus en proche de zéro) au fur à mesure qu'il remonte dans les couches du réseau. Nous pouvons expliquer ce problème en regardant l'influence qu'ont les poids des différentes couches sur la sortie (et donc l'erreur) du le réseau. Dû à l'aspect de composition de fonctions d'un réseau, un petit changement dans les paramètres des premières couches du réseau entraînera un gros changement sur la sortie (un peu à la manière de l'effet papillon). Au contraire, une petite variation dans les paramètres des couches proches de la sortie aura beaucoup moins d'influence. Par conséquent, si l'erreur (le gradient) est distribuée uniformément sur l'ensemble des poids du

réseau, alors il est normal d'obtenir un gradient beaucoup plus faible sur les couches d'entrée.

Ce phénomène, est au cœur des supervisions intermédiaires du réseau GoogleNet vu dans la partie précédente qui permettent de "rapprocher" les couches intermédiaires d'une sortie et donc d'obtenir un gradient plus élevé sur ces dernières. Néanmoins, cela limite aussi la liberté d'abstraction des couches intermédiaires puisqu'en se retrouvant proche de la sortie elles doivent être capables de fournir des caractéristiques proches des prédictions.

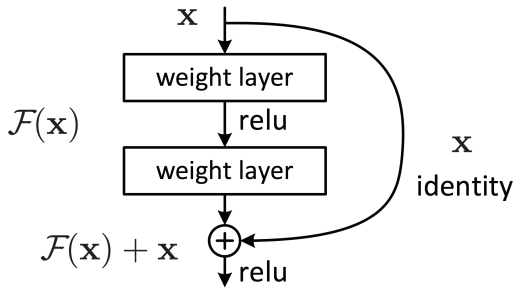


Fig. 2.13.: Illustration d'une couche résiduelle [He+16a]. L'entrée est additionnée aux résultats du mapping. Figure extraite du papier d'origine [He+16a]

Une solution élégante au problème de disparition de gradient, déjà existante dans les algorithmes de quantifications d'images a été introduite dans les réseaux de neurones convolutifs par He *et al*. [He+16a] et leur réseau appelé ResNet. Dans ce réseau ils utilisent des connexions appelées connexions résiduelles, qui consiste à ajouter l'entrée de la couche avec sa sortie (voir schéma 2.13). Formellement, si l'on considère $\mathcal{X}_i$ les cartes de caractéristiques à la couche i du réseau et $\Theta_i(\mathcal{X}_i)$ la transformation des cartes de caractéristiques de la couche i à la couche $i+1$, alors la transformation classique s'écrit sous la forme suivante :

$$\mathcal{X}_{i+1} = \Theta_i(\mathcal{X}_i)$$

Une couche résiduelle consiste à additionner l'entrée de la transformation au résultat obtenu. Ainsi la formule devient :

$$\mathcal{X}_{i+1} = \Theta_i(\mathcal{X}_i) + \mathcal{X}_i$$

Intuitivement, $\Theta_i(\mathcal{X}_i)$ ne correspond plus à l'opération de transformation de $\mathcal{X}_i$ à $\mathcal{X}_{i+1}$ mais à la différence entre les deux ($\Theta_i(\mathcal{X}_i) = \mathcal{X}_{i+1} - \mathcal{X}_i$). Ainsi, l'erreur induite par une couche résiduelle dépend de la fonction $\Theta_i(\mathcal{X}_i)$, mais aussi directement de l'entrée $\mathcal{X}_i$ permettant donc au gradient de remonter à travers cette connexion et donc de remonter plus facilement jusqu'aux premières couches. Grâce à cette technique, He *et al*. ont pu entraîner des réseaux extrêmement profonds allant jusqu'à

152 couches dans les travaux d'origines et montant, dans des travaux ultérieurs, jusqu'à plus de 1000 couches.

Par la suite, Huang *et al.* [Hua+16a] ont amélioré le réseau ResNet en montrant qu'une majorité des couches du réseau ne contribuent que très peu à la prédiction finale. Ils ont donc développé un sous-échantillonnage stochastique, coupant aléatoirement des blocs résiduels (mais en gardant la connexion d'identité) permettant d'entraîner des réseaux encore plus profonds (allant jusqu'à 1202 couches). Cette coupure stochastique lors de l'entraînement force chacune les couches à contribuer de manière équivalente. En effet, à la manière du dropout, aucune couche ne peut être déterminante dans la prise de décision du réseau puisqu'elles sont aléatoirement supprimées lors de l'entraînement, chacune doit donc participer à la prédiction finale tout en collaborant avec les autres couches.

En repartant de l'idée qu'aucune couche ne doit avoir trop d'influence sur la sortie finale du réseau, Srivastava *et al.* [Sri+15] ont développé, en parallèle des travaux sur ResNet, un réseau appelé Highway Networks. Leur réseau peut être vu comme une généralisation des ResNet puisque, au lieu d'additionner simplement l'entrée de la couche avec sa sortie, ils utilisent une fonction avec un facteur permettant l'interpolation entre les couches d'entrée avec les couches de sortie. De plus, le facteur d'interpolation est une couche de convolution supplémentaire dépendant des entrées. Concrètement la transformation de la couche i à la couche $i+1$ est obtenue par la formule :

$$\mathcal{X}_{i+1} = \Theta_i(\mathcal{X}_i) \times \Theta'_i(\mathcal{X}_i) + (1 - \Theta'_i(\mathcal{X}_i)) \times \mathcal{X}_i$$

Intuitivement, la fonction $\Theta_i(\mathcal{X}_i)$ permet de créer des abstractions et la fonction $\Theta'_i(\mathcal{X}_i)$ permet au réseau de décider quelles quantités de ces abstractions doivent être passées à la sortie. Avec cette structure ils parviennent à entraîner un réseau de plus de 100 couches.

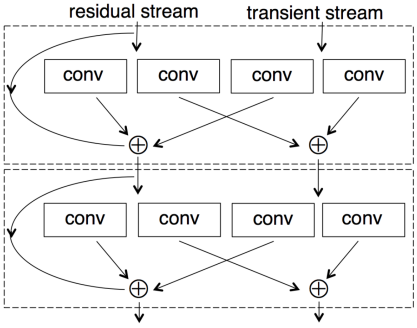


Fig. 2.14.: Illustration de deux couches RiR (ResNet in ResNet) consécutive. Schéma tiré du papier [Tar+16]

Suite à ces avancées sur les connexions résiduelles, Targ *et al.* [Tar+16] reprennent les modules d'inception de Szegedy *et al.* [Sze+15a] en y ajoutant des connexions résiduelles (voir figure 2.12). Leurs modules, appelés RiR (ResNet in ResNet) possèdent quatre couches de convolution générant deux sorties, l'une possédant une connexion résiduelle et l'autre non. Ainsi les couches de convolution reliées à la sortie non résiduelle ont la possibilité d'apprendre des abstractions importantes alors que les couches reliées à la sortie résiduelle permettent de faire remonter le gradient.

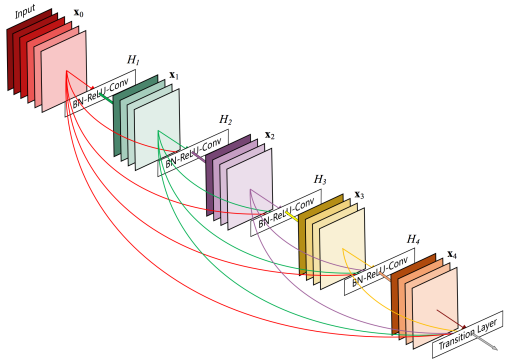


Fig. 2.15.: Illustration d'un bloc dense de 5 couches avec un facteur de croissance de 4. Schéma tiré du papier [Hua+16b]

Finalement, Huang *et al.* [Hua+16b] ont poussé les connexions résiduelles encore plus loin en créant un réseau appelé DenseNet, possédant des blocs résiduels denses. Dans ces derniers, chaque couche de convolution prend en entrée les cartes de caractéristiques extraites par toutes les couches précédentes (voir figure 2.15) créant ainsi des sauts de connexion entre les couches internes des blocs. Le calcul des cartes de caractéristiques de la couche $i+1$ correspond donc à une transformation prenant en entrée la totalité des couches précédentes :

$$\mathcal{X}_{i+1} = \Theta_i(\mathcal{X}_i, \mathcal{X}_{i-1}, ..., \mathcal{X}_1, \mathcal{X}_0)$$

On pourrait croire que, comme les couches les plus profondes du bloc utilisent les cartes de caractéristiques de la totalité des couches précédentes, le nombre de paramètres du réseau serait très important, alors que, de façon contre intuitive, ces blocs contiennent moins de paramètres qu'un réseau classique, car chaque couche de convolution produit le même nombre de cartes de caractéristiques. Ainsi, la couche au niveau l prend en entrée $k_0 + k \times (l-1)$ cartes de caractéristiques où k_0 est le nombre de cartes de caractéristiques d'entrée (généralement 3 pour les images RGB) et k est le nombre de cartes produites par chaque couche de convolution.

2.4 Conclusion

Dans ce chapitre nous avons vu les différents éléments de construction d'un réseau de neurones ainsi que les algorithmes utilisés pour les entraîner. Nous avons pu constater que le choix d'une architecture ainsi que de l'optimiseur et de ces hyper-paramètres demande une bonne compréhension de leur fonctionnement. Nous avons vu comment ont évolué les différentes architectures dont la tendance est d'ajouter toujours plus de couches de calculs, créant des réseaux de plus en plus profonds. Or cette profondeur, qui semble améliorer systématiquement les résultats, entraîne des difficultés dans l'entraînement des réseaux. C'est pourquoi de nombreux mécanismes, tels que les connexions résiduelles, ont été développés et font l'objet de nombreuses recherches.

Aujourd'hui, l'apprentissage profond a prouvé sa grande efficacité et a permis des avancées considérables dans de nombreux domaines. Parmi les très nombreuses contributions des réseaux de neurones, on peut notamment citer l'algorithme AlphaGo de l'équipe *DeepMind Technologies* qui, en 2015, a battu le champion du monde au jeu de Go, jeu dont la complexité en avait fait un challenge jusqu'alors non résolu. Ils ont aussi permis de faire des avancées importantes dans le domaine du traitement de la langue avec, par exemple, le traducteur DeepL [1] qui permet une traduction de langue automatique à l'aide de réseaux de neurones. Les réseaux de neurones sont aussi utilisés dans la recherche scientifique comme par exemple en physique des particules où ils servent à identifier les particules (muon, électron, B-jet,...) qui traversent un détecteur et à les catégoriser en fonction de la trace qu'elles laissent sur le capteur.

Néanmoins, confronter l'apprentissage profond à de nouvelles applications n'est pas toujours une tâche aisée. Le choix de l'architecture est une question très compliquée et demande souvent une connaissance importante des différents mécanismes et problèmes d'optimisation et de paramétrisation que l'on rencontre dans les réseaux. De plus, lors de l'étude d'une nouvelle application, il est nécessaire d'évaluer la complexité de la tâche ainsi que les qualités et nombre de données d'entraînement disponibles avant de décider d'une architecture, ce qui demande une certaine expertise. Un réseau trop complexe avec trop peu de données d'entraînement générera un phénomène de sur-apprentissage alors qu'un réseau trop simple pour une tâche plus complexe ne permettra pas d'obtenir les résultats optimaux. Néanmoins, nous avons vu qu'il est possible et souvent préférable de transférer les connaissances d'un réseau pré-entraîné sur une grande base plutôt que de créer et d'entraîner une architecture à partir de rien. Par conséquent, quand cela est possible, il est intéressant d'utiliser

1. https ://www.deepl.com/

les architectures que nous avons vues dans ce chapitre et de les spécialiser sur la nouvelle tâche.

3 Sous-échantillonnage mixte appliqué à la constance chromatique

3.1 La constance chromatique

Def. 3.1 *Constance chromatique*

Les méthodes de constance chromatique (ou "color constancy" en anglais) ont pour but de prédire les propriétés de réflectance des surfaces des objets composant une scène de manière indépendante des propriétés de l'illuminant (éclairage).

Lorsqu'une image est capturée par un appareil d'acquisition (appareil photographique, caméra vidéo, scanner etc.) ce dernier enregistre, à l'aide de capteurs photosensibles, les différentes intensités du spectre lumineux. La majorité des appareils d'acquisition ne capture que la partie visible du spectre lumineux en combinant les trois composantes rouge, vert et bleu (RGB). Dans le cas général, nous pouvons supposer que la distribution de lumière $I(\lambda)$ qui atteint le capteur (ou l'œil chez l'humain) en fonction de la longueur d'onde λ est égale au produit des propriétés de réflectance des objets $S(\lambda)$ et de la distribution de l'illuminant $E(\lambda)$.

$$I(\lambda) = E(\lambda) \times S(\lambda) \tag{3.1}$$

La perception de la couleur d'un objet dépendra donc des propriétés (constantes) de réflectance $S(\lambda)$ de ce dernier, mais également de l'illuminant $E(\lambda)$ qui éclaire la scène.

Chez l'être humain, le système visuel permet de percevoir les couleurs des objets (les propriétés de réflectance) indépendamment des couleurs ou des effets d'ombre de l'éclairage. En effet, un même objet éclairé par le soleil ou par un tube fluorescent (plus connu, à tort, sous le nom *tube néon*) semblera garder la même couleur. Si cette capacité, appelée *constance chromatique*, est naturelle chez l'être humain (ainsi que chez certaines espèces animales) nous verrons qu'elle présente une difficulté dans le domaine de la vision par ordinateur. Pourtant, être capable d'extraire des descripteurs indépendants de l'éclairage peut être déterminant pour le bon fonctionnement de nombreux algorithmes. Par exemple, un algorithme contrôlant la qualité des pièces

d'une chaine de production peut donner de bons résultats sous une lumière naturelle, mais avoir des résultats médiocres sous un éclairage artificiel.

3.1.1 Définition de l'objectif

L'objectif des méthodes de constance chromatique dans le domaine de la vision par ordinateur est de traiter des images de façon à les rendre invariantes aux propriétés des illuminants.

Les intensités lumineuses enregistrées par les périphériques d'acquisition ne dépendent pas uniquement des propriétés de réflectance des objets composants une scène, mais de plusieurs facteurs. Pour un canal $c \in \{R, G, B\}$, l'intensité lumineuse I_c mesurée par le capteur pour un pixel situé en (x, y) est notée $I_c(x, y)$. Pour une surface matte, elle est égale à l'intégration sur le spectre visible ω du produit de l'illuminant $E(\lambda)$, de la réflectance de la surface $S(x, y, \lambda)$ et de la sensibilité spectrale du capteur $C_c(\lambda)$ pour le canal c en fonction de la longueur d'onde λ :

$$I_c(x, y) = \int_{\omega} E(\lambda) S(x, y, \lambda) C_c(\lambda) \, \mathrm{d}\lambda \tag{3.2}$$

De cette équation, nous distinguons deux inconnues : la réflectance $S(\lambda)$ et les propriétés de l'illuminant $E(\lambda)$. Afin de traiter des images couleurs et de les rendre invariantes aux propriétés des éclairages, les méthodes de constance chromatique comportent généralement deux étapes. La première étape consiste à estimer l'éclairage $E(\lambda)$ afin qu'une seconde étape supprime l'influence de ce dernier. L'estimation de l'illuminant est donc une étape primordiale puisque la qualité de la correction dépendra directement de cette estimation. Obtenir la meilleure estimation possible de l'illuminant d'une scène est donc l'objectif principal d'une majorité des méthodes de constance chromatique.

3.1.2 Correction des couleurs

Lorsque que l'illuminant est connu (ou estimé), supprimer son influence d'une image revient à changer l'illuminant de la scène pour un illuminant standard de référence, souvent l'illuminant blanc. En effet, l'illuminant blanc correspond à un spectre lumineux complet. Ainsi une surface éclairée par un illuminant blanc renvoie une couleur due uniquement à ses propriétés de réflexion.

En étudiant le système visuel humain, Von Kries a proposé en 1902 un modèle d'adaptation sur lequel repose la majorité des algorithmes de constance chromatique

[Kri70]. Son modèle repose sur l'hypothèse que les sensibilités spectrales de l'œil sont indépendantes. En traitement d'image, cela revient à supposer que les différents canaux R,G,B sont indépendants, c'est-à-dire qu'ils sont sensibles à des parties du spectre lumineux qui ne se recouvrent pas.

La transformation permettant de modifier les valeurs R_1, G_1, B_1 des pixels d'une image d'origine vers les valeurs R_2, G_2, B_2, correspondants aux mêmes surfaces éclairées par un illuminant blanc, est réalisée en ajustant trois coefficients k_R, k_G et k_B, chacun associé à un canal différent et indépendant :

$$\begin{bmatrix} R_2 \\ G_2 \\ B_2 \end{bmatrix} = \begin{bmatrix} k_R & 0 & 0 \\ 0 & k_G & 0 \\ 0 & 0 & k_B \end{bmatrix} \times \begin{bmatrix} R_1 \\ G_1 \\ B_1 \end{bmatrix} \tag{3.3}$$

Les paramètres k_R, k_G et k_B sont calculés afin qu'une surface de référence R_e, G_e, B_e censée apparaître blanche dans l'image d'origine, apparaisse effectivement blanche (ait pour valeur $R = G = B = 1$) dans l'image transformée :

$$\begin{bmatrix} k_R & 0 & 0 \\ 0 & k_G & 0 \\ 0 & 0 & k_B \end{bmatrix} \times \begin{bmatrix} R_e \\ G_e \\ B_e \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \Leftrightarrow \begin{array}{r} k_R = \frac{1}{R_e} \\ k_G = \frac{1}{G_e} \\ k_B = \frac{1}{B_e} \end{array} \tag{3.4}$$

Or les différences chromatiques observées d'une surface censée être blanche sont dues à l'illuminant. Ainsi, R_e, G_e, B_e correspondent aux composantes couleurs de l'illuminant de l'image d'origine.

Comme chaque pixel subit la même transformation, il est nécessaire de supposer que l'illuminant est uniforme sur la totalité de l'image. Il est intéressant de noter que, de par la normalisation des paramètres k, il n'est pas nécessaire de connaitre l'intensité de l'illuminant d'origine.

3.1.3 Données linéaires et non linéaires

Lorsqu'une image est capturée à l'aide d'un périphérique d'acquisition, il est important de noter que ce dernier applique plusieurs transformations aux données brutes enregistrées par le capteur. Ces opérations ont pour objectif de rendre visuellement plus agréable le rendu final de l'image, par exemple en augmentant le contraste. Ce post-traitement est non linéaire.

La plupart des algorithmes de constance chromatique sont conçus pour être opérés sur les images linéaires (sans post-traitement) afin d'être intégrés directement dans

les appareils d'acquisition. Appliquer ces algorithmes sur des images non-linéaires (après post-traitement) peut donc dégrader la qualité des résultats. Lors de l'évaluation des algorithmes il est donc important de faire la différence entre les images linéaires et non-linéaires.

Bien que les transformations non-linéaires fassent partie des secrets des constructeurs et sont donc généralement inconnues, il est possible, à partir d'une image non linéaire (tel que les images JPEG obtenues en sortie des appareils photographiques), d'obtenir une approximation grossière de l'image brute enregistrée par le capteur. Par conséquent, les algorithmes de constance chromatique sont généralement testés à la fois sur des données linéaires et non-linéaires. En pratique, il a été constaté que les approches conçues pour les images linéaires donnent aussi de bonnes estimations sur les images non-linéaires.

Dans ce contexte, les approches basées sur l'apprentissage automatique, (telle que celle que nous présenterons dans la partie 3.3), présentent l'avantage de pouvoir fonctionner sur les deux types d'images puisque la phase d'apprentissage permettra de s'adapter au contexte.

3.1.4 Métrique

Pour évaluer la qualité des différents algorithmes de constance chromatique, il faut définir une métrique permettant de mesurer les erreurs faites par ces derniers.

Soit $\mathbf{e} = \{e_R, e_G, e_B\}$ le véritable illuminant d'une image (la vérité terrain) et $\hat{\mathbf{e}} = \{\hat{e}_R, \hat{e}_G, \hat{e}_B\}$ l'illuminant estimé. L'intensité de l'illuminant n'est pas recherchée, seule sa chromaticité (teinte) est nécessaire afin d'opérer une correction des couleurs (voir partie 3.1.2). Par conséquent, l'erreur utilisée est la mesure de l'angle entre les deux vecteurs $\mathbf{e}$ et $\hat{\mathbf{e}}$.

Def. 3.2 *Erreur angulaire*

L'erreur angulaire, notée $\epsilon(\mathbf{e_1}, \mathbf{e_2})$ entre deux vecteurs e_1 et e_2 est obtenue par :

$$\epsilon(\mathbf{e_1}, \mathbf{e_2}) = \cos^{-1}\left(\frac{\mathbf{e_1}^t \mathbf{e_2}}{||\mathbf{e_1}|| \times ||\mathbf{e_2}||}\right) \tag{3.5}$$

Où $\mathbf{e_1}^t \mathbf{e_2}$ est le produit scalaire entre le vecteur $\mathbf{e_1}$ et le vecteur $\mathbf{e_2}$ et $||.||$ est la norme euclidienne des vecteurs.

Ainsi $\epsilon(\hat{\mathbf{e}}, \mathbf{e})$ permet de mesurer une erreur entre les vecteurs de couleurs de la vérité terrain et de l'illuminant indépendante de leurs normes.

3.1.5 Bases de données

Afin d'évaluer les différents algorithmes de constance chromatique, il existe dans la littérature, différentes bases de données de tests. Ces bases de données sont constituées d'images associées aux vérités terrains des illuminants, c'est-à-dire la véritable valeur de l'illuminant qui éclaire chacune des scènes. Bien qu'il existe de nombreuses bases de données, nous nous sommes intéressés à cinq d'entre elles qui présentent des images naturelles de scènes intérieures et/ou d'extérieures. Nous avons choisi ces dernières car elles ont l'avantage de posséder un grand nombre d'images d'entraînement, ce qui est une condition essentielle pour le bon entraînement des méthodes d'apprentissage basées sur les réseaux de neurones.

SFU Gray-ball originale

La base de données SFU Gray-ball originale a été développée par Funt *et al.* [CF03] en 2003. Elle contient 11346 images extraites de 15 séquences vidéo. La vérité terrain est obtenue à l'aide d'une boule grise montée sur la caméra et visible sur chacune des images. La couleur grise de la boule étant connue, la chromaticité évaluée sur la boule permet d'obtenir directement la chromaticité de l'illuminant.

Les images de cette base étant extraites de séquences vidéo, elles sont souvent très similaires entre elles. Afin d'avoir une évaluation plus juste des algorithmes de constance chromatique, il est nécessaire de prendre en compte cette corrélation entre les images lors de la séparation de la base en ensembles d'entrainement et de test. Les algorithmes d'apprentissage sont donc entrainés 15 fois, en laissant à chaque fois un sous-ensemble d'images différent de côté, sous-ensemble utilisé pour vérifier les performances de l'algorithme sur des images non utilisées pendant l'entrainement. Cette technique très répandue en apprentissage automatique est appelée validation croisée ("cross-validation" en anglais). Cela permet d'être sûr que les images corrélées entre elles soient dans le même ensemble (entrainement ou test) et ainsi empêche d'avoir des résultats faussement bons dû à un sur-apprentissage.

De plus, la boule grise, dont est extraite la vérité terrain, est visible sur chaque image, il est donc important de la masquer, sinon le réseau pourrait apprendre à extraire la couleur de l'illuminant directement sur cette dernière. Nous faisons cela en dessinant un disque noir à l'emplacement de la boule.

SFU Gray-ball linéaire

Comme nous l'avons vu dans la partie 3.1.3, il est important de faire la différence entre les images linéaires et non linéaires. Les images constituant la SFU Gray-ball originale étant des images enregistrées par une caméra, cette dernière applique des traitements modifiant les valeurs des pixels. Ce sont donc des images dites non linéaires.

Une version de la base de données, appelée SFU Gray-ball linéaire est obtenue en appliquant aux images une approximation de la transformation inverse opérée par la caméra. Cette transformation, appelée transformation gamma, consiste à élever toutes les valeurs des pixels de l'image à la puissance 2.2. Une fois les images corrigées, les vérités terrains sont recalculées en utilisant la boule grise présente sur ces dernières.

La nouvelle base ainsi créée doit être utilisée avec les mêmes précautions que son équivalente originale, c'est-à-dire avec une validation croisée de 15 sous-ensembles et sans oublier de masquer la boule grise présente sur les images.

Color-Checker originale

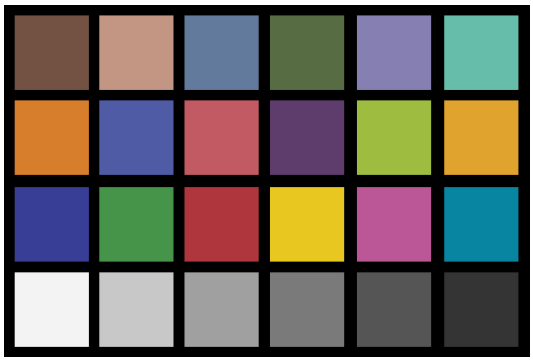


Fig. 3.1.: Mire de calibrage utilisée pour récupérer les informations des illuminants d'une scène. La couleur des différents carrés est connue, les différences chromatiques mesurées sont donc dues aux illuminants.

Une deuxième base de données dédiée aux problèmes de constance chromatique a été développée par Gehler *et al.* [Geh+08]. Ils ont collecté 568 photographies du monde réel avec deux appareils photographiques différents. Une mire de calibrage (voir figure 3.1) est incluse dans chaque photographie afin de récupérer la couleur de l'illuminant. Parmi les 568 images, 86 ont été photographiées avec un appareil photo de modèle Canon 1D, les autres 482 photographies étant prises avec modèle Canon 5D. Lors de l'évaluation d'un algorithme reposant sur l'apprentissage automatique,

les auteurs de la base recommandent d'utiliser une validation croisée avec trois sous-ensembles afin de créer des ensembles d'entraînements et de tests.

Color-Checker linéaire par Shi

De la même manière qu'avec la base de données SFU Gray-ball, la base Color-Checker originale contient des photographies non linéaires. Afin d'obtenir une base linéaire, l'équipe de Shi*et al*. [SF10] ont modifié la base d'origine en approximant une version linéaire des images. Suite aux modifications, les images capturées avec l'appareil photographique Canon 5D ont subi un décalage de leurs niveaux de noir. C'est-à-dire que la valeur des pixels correspondant à une couleur noire n'est plus à $R = G = B = 0$ mais à $R = G = B = 129$. Pour utiliser cette base il est donc important de soustraire cette valeur à tous les pixels de ces images afin d'être cohérent avec les images prises par l'appareil photographique Canon 1D.

Color-Checker retraitée

Finalement, une deuxième version retraitée de la Color-Checker a été fournie par l'équipe de Lynch *et al*. [Lyn+13]. Ils sont partis du constat que les images de la base Color-Checker linéaire par Shi possédaient une forte teinte anormale de cyan. Ils ont donc retraité leur propre version de la Color-Checker originale afin de linéariser les images.

Ils ont limité la nouvelle base aux 482 photographies obtenues avec l'appareil Canon 5D. Toutes les images ayant subi le même post-traitement, les erreurs d'approximations obtenues lors des modifications seront cohérentes entre elles.

3.2 État de l'art

Il existe trois grandes catégories d'algorithmes pour résoudre le problème de constance chromatique. La première catégorie contient les algorithmes les plus utilisés dans les dernières décennies et exploitent les propriétés statistiques des couleurs des images. La seconde catégorie regroupe les méthodes basées sur les propriétés physiques. Finalement, la troisième catégorie inclue les méthodes basées sur l'apprentissage automatique.

3.2.1 Méthodes basées sur les propriétés statistiques

Le patch blanc (*White patch*)

L'hypothèse faite par la méthode du patch blanc [CF99] est que, dans une scène, les valeurs maximales mesurées sont dues à un reflet direct de l'illuminant sur une surface. Intuitivement cela peut être comparé à une personne regardant une scène et étant éblouie par le reflet du soleil dans une vitre ou sur une surface blanche. La méthode du patch blanc consiste donc à récupérer les valeurs maximales obtenues pour chacun des canaux d'une image et de les considérer comme étant les valeurs de l'illuminant.

L'estimation de l'illuminant par la méthode du patch blanc est donc obtenue de la manière suivante :

$$\begin{aligned} \hat{e}_R &= max(I_R) \\ \hat{e}_G &= max(I_G) \\ \hat{e}_B &= max(I_B) \end{aligned} \tag{3.6}$$

Où $max(I_c)$ est la valeur maximale pour le canal $c \in \{R, G, B\}$ des pixels de l'image I.

Si la simplicité de la méthode est son point fort, elle possède néanmoins plusieurs faiblesses. En effet, le bruit des capteurs ainsi que les troncatures opérées par l'appareil, due aux saturations des capteurs (image surexposées), peuvent fausser les valeurs de l'image et impacter le résultat de l'algorithme. Dans une étude, Funt *et al.* [FS10] ont montré qu'en utilisant des images HDR (High Dynamic Range), qui sont des images capturées de manière à ne pas avoir de zones saturées, l'algorithme du patch blanc donne de bien meilleurs résultats, qu'avec des images classiques.

Chambah [Cha01] a proposé d'améliorer la robustesse de l'algorithme en affectant la moyenne des pixels dont la valeur est au-dessus d'un certain seuil plutôt que de récupérer uniquement la valeur maximale. Cela a pour effet de lisser les erreurs dues aux imperfections et traitement du capteurs.

Le monde gris (*Gray world*)

Si la méthode du patch blanc considère une approche locale en cherchant la couleur de l'illuminant dans des fortes réflexions, l'algorithme du monde gris de Buchsbaum et Gershon [Buc80], propose une approche globale. L'idée de l'algorithme a été inspiré par Helmholtz [VHS05] qui a émis l'hypothèse que le phénomène de constance

chromatique chez l'être humain repose sur notre capacité à nous adapter à un niveau moyen de couleurs.

L'algorithme du monde gris repose sur l'hypothèse que, sous un éclairage de référence, la moyenne des réflectances des surfaces des objets composant une scène doit donner une couleur proche du gris. Ainsi, sous un éclairage quelconque, l'écart de la moyenne des pixels d'une image par rapport à la valeur de référence est dû à l'illuminant. L'estimation de l'illuminant est donc obtenue de la manière suivante :

$$\begin{aligned} \hat{e}_R &= moy(I_R) \\ \hat{e}_G &= moy(I_G) \\ \hat{e}_B &= moy(I_B) \end{aligned} \tag{3.7}$$

Où $moy(I_c)$ est la moyenne des valeurs des pixels du canal c dans l'image I.

La faiblesse de la méthode repose sur l'hypothèse forte de la couleur de référence (couleur grise) d'une image. Par exemple, une image d'une forêt aura une teinte majoritairement verte qui ne sera pas due à l'illuminant, mais bien à la nature verte des surfaces photographiées. Si la couleur moyenne ne prend pas cela en compte et considère que le niveau moyen devrait être gris, l'estimation de l'illuminant sera faussée et l'image corrigée aura une teinte grise. Malgré cette faiblesse, l'algorithme du monde gris a démontré empiriquement des résultats satisfaisants et reste un des plus simples à utiliser. De plus, les résultats obtenus pour des images présentant une grande diversité des couleurs surpassent souvent certaines méthodes plus complexes.

Nuances de gris (*Shades of gray*)

Finlayson et Trezzi [FT04] ont amélioré l'algorithme du monde gris en considérant une généralisation de la fonction de moyenne en utilisant la norme de Minkowski ou L_p-norme.

L_p-norme

Def. 3.3 *Pour tout entier réel $p \geq 1$ la norme de Minkowski (ou L_p-norme) du vecteur $\mathbf{x}$ est égale à :*

$$||\mathbf{x}||_p = (\sum_i |\mathbf{x}_i|^p)^{1/p} \tag{3.8}$$

Cette formule, généralise plusieurs normes connues en fonction du paramètre p. Notamment, lorsque $p = 1$ alors la L_p-norme est égale à la distance de Manhattan, lorsque $p = 2$ il s'agit de la norme euclidienne et lorsque $p = \infty$ alors la L_p-norme est égale à la valeur maximale.

L'estimation de l'illuminant est obtenue en récupérant les normes L_p des composantes R,G,B d'une image :

$$\begin{aligned} \hat{e}_R &= ||I_R||_p \\ \hat{e}_G &= ||I_G||_p \\ \hat{e}_B &= ||I_B||_p \end{aligned} \tag{3.9}$$

Finlayson et Trezzi ont constaté que si l'on choisit $p = 1$ alors l'algorithme est équivalent à celui du monde gris (à une constante de mise à l'échelle près) et que si l'on choisit $p = \infty$ c'est équivalent à la méthode du patch blanc.

Ils ont ensuite étudié les meilleures valeurs de p pour différentes bases de données et ont conclu que $p = 6$ donnait généralement de bons résultats.

Les bordures grises (*Gray edges*)

Finalement, Van de Weijer *et al*. [Wei+07] ont proposé une généralisation supplémentaire des algorithmes vu précédemment en reprenant la formule des bordures grises et en exploitant la dérivées n-ième des images.

Leur, formule exprimée comme suit, correspond à la norme p de Minkowski de la dérivée d'ordre n de l'image, obtenue par convolution d'un filtre de dérivation Gaussien d'écart type σ.

$$\left(\int \left| \frac{\partial^n f_{c,\sigma}(\mathbf{x})}{\partial \mathbf{x}^n} \right|^p d\mathbf{x} \right)^{1/p} = k e_c, \quad c = \{R, G, B\}, \tag{3.10}$$

Le paramètre k est une constante de mise à l'échelle qui ne dépend pas du canal de couleur c.

L'équation signifie que la norme L_p de la dérivée n-ième des couleurs d'une image est liée à l'illuminant. Cette hypothèse est plus ou moins fondée en fonction des valeurs de n, p et σ. Dans leurs travaux, Van de Weijer *et al*. ont proposé plusieurs ensembles de valeurs possibles pour ces paramètres en fonction des bases de données considérées.

L'approche de Gao *et al*. [Gao+14]

La méthode qui a grandement inspiré nos travaux a été proposée par Gao *et al*. [Gao+14]. En plus d'être, au moment de nos recherches, la méthode présentant les meilleurs résultats, leur algorithme utilise une approche originale de la constance chromatique qui n'entre pas dans la famille des algorithmes généralisés par Van de

Weijer avec l'équation 3.10. Leur approche combine les informations locales de réflectance (patch blanc) à une estimation globale de l'illuminant (monde gris). C'est cette idée de comparaison d'informations à plusieurs niveaux (local et global) qui nous a paru intéressante et que nous avons voulu adapter aux réseaux de neurones.

L'originalité de la méthode de Gao *et al.* [Gao+14] est d'estimer directement les propriétés de réflectance des surfaces des objets en calculant le rapport entre des surfaces localement normalisées et une moyenne globale. Ils commencent par normaliser chacun des pixels de l'image par rapport à un voisinage local. Pour cela, ils divisent l'image en fenêtres de taille fixe et récupèrent les valeurs maximales de chacune des fenêtres. Les pixels de l'image sont ensuite normalisés en divisant leurs valeurs par la valeur maximale de la fenêtre à laquelle ils appartiennent. Cette image est appelée image des réflectances localement normalisées. Dans leur étude, ils ont testé plusieurs tailles de fenêtres possibles et ont montré qu'une taille de 20×20 obtenait les meilleurs résultats. L'obtention de l'estimation de l'illuminant se fait en divisant la moyenne des valeurs de l'image localement normalisées avec la moyenne de l'image d'origine. Ils ont montré à l'aide de plusieurs bases de données, que le rapport ainsi obtenus est comparable à la valeur de l'illuminant (à une constante de mise à l'échelle près).

La méthode ainsi développée possède un unique hyper-paramètre qui est la taille (en pixels) des fenêtres. Dans leurs travaux, Gao *et al.* ont fait un rapprochement entre leur méthode et les méthodes existantes. Ils ont notamment noté que, pour des valeurs extrêmes de la taille des régions, à savoir des régions de taille 1 pixel (respectivement une seule région de la taille de l'image) leur algorithme est équivalent à la méthode du monde gris (respectivement du patch blanc).

Leur méthode peut donc être considérée comme étant une combinaison de l'algorithme de monde gris et celui de patch blanc. C'est cette idée de combinaison que nous avons étudiée et que nous expliquons en détails dans la partie 3.3.

3.2.2 Méthodes basées sur les propriétés physiques

Afin de pouvoir expliquer le fonctionnement et les inconvénients des méthodes basées sur les propriétés physiques, il est nécessaire d'introduire les termes de réflexions spéculaires et diffuses (voir Figure 3.2). Une réflexion est dite spéculaire quand les rayons lumineux reçus par une surface sont renvoyés dans leurs quasi-totalité dans une seule direction (reflet de surface). À l'inverse, une réflexion est dite diffuse quand les rayons lumineux sont renvoyés dans toutes les directions après avoir été modifié par la surface. Les réflexions spéculaires correspondent à des reflets intenses et ponctuels et sont présents sur les surfaces brillantes. Les réflexions

diffuses donnent des surfaces éclairées uniformément tel que celles obtenues, par exemple, avec de la peinture mate.

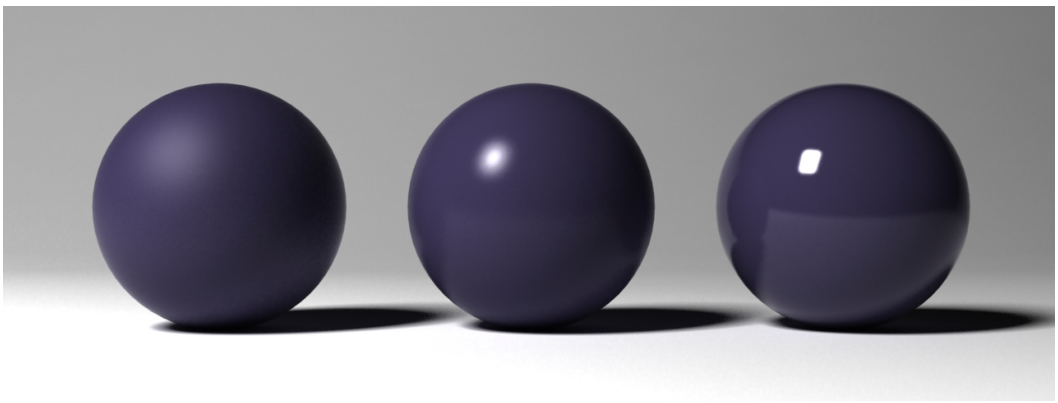


Fig. 3.2.: La sphère à gauche possède une réflexion diffuse tandis que la sphère à droite a une réflexion spéculaire. La sphère du milieu est une combinaison entre réflexion diffuse et spéculaire.

Les méthodes basées sur les propriétés physiques exploitent majoritairement le modèle de réflexion dichromatique proposé par Shafer *et al.* [FS01] qui intègre à la fois la réflexion diffuse et la réflexion spéculaire au modèle de la réflexion de la lumière.

Tan *et al.* [Tan+04] ont proposé un modèle qui se base sur les reflets spéculaires présents dans une scène. Ils ont montré que l'information colorimétrique obtenue sur des reflets spéculaires est fortement liée à l'illuminant. L'inconvénient étant que leur méthode repose essentiellement sur la qualité de la détection des reflets spéculaires.

De la même manière, l'approche proposée par Finlayson et Shaefer [FS01] requière une segmentation précise de l'image comme prétraitement afin que chaque surface de l'image soit correctement identifiée comme appartenant à une région uniforme.

Ces méthodes ont l'avantage d'être précises, mais reposent sur la détection des reflets spéculaires d'une scène, ce qui est une tâche complexe. En pratique, elles ne sont donc utilisables que dans des environnements calibrés, ce qui les rend difficilement utilisable pour des appareils d'acquisition destinés au grand public.

3.2.3 Méthodes basées sur l'apprentissage automatique

Conversion de gamut (*Gamut mapping*)

Le gamut de couleurs d'un éclairage correspond à l'ensemble des couleurs pouvant être observées sous un illuminant. Les méthodes basées sur la conversion de gamut élaborées par Forsyth [For90] visent à trouver le gamut des couleurs obtenues sous un illuminant blanc. La phase d'apprentissage consiste donc, à l'aide des bases d'images de références (voir partie 3.1.5), à définir l'ensemble des couleurs existantes afin de créer un gamut. Ce gamut est appelé gamut de référence. Ensuite, étant donnée une nouvelle image, la conversion de gamut consiste à estimer la meilleure transformation possible pour transférer le gamut des couleurs de l'image vers le gamut de référence.

Cette idée de base a ensuite été améliorée par Gijsenji *et al.* [Gij+10] qui, pour les mêmes raisons que la méthode des bordures grises (voir partie 3.2.1), utilise les dérivés de l'image.

La méthode de couleur par corrélation (color-by-correlation) proposée par Finlayson *et al.* [Fin+01] peut être comparée à la conversion de gamut. Néanmoins, au lieu d'utiliser des gamuts, ils utilisent une matrice qui mesure les corrélations entre les couleurs des illuminants et les chromaticités pouvant apparaître sous ces éclairages. La phase d'apprentissage consiste à construire la matrice à partir des vérités terrains associées aux images d'exemple. Etant donnée une nouvelle image, la matrice apprise permet, à partir des chromaticités de l'image, d'estimer les probabilités que chacun des illuminants présents dans la base d'exemples soit l'illuminant de la scène.

Basé sur l'exemple

Joze et Drew [JD14] ont proposé en 2014 de récupérer les images des bases de données d'entrainement (vue dans la partie 3.1.5) et d'en extraire des caractéristiques associées aux vérités terrains afin de créer une base d'exemples.

L'entraînement de leur algorithme, qui consiste donc à créer une base d'exemples, ce déroule en plusieurs étapes. Ils commencent par segmenter les images des bases d'entrainement en plusieurs régions de surface homogène notées R. De ces régions sont extraites trois informations $\{\mathbf{e}; \mathbf{p}; \mathbf{c}\}$ avec $\mathbf{e}$ la vérité terrain de l'illuminant de l'image, $\mathbf{p}$ une estimation de l'illuminant obtenu avec la méthode du patch blanc (voir partie 3.2.1) et $\mathbf{c}$ un vecteur de caractéristiques extraites sur la région considérée et

exploitant les informations de textures et de couleurs. Chaque triplet $R = \{\mathbf{e}; \mathbf{p}; \mathbf{c}\}$ constitue une entrée différente dans leur base d'exemples.

Une fois la base d'exemples construite, ils peuvent estimer l'illuminant d'une nouvelle image. Pour cela, ils commencent par segmenter la nouvelle image en plusieurs régions de la même façon que lors de la création de la base d'exemples. Pour chaque nouvelle région R_i, ils calculent le vecteur de caractéristiques $\mathbf{c}_i$ ainsi qu'une estimation $\mathbf{p}_i$ obtenue avec la méthode du patch blanc. En comparant le vecteur $\mathbf{c}_i$ aux vecteurs $\mathbf{c}$ présents dans la base d'exemples ils en extraient les 10 exemples $R_j, j \in [1, 10]$ les plus proches de la région R_i. Pour chacun des 10 exemples R_j ainsi obtenus, ils calculent une matrice de correction $\mathbf{M}_{ij}$ telle que $\mathbf{e} = \mathbf{M}_{ij} \times \mathbf{p}$. Cette matrice correspond à la transformation à appliquer pour corriger l'estimation $\mathbf{p}$ obtenue avec la méthode du patch blanc et obtenir le véritable illuminant $\mathbf{e}$. Ces matrices de corrections $\mathbf{M}_{ij}$ sont appliquées à l'estimation $\mathbf{p}_i$ faite sur la nouvelle région afin d'obtenir une estimation plus précise de l'illuminant $\mathbf{e}_i$:

$$\mathbf{e}_i = \frac{1}{10} \sum_{j=1}^{10} \mathbf{M}_{ij} \times \mathbf{p}_i$$

Cette méthode présente l'avantage de donner une estimation par régions permettant ainsi d'estimer plusieurs illuminants par image (un pour chaque région) dans le cas où l'hypothèse d'un illuminant uniforme n'est pas respectée. Dans leur évaluation, un seul illuminant est recherché par conséquent ils calculent la moyenne des illuminants des régions pour obtenir une estimation unique.

Cette méthode a été publiée lorsque nous étions en train de travailler sur le problème de la constance chromatique. À ce moment-là, elle présentait les meilleurs résultats sur les bases de tests. Néanmoins, cette technique, bien qu'efficace en termes de précision, présente deux désavantages. Premièrement, il est nécessaire de stocker tous les exemples dans une base de données ce qui rend la méthode lourde en termes de stockage mémoire. Deuxièmement, l'estimation d'un nouvel illuminant impose un processus de plusieurs étapes, telles que la segmentation de l'image (dont la qualité affecte grandement les résultats finaux), l'extraction de caractéristiques pour chacune des régions, la recherche depuis la base d'exemples des 10 plus proches voisins (pour chacune des régions) et finalement l'estimation de la couleur de l'illuminant. Ceci en fait une méthode complexe et trop lourde si l'objectif est de l'intégrer directement dans des appareils portatifs (appareil photographique ou caméra).

Réseaux de neurones

Cardei *et al*. [Car+02] proposent d'estimer l'illuminant d'une image à l'aide d'un réseau de neurones. Pour passer une image dans le réseau, ils commencent par convertir l'espace chromatique R,G,B de cette dernière en un espace r, g avec $r = R/(R+G+B)$ et $g = G/(R+G+B)$. L'espace r, g est ensuite échantillonné avec un pas fixe afin de construire un histogramme des paires r, g présentes dans l'image. La couche d'entrée du réseau possède un neurone par paire r, g échantillonnée. Les neurones sont ensuite activés ou non en fonction de la présence ou non d'une paire r, g dans l'image étudiée. Le réseau utilisé est un réseau de perceptrons multi-couches avec une couche cachée et deux neurones en sortie pour prédire l'illuminant (dans l'espace chromatique r, g). Ils entrainent ensuite leur réseau de manière classique par rétro-propagation (voir chapitre 2.2.2) en utilisant les vérités terrains des bases d'entrainement.

En même temps que nos recherches, une autre méthode basée sur les réseaux convolutifs a été développée par Bianco *et al*. [Bia+15]. Dans leurs travaux ils utilisent un réseau convolutif avec deux couches de convolution suivies d'un perceptron multi-couches avec deux couches cachées. Contrairement au réseau de Cardei *et al*., ils ont trois neurones en sortie afin de prédire la couleur de l'illuminant directement dans l'espace R,G,B. Pour entrainer leur réseau ils commencent par utiliser des sous-parties des d'images d'entrainement puis, une fois la convergence atteinte, ils affinent l'apprentissage en utilisant les images complètes. Cette approche donne de très bons résultats sur l'unique base de données (Color-Checker linéaire par Shi) qu'ils ont utilisée pour leurs expériences.

3.3 Sous-échantillonnage mixte appliqué à la constance chromatique

Dans nos travaux nous avons abordé le problème de constance chromatique à l'aide des réseaux de neurones. Nous sommes partis des travaux de Gao *et al*. [Gao+14] (voir partie 3.2.1) qui proposent une approche originale. En plus d'obtenir, à l'époque, les meilleurs résultats en termes de constance chromatique, leur méthode présente des caractéristiques intéressantes à exploiter à l'aide de réseaux de neurones tel que la comparaison d'informations obtenues à différents niveaux (local et global).

3.3.1 Sous-échantillonnage mixte

La méthode développée par Gao *et al.* [Gao+14] présente un intermédiaire entre une comparaison locale de l'information de réflectance et une information globale. L'information globale correspond à la méthode du monde gris et l'information locale à la méthode du patch blanc (voir partie 3.2.1). Nous sommes partis du constat que l'opérateur de sous-échantillonnage, que nous avons vu dans le chapitre 2.1.4 et qui est grandement utilisé dans les réseaux de neurones convolutifs, peut être équivalent aux méthodes du monde gris et du patch blanc.

L'average pooling (agrégation par valeur moyenne) agrège les valeurs d'une carte de caractéristiques en retournant la moyenne des valeurs sur une fenêtre d'observation de taille S (voir figure 3.3). La méthode du monde gris, qui considère la moyenne des valeurs des pixels comme étant la couleur de l'illuminant, est donc strictement équivalente à une opération de sous-échantillonnage dont la fenêtre d'observation serait égale à la taille de l'image.

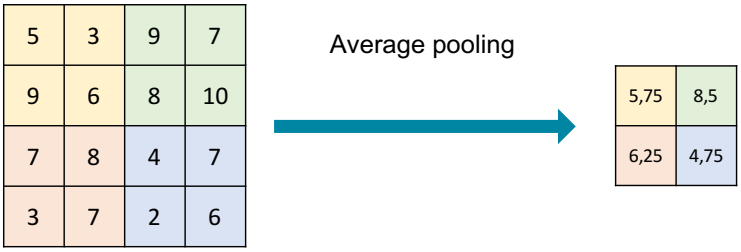


Fig. 3.3.: Exemple d'average pooling avec une fenêtre d'observation de 2×2.

De manière équivalente, le maximum pooling (agrégation par valeur maximale) est équivalent à la méthode du patch blanc. En effet, le maximum pooling consiste à agréger les valeurs d'une carte de caractéristiques en récupérant uniquement la valeur maximale dans une fenêtre d'observation de taille S (voir figure 3.4). Ainsi, si la fenêtre d'observation est de la taille de l'image, la valeur récupérée sera la valeur maximale dans l'image, ce qui correspond à la méthode du patch blanc.

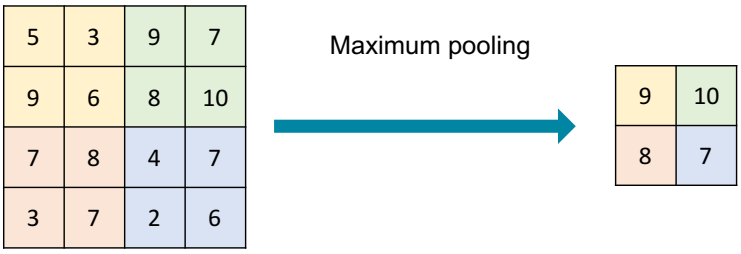


Fig. 3.4.: Exemple de maximum pooling. La fenêtre d'observation est de 2×2.

De plus, nous avons utilisé la même généralisation que celle faite par Finlayson et Trezzi pour la méthode des bordures grises (voir partie 3.2.1) à savoir l'utilisation

de la norme L_p. Pour rappel, la norme L_p d'un vecteur $\mathbf{x}$ de dimension d est calculée avec la formule suivante :

$$||\mathbf{x}||_p = (\sum_{i=1}^{d} |\mathbf{x}_i|^p)^{1/p} \tag{3.11}$$

Cette fonction généralise les fonctions de norme L_1 et de maximum pour $p = 1$ et $p = \infty$.

Finlayson et Trezzi ont montré qu'il peut être intéressant d'utiliser cette norme pour le problème de la constance chromatique. C'est pourquoi, nous avons utilisé un L_p pooling et testé plusieurs valeurs possibles pour le paramètre p.

Nous avons vu que les opérations de sous-échantillonnage peuvent généraliser certaines méthodes de constance chromatique et nous savons que l'intérêt de la méthode de Gao *et al*. est de comparer les informations à plusieurs niveaux. Nous nous sommes demandé comment, à l'aide des opérateurs de sous-échantillonnage, nous pouvions combiner de la même façon les informations. Pour cela, nous avons développé ce que nous avons appelé le sous-échantillonnage mixte (*mixed pooling* en anglais), qui a pour principe de combiner plusieurs opérateurs de sous-échantillonnage. L'image (ou les cartes de caractéristiques) d'entrée est donnée à deux opérateurs de sous-échantillonnage utilisés en parallèle et dont les sorties sont regroupées pour être utilisées dans la suite du réseau (voir figure 3.5). Afin de généraliser la méthode de Gao *et al*. nous avons commencé par utiliser en parallèle un maximum pooling et un average pooling. Nous avons ensuite testé plusieurs combinaisons possibles avec différent L_p pooling.

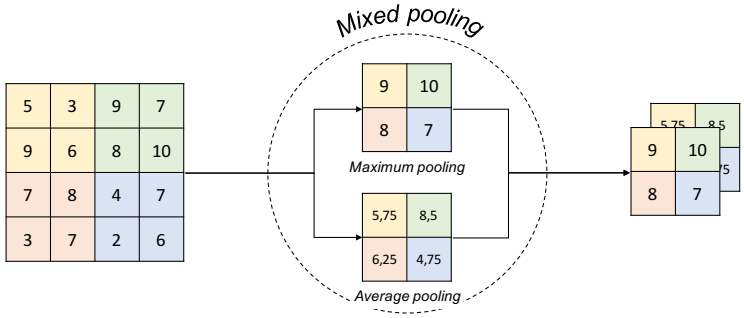


Fig. 3.5.: Illustration de notre sous-échantillonnage mixte. La fenêtre d'observation est de 2×2.

3.3.2 Réseaux de neurones avec sous-échantillonnage mixte

Pour tester notre sous-échantillonnage mixte, nous l'avons inclus dans deux architectures de réseaux de neurones différentes. La taille de la fenêtre d'observation a été choisie de manière à correspondre aux tailles des sous régions proposées par Gao

et al., à savoir 20×20. Une fenêtre d'observation aussi grande pour les opérateurs de sous-échantillonnage n'est pas commune dans les réseaux de neurones, mais permet de limiter le nombre de neurones utilisés pour le perceptron multi-couches. De plus, cela permet de capturer des informations locales plus cohérentes. En effet, la sortie de l'opérateur de sous-échantillonnage mixte est utilisée par un réseau de neurones. Si l'on avait utilisé une fenêtre de taille 2×2 (plus commune pour les opérations de sous-échantillonnage) le réseau aurait eu en entrée une image dans sa quasi-totalité, proche de l'image originale. A l'inverse, une taille de fenêtre trop grande ne donnerait pas beaucoup d'informations exploitables au réseau. Nous en sommes donc venus à la même conclusion que Gao *et al.*, à savoir qu'une fenêtre de taille 20×20 pouvait être un bon compromis.

Dans un premier temps, notre sous-échantillonnage mixte a été utilisé en entrée d'un réseau multi-couches classique. Les deux opérateurs de sous-échantillonnage utilisés en parallèle prenant en entrée l'image brute et transmettant au perceptron multi-couches les informations extraites. Le réseau est ensuite entrainé, par rétro-propagation, à combiner ces informations locales pour fournir une estimation globale de l'illuminant. Cette architecture est illustrée dans la figure 3.6.

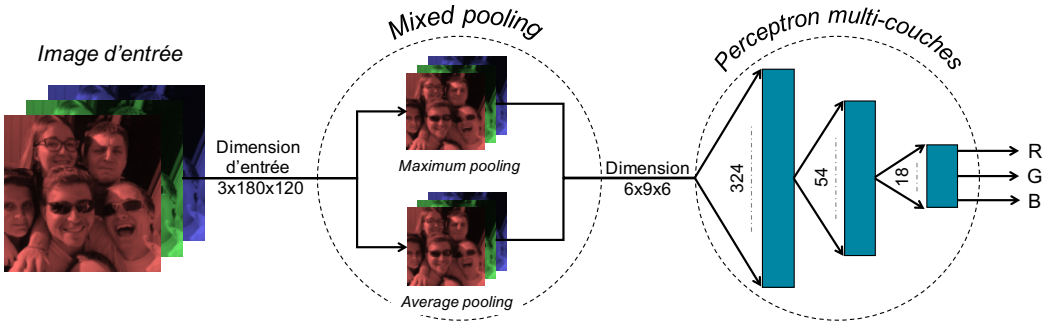


Fig. 3.6.: Notre architecture de réseau de neurones basée sur le sous-échantillonnage mixte avec un support large (20×20) combiné à un perceptron multi-couches.

Bien qu'une architecture de type perceptron multi-couches présente de bons résultats, elle ne permet pas d'extraire des caractéristiques spatiales de l'image. Or il a été démontré par la littérature que des informations, telles que les dérivées d'une image, peuvent être utiles (voir partie 3.2.1). Nous avons donc étendu l'architecture à un réseau de type réseau de neurones convolutif. Pour cela une couche de convolution a été ajoutée avant notre opérateur de sous-échantillonnage mixte. Dans les réseaux de neurones, les premières couches de convolutions ont pour rôle d'extraire des caractéristiques spatiales des images. Ainsi, en ajoutant cette couche de convolution en entrée du réseau, ce dernier a théoriquement la possibilité d'extraire la dérivée de l'image ou toutes autres caractéristiques utiles, généralisant ainsi d'autres familles d'algorithmes telle que la méthode des bordures grises vue dans la partie 3.2.1.

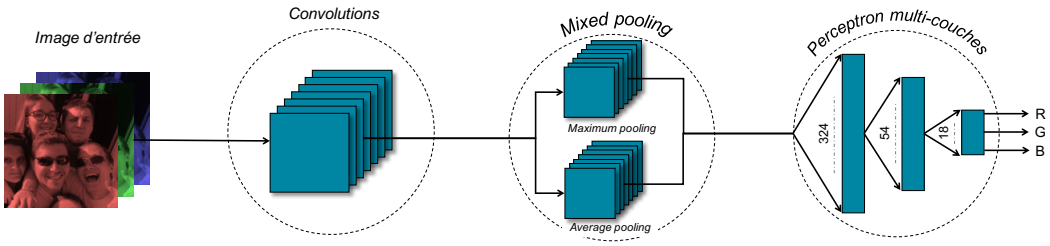


Fig. 3.7.: Architecture de réseau de neurones convolutif. L'image est passée dans une couche de convolutions, suivie de notre sous-échantillonnage mixte et enfin fourni à un perceptron multi-couches.

Pour nos expériences, nous avons utilisé un réseau avec 12 filtres de convolution de taille 7×7. En effet, la taille du masque de convolution est comparable au paramètre σ de l'équation 3.10 et la taille 7×7 est suffisamment large pour permettre d'apprendre des caractéristiques développées (telle que des dérivées avec un support large). Les cartes de caractéristiques obtenues après les couches de convolutions sont ensuite passées à notre opérateur de sous-échantillonnage mixte avant d'être empilées et utilisées par un perceptron multi-couches. L'architecture est présentée dans la figure 3.7.

3.3.3 Augmentation artificielle de données

Les réseaux de neurones sont connus pour nécessiter énormément de données d'apprentissage (du fait du nombre conséquent de paramètres du modèle). Par conséquent il est courant d'utiliser certaines astuces pour augmenter artificiellement le nombre de données présentes dans une base d'entrainement. Par exemple, les réseaux de convolutions n'étant pas invariants aux transformations de base telles que des translations et rotations, il est fréquent d'appliquer aléatoirement ce genre de modifications aux images d'entrainement créant ainsi, pour les réseaux, de nouvelles images d'entrées. Dans nos travaux, nous avons exploré trois méthodes différentes, décrites ci-dessous, pour augmenter artificiellement le nombre d'images d'entrainement.

Transfert d'illuminant

L'idée du transfert d'illuminant est de créer plusieurs exemples d'entrainement à partir de chacune des images présentes dans les bases d'entrainement et dont les vérités terrains sont connues. Pour cela nous utilisons la technique de correction des couleurs de Von Kries que nous avons vue dans la partie 3.1.2.

Le modèle d'adaptation de Von Kries est généralement utilisé pour retirer l'influence d'un illuminant d'une image. Cela revient à adapter l'image sous un illuminant d'origine pour lui donner l'apparence qu'elle aurait sous un illuminant blanc. Néanmoins, il est possible de modifier l'image pour lui donner l'apparence qu'elle aurait sous n'importe quel illuminant.

Pour cela, il suffit d'adapter la formule 3.3 de la manière suivante :

$$\begin{bmatrix} R_2 \\ G_2 \\ B_2 \end{bmatrix} = \begin{bmatrix} e2_R/e1_R & 0 & 0 \\ 0 & e2_G/e1_G & 0 \\ 0 & 0 & e2_B/e1_B \end{bmatrix} \times \begin{bmatrix} R_1 \\ G_1 \\ B_1 \end{bmatrix} \tag{3.12}$$

Où $[R_1, G_1, B_1]^T$ respectivement $[R_2, G_2, B_2]^T$ sont les couleurs des pixels d'origine respectivement transformées et $[e1_R, e1_G, e1_B]^T$ et $[e2_R, e2_G, e2_B]^T$ sont les illuminants d'origine et de cible. En utilisant cette technique, nous pouvons modifier les images existantes pour changer l'illuminant et ainsi créer de nouveaux exemples d'entrainement.

Néanmoins, mal utilisée, cette technique peut être néfaste à l'apprentissage. En effet, tous les illuminants ne sont pas présents dans une base d'entraînement. Par exemple, sous un éclairage naturel, il ne peut y avoir d'illuminant bleu. Modifier les images pour ajouter l'influence d'illuminants trop différents par rapport aux éclairages d'origine peut induire ce qu'on appelle un décalage de domaine (domain shift), c'est-à-dire avoir des vérités terrains trop éloignées des vérités terrains d'origine, ce qui pourrait obliger le réseau à s'adapter à des situations qui ne peuvent pas se produire. Afin d'éviter cela, nous simulons de nouvelles images à partir d'illuminants déjà présents dans la base d'entrainement. C'est-à-dire que, lors de la modification d'une image, l'illuminant cible est tiré aléatoirement dans les illuminants présents dans la base de données plutôt que créé aléatoirement. Ainsi le réseau n'aura pas à s'adapter à des illuminants incohérents. Grâce à l'utilisation de cette technique, une base de données possédant N images annotées peut donner lieu à $N(N-1)$ images.

On pourrait aussi limiter les différences entre les illuminants cibles et ceux d'origine afin d'éviter d'avoir des écarts trop importants, créant des décalages de contexte. Un décalage de contexte pourrait être, par exemple, d'utiliser un illuminant naturel (le soleil) dans une scène d'intérieur (éclairée par un tube fluorescent). Nous avons observé qu'en pratique, ce genre de limitation n'est pas nécessaire. Cela indique que le réseau n'utilise pas le contexte d'une scène pour prédire un illuminant. Ceci s'explique par la simplicité des réseaux utilisés, qui ne leur permet pas d'apprendre ce genre de caractéristiques trop complexes.

Fusion des patchs

La tâche de constance chromatique consiste à estimer la couleur de l'illuminant d'une scène en faisant l'hypothèse que l'illuminant est uniforme, c'est-à-dire qu'il n'y a qu'un seul illuminant qui éclaire la scène. Par conséquent, estimer la couleur de l'illuminant sur la totalité de la scène revient, théoriquement, au même que d'estimer l'illuminant sur une sous-partie de cette dernière. Dans nos travaux nous avons utilisé cela afin d'obtenir plus de données d'entrées.

Au lieu d'entrainer nos réseaux avec des images complètes, nous découpons des patchs de façon aléatoire dans les images. Ces patchs sont ensuite redimensionnés pour correspondre à la taille des images d'entrée du réseau. La vérité terrain associée à chacun des patchs est la même que celle de l'image complète. Durant l'étape de test, une image est découpée aléatoirement en plusieurs patchs. On obtient ainsi plusieurs estimations d'illuminants (une estimation par patch par image). L'estimation globale de l'illuminant est calculée comme étant la moyenne des illuminants obtenus sur chacun des patchs.

Apprentissage semi-supervisé

Les bases de données existantes pour la constance chromatique possèdent un faible nombre d'images (ou de nombreuses images très ressemblantes dans le cas de la SFU gray-ball). Le principe de l'apprentissage semi-supervisé est de mélanger aux images existantes des images dont la vérité terrain est inconnue. De telles images peuvent être trouvées facilement en grand nombre. Notamment, nous avons utilisé la base de données ImageNet [Den+09] (voir partie 2.3.1) possédant plusieurs millions d'images.

La vérité terrain sur cette base étant inconnue, il faut trouver une autre fonction de coût (voir partie 3.1.4) pour entrainer le réseau. Pour cela nous nous basons sur l'hypothèse que les scènes sont éclairées par un unique illuminant. Cette hypothèse implique que si l'on découpe l'image en deux parties, la valeur de l'illuminant, bien qu'inconnue, doit être identique sur chacune d'entre elles. Une fonction de perte basée sur la différence angulaire entre les deux vecteurs R,G,B des illuminants est donc utilisée, pénalisant le réseau si ce dernier prédit des illuminants différents pour les deux parties d'une même image.

3.3.4 Résultats

Les résultats que nous présentons dans cette partie ont été obtenus en 2014 avec l'API Torch7 [Col+11]. Les réseaux ont été entrainés jusqu'à convergence par rétro-propagation en utilisant la méthode RProp (Resilent Backpropagation) [RB93]. Depuis ces travaux, d'importantes avancées ont été réalisées à la fois sur l'entrainement des réseaux de neurones (comme par exemple la batch-normalisation [IS15b]) mais aussi pour les méthodes de constance chromatique. Ces nouveaux travaux seront brièvement présentés à la fin de ce chapitre.

Comparaison des différents pooling

Pooling	L^1	L^2	L^3	L^4	L^5	L^6	L^7	L^8	L^∞	Moy.	Moy/L^∞	L^5/L^∞
Color-Checker linéaire par Shi[SF10]												
Moyenne	7,30	4,57	3,78	3,55	3,42	3,49	3,45	3,33	**3,31**	3,77	3,4	3,33
Médiane	3,85	3,63	3,33	2,81	2,79	2,8	2,64	2,68	**2,59**	3,04	2,78	2,63
SFU Gray-ball linéaire[CF03]												
Moyenne	14,67	10,42	8,9	8,36	8,68	8,48	9,35	9,34	8,16	9,03	8,35	**7,65**
Médiane	13,44	8,95	7,59	7,47	7,7	7,62	8,92	8,58	7,08	7,8	6,98	**6,53**

Tab. 3.1.: Comparaison des influences des différents sous-échantillonnage sur les erreurs angulaire moyenne et médiane. Les tests sont effectués avec un réseau sans couche convolutive. L^∞ est le maximum pooling. L^5/L^∞ est notre sous-échantillonnage mixte avec deux pooling en parallèle.

Nous avons vu dans la partie 3.3.1 que plusieurs opérateurs de sous-échantillonnage étaient utilisables. Nous avons fait des tests afin de choisir la meilleure comparaison parmi ces derniers. Les résultats fournis dans le tableau 3.1 présentent les erreurs angulaires moyennes et médianes obtenues avec des opérateurs de sous-échantillonnage de type L_p pooling avec $p \in [1, 8]$ ainsi qu'avec un maximum pooling et un average pooling. Les deux dernières colonnes du tableau présentent les résultats obtenus avec un sous-échantillonnage mixte utilisant en parallèle un average pooling et un maximum pooling (noté Moy/L^∞) et un sous-échantillonnage mixte utilisant en parallèle un maximum pooling et un L_5 pooling (noté L^5/L^∞). Nous comparons les résultats sur deux bases de données différentes, la Color-Checker par Shi et la SFU Gray-ball linéaire (voir partie 3.1.5).

Les résultats obtenus avec la base SFU Gray-ball linéaire indiquent que le sous-échantillonnage mixte L^5/L^∞ donne les meilleurs résultats. Ceci n'est pas le cas sur la base Color-Checker by Shi dont l'utilisation du maximum pooling (L^∞) permet d'obtenir les meilleurs résultats. Néanmoins, le sous-échantillonnage mixte L^5/L^∞ obtient des résultats suffisamment proches pour penser que la différence d'erreur est négligeable, ce qui nous permet d'affirmer que le sous-échantillonnage mixte L^5/L^∞ est aussi une bonne stratégie sur cette base de données.

L'augmentation artificielle de données et comparaison des architectures

	Sans couche convolutive		Avec couche convolutive	
Methodes	Moyenne	Médiane	Moyenne	Médiane
Sans augmentation	**3,33**	**2,63**	3,91	3,06
Transfert d'illuminant	3,38	2,69	**3,49**	**2,73**
Fusion des patchs	3,66	3,00	3,78	2,95
Semi-supervisé	4,02	3,19	4,01	3,09

Tab. 3.2.: Effet des différentes stratégies d'augmentations artificielles de données sur la base de données Color-Checker linéaire par Shi. Le tableau compare les deux architectures de réseau, à savoir avec et sans couches de convolution.

Le tableau 3.2 présente l'impact des différentes méthodes d'augmentation artificielle de données sur les deux architectures proposées précédemment (voir partie 3.3.2).

Il est surprenant de constater qu'aucune des techniques d'augmentation artificielle de données n'améliore les résultats obtenus avec le réseau de perceptrons multi-couches. En effet, théoriquement, un plus grand nombre de données devrait permettre un meilleur apprentissage et donc devrait améliorer les prédictions. L'observation des résultats nous permet d'expliquer pourquoi ce n'est pas le cas. Premièrement, nous pouvons constater que la méthode de fusion des patchs dégrade les résultats. Cela indique que l'information extraite par le réseau est plus précise sur une image complète que sur une sous-partie de l'image. Le réseau de perceptrons multi-couches apprend donc à extraire des informations globales des images. Ainsi, plus l'image possède de diversité, meilleures sont les prédictions. À l'inverse, utiliser des sous-images possédant moins de diversité ne permet pas d'extraire des informations précises et mène à une prédiction moins fiable. On peut donc en conclure que le réseau de perceptrons multi-couches apprend un modèle comparable à celui du monde gris : un algorithme basique, reposant sur des propriétés statistiques simples.

Ensuite, le transfert d'illuminants donne des résultats équivalents à ceux obtenus sans augmentation de données. Cela permet de conclure que le modèle appris par le réseau de perceptrons multi-couches est effectivement basique et que ce dernier manque d'expressivité. Il atteint ses limites avec les images d'origines et augmenter le nombre d'images ne permet pas un meilleur apprentissage. Il est donc nécessaire d'utiliser un modèle plus expressif, c'est ce que nous avons fait avec le réseau convolutif.

Le premier constat des résultats obtenus avec le réseau convolutif est qu'ils sont moins bons qu'avec le réseau de perceptrons multi-couches. Ceci est étonnant puisque la plus grande expressivité du réseau devrait permettre un meilleur apprentissage. On peut cependant constater que la fusion des patchs et surtout le transfert d'illumi-

nants augmente significativement la précision du réseau convolutif. Cela démontre clairement un manque de données d'apprentissage permettant d'apprendre une bonne représentation mais met aussi en avant l'intérêt des techniques de fusion de patchs et de transfert d'illuminants pour les méthodes de constance chromatique. Nous en concluons que le développement de bases de données possédant un nombre d'images plus important serait grandement bénéfique aux techniques de constance chromatique basées sur les réseaux de neurones convolutifs.

Finalement, les résultats obtenus avec la technique d'apprentissage semi-supervisé sont, quelle que soit l'architecture utilisée, moins bons que les résultats obtenus sans augmentation de données. Ceci s'explique par la complexité de mise en œuvre de la technique, qui nécessite un équilibre très précis entre les données supervisées et non supervisées. Nous sommes convaincus de l'utilité de cette technique, bien qu'elle nécessite un investissement de recherche plus important.

Pour résumé, les résultats obtenus avec les méthodes d'augmentation artificielle de données nous permettent plusieurs affirmations :

— Le réseau de perceptrons multi-couches apprend à utiliser les statistiques globales des images.

— C'est l'expressivité du réseau de perceptrons multi-couches qui est un facteur limitant pour l'obtention de meilleurs résultats.

— Le réseau convolutif souffre d'un manque de données d'entrainement et pourrait obtenir de meilleurs résultats avec le développement de bases de données plus importants.

Comparaison avec l'état de l'art

La table 3.3 présente les erreurs angulaires moyennes et médianes obtenues avec les méthodes de l'état de l'art. La table est divisée en trois parties : premièrement les méthodes basées sur les propriétés statistiques (vu dans la partie 3.2.1) et donc sans apprentissage automatique, deuxièmement les méthodes basées sur l'apprentissage automatique et finalement les résultats obtenus avec notre méthode. Les résultats que nous comparons avec l'état de l'art sont ceux de l'architecture sans la couche de convolutions puisque les tests réalisés sur les différentes architectures ont montré que cette dernière est la plus efficace.

Nous comparons les résultats obtenus avec un unique maximum pooling et ceux obtenus avec notre sous-échantillonnage mixte. Les résultats montrent que notre méthode surpasse les méthodes sans apprentissage (telles que le monde gris, le patch

Méthodes	C-C-O	C-C-Shi	C-C-R	G-B-O	G-B-L
Monde gris [Buc80]	9.8 / 7.4	4.78 / 3.63	5.33 / 3.98	7.9 / 7.0	13.0 / 11.0
Patch blanc [FS10]	8.1 / 6.0	5.31 / 3.15	6.44 / 4.10	6.8 / 5.3	12.7 / 10.5
Nuances de gris [FT04]	7.0 / 5.3	4.40 / 2.72	3.98 / **2.35**	6.1 / 5.3	11.6 / 9.7
Monde gris général [Wei+07]	7.0 / 5.3	4.21 / 2.70	×	6.1 / 5.3	11.6 / 9.7
Bordures grises du premier ordre [Wei+07]	7.0 / 5.2	3.72 / 2.86	5.02 / 2.88	5.9 / 4.7	10.6 / 8.8
Bordures grises du second ordre [Wei+07]	7.0 / 5.0	3.59 / 2.64	×	6.1 / 4.9	10.7 / 9.0
Estimation locale de réflectance [Gao+14]	×	3.4 / 2.6	×	6.0 / 5.1	×
Large Col. diff. [Che+14c]	×	3.52 / 2.14	×	×	×
Patch gris [Yan+15]	×	4.6 / 3.1	×	6.1 / 4.6	×
Basé sur l'exemple [JD14]	**5.2** / **3.7**	3.1 / 2.3	×	**4.4** / **3.3**	**8.0** / **6.5**
Corr.-moments [Fin13]	×	**2.8** / 2.0	×	×	×**
SVRC-R [Li+14]	×	× / **1.97**	×	×	× / 6.81*
CNN [Bia+15]	×	**2.63** / **1.98**	×	×	×
Single max pooling	6.18 / 5.03	3.31 / 2.59	**3.69** / **2.70**	5.18 / 4.51	8.16 / 7.08
Mixed MaxL^5 pooling	**6.17** / **4.92**	3.33 / 2.63	**3.70** / 2.80	**4.94** / **4.28**	**7.65** / **6.53**

Tab. 3.3.: Comparaison avec l'état de l'art des méthodes existantes lors de nos travaux. La moyenne (en premier) et la médiane (en deuxième) des erreurs sont reportées. Pour chaque base de données, les deux meilleurs résultats sont en gras. '*' signifie testé sur un sous-ensemble (non corrélé) de données (1135 images parmi les 11346). '**' signifie que les résultats fournis par Finalayson [Fin13] ne peuvent pas être justement comparés puisqu'ils sont évalués uniquement sur 150 images (parmi les 11346) et n'utilisent que trois sous-ensembles pour la cross validation, ce qui ne respecte pas les 15 sous-ensembles utilisés par les autres approches. Les deux dernières lignes montrent nos résultats sans utilisation de couches de convolutions.

blanc ou l'estimation local de réflectance de Gao *et al.* [Gao+14]). Nos résultats sont comparables aux autres méthodes basées sur l'apprentissage automatique.

On peut noter aussi que l'utilisation d'un sous-échantillonnage mixte (avec un opérateur de maximum pooling en parallèle d'un opérateur de L^5 pooling) permet d'obtenir de meilleurs résultats qu'un unique maximum pooling. Cela confirme que

l'utilisation de plusieurs opérateurs en parallèle permet d'extraire des informations différentes et utiles pour le réseau.

3.4 Etat de l'art depuis les travaux

Dû à la popularité croissante des réseaux de neurones convolutifs, d'autres méthodes ont été développées depuis nos travaux afin de résoudre le problème de constance chromatique.

La méthode la plus simple a été développée par Qian *et al.* [Qia+16] qui extrait des caractéristiques d'une image en utilisant un réseau pré-entraîné sur ImageNet. Ils utilisent ensuite ces caractéristiques avec un algorithme de régressions structurées classique (dans leurs expériences ils utilisent des MSVR ("*multi-output support vector regression*") afin d'obtenir une prédiction). La simplicité de la méthode démontre la qualité des caractéristiques extraites par les réseaux de neurones.

Shi *et al.* [Shi+16] ont créé une structure de réseau spécialisé à la tâche de constance chromatique. Leur réseau est composé de deux réseaux, un appelé "Hypotheses Network" (HypNet) entraîné à produire deux estimations d'illuminant et un second, appelé "Selection Network" (SelNet) qui est entraîné à choisir la meilleure des deux estimations du premier. Étant donnée une image, le premier réseau (HypNet) propose ainsi plusieurs estimations locales. L'idée du second réseau (SelNet) est de choisir parmi ces estimations la meilleure. Les auteurs indiquent ainsi que le réseau SelNet permet d'obtenir de meilleurs résultats qu'un simple sous-échantillonnage.

Dans leurs travaux Barron et Tsai [BT17] ont étudié les aspects importants que doivent fournir les méthodes de constance chromatique afin d'être applicables directement dans les dispositifs d'acquisition (appareil photographique). Ils ont listé quatre aspects :

— La vitesse : les algorithmes doivent être rapides pour permettre un traitement rapide des images sans ralentir les dispositifs d'acquisition.

— Les entrées pauvres : les algorithmes doivent être capables d'utiliser des entrées en faible résolution (des prévisualisations) afin d'accélérer les traitements et surtout d'être applicable sur les prévisualisations généralement créées par les dispositifs d'acquisition.

— L'incertitude : un algorithme doit pouvoir donner une valeur de certitude sur ses prédictions pour éventuellement avertir l'utilisateur.

— La cohérence temporelle afin de permettre des prises de vues cohérentes pour les caméras vidéo ou les prises de photos en rafale.

Ils se basent sur les travaux précédents de Barron [Bar15], qui abordent l'estimation de l'illuminant comme une localisation de l'image dans un espace log-chromatique à deux dimensions, en transformant l'image dans l'espace de Fourier afin d'appliquer leur localisation dans cet espace. Le passage dans l'espace de Fourier leur permet de plier l'image d'entrée transformant la localisation dans un espace 2D en localisation sur un tore.

Même si la méthode de Joze et Drew [JD14] permet d'estimer plusieurs illuminants pour une image, leur algorithme repose sur les illuminants de régions segmentées et ne permet pas en pratique d'estimer de nombreux illuminants. Récemment, Duchêne *et al*. [Duc+17] ont proposé une méthode automatique permettant d'estimer de multiples illuminants par image d'entrée. Un outil de correction des couleurs a également été proposé, offrant un contrôle utilisateur intuitif et permettant d'obtenir des résultats de qualité.

3.5 Conclusion

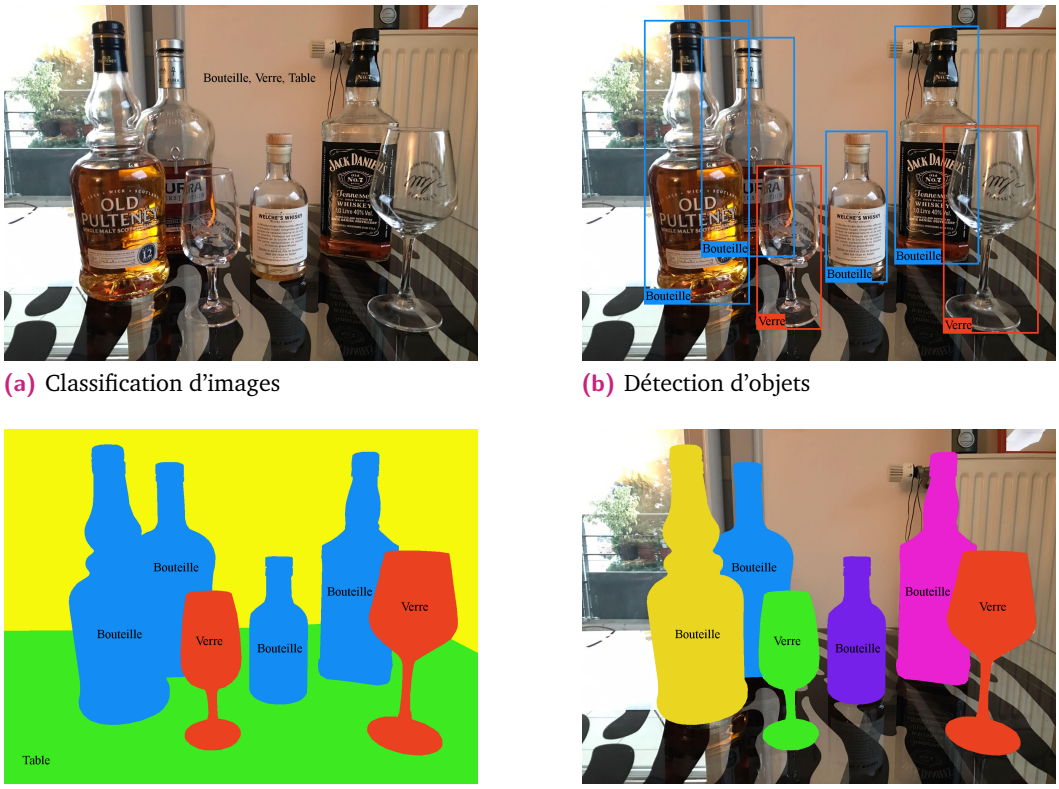
Dans ce chapitre nous avons fait un état de l'art des différents algorithmes de constance chromatique. En étudiant les méthodes classiques, basées sur les propriétés statistiques des images, nous avons montré comment l'opérateur de sous-échantillonnage permet à un réseau de neurones de modéliser ces méthodes. Puis, en partant des méthodes plus récentes, qui généralisent les algorithmes classiques, nous avons adapté l'opérateur de sous-échantillonnage afin de créer un opérateur de sous-échantillonnage mixte permettant au réseau de modéliser une fonction elle aussi plus générale, permettant donc une combinaison des différents algorithmes. Enfin, même si nous avons pu montrer l'efficacité de cet opérateur en utilisant plusieurs bases de données dédiées à la constance chromatique, nous avons découvert un manque crucial de données d'entraînement, ce qui nous a poussé à développer des techniques afin d'augmenter artificiellement leur nombre.

Finalement, dans ces travaux nous avons montré que, même si un réseau permet de modéliser des fonctions très complexes, il est intéressant d'adapter son architecture à la tâche étudiée. L'opérateur de sous-échantillonnage mixte que nous avons créé est une base intéressante pour la tâche de constance chromatique et il serait intéressant d'étudier les différentes adaptations réalisables. Notamment, l'évolution récente des réseaux convolutifs avec les architectures à prédiction dense (que nous verrons dans le chapitre 4) ainsi que les très populaires réseaux GAN (Generative Adversarial Network) ouvrent de nombreuses perspectives à la fois dans l'estimation de l'illuminant, mais aussi possiblement dans la création artificielle de nouveaux

exemples d'entraînement voir même la correction des images. Ces aspects, allant au-delà des objectifs fixés pour cette thèse, n'ont pas été abordés dans cette thèse.

Au-delà des problèmes de constance chromatique et d'invariance aux changements d'éclairage auxquels sont confrontés les descripteurs image bas niveaux, nous nous sommes également intéressés dans cette thèse aux descripteurs haut niveaux et plus précisément à la segmentation sémantique.

4 Segmentation sémantique d'images



(a) Classification d'images

(b) Détection d'objets

(c) Segmentation sémantique

(d) Segmentation sémantique d'instance

Fig. 4.1.: Comparaison des différentes tâches de reconnaissance d'images. De la tâche la moins précise à la plus précise : Classification d'image, Détection d'objets, Segmentation sémantique, Segmentation sémantique d'instance.

4.1 Description de la tâche

segmentation sémantique

Def. 4.1 *La segmentation sémantique d'images est une tâche qui consiste à attribuer une classe sémantique (par exemple* arbre*,* voiture*,* route *etc.) à chacun des pixels d'une image.*

Avec les progrès spectaculaires faits dans le domaine de l'apprentissage automatique et notamment des réseaux de neurones convolutifs, les résultats en reconnaissance d'objets ou compréhension de scènes sont devenus de plus en plus précis. La tâche de segmentation sémantique, qui est le sujet de ce chapitre s'inscrit dans un objectif

de reconnaissance d'objets de plus en plus complexe. On peut considérer que ses origines remontent à la tâche de classification d'image, dont l'objectif est, étant donnée une image, de prédire la classe sémantique du ou des objets présents sur cette dernière (voir figure 4.1a). Il s'agit donc d'une tâche de classification, puisque les valeurs de sortie appartiennent à un domaine discret et fini. Concrètement, la sortie du réseau consiste en un vecteur, de taille équivalente au nombre de classes sémantiques considérées et dont chacune des valeurs correspondent à la probabilité de l'image d'appartenir à la classe correspondante.

Le principal problème de la classification d'image est qu'elle ne permet pas la localisation des objets dans une scène, mais indique juste leur présence. Pour pallier ce problème, la tâche de détection d'objet vise à fournir les boites englobantes des objets présents dans une scène (voir figure 4.1b). Une boite englobante, en deux dimensions, est le rectangle le plus petit permettant d'englober un objet. Il est défini par quatre valeurs : les positions X, Y de la boite dans l'image (cette position correspond souvent au coin supérieur gauche de la boite) ainsi que la hauteur H et la largeur W de la boite. Il s'agit donc d'une tâche de régression puisque les prédictions recherchées sont des valeurs décimales, incluses dans un espace continu. En plus de la prédiction de position et de taille de la boite, il est possible de prédire en même temps la classe sémantique de l'objet détecté. C'est par exemple le cas pour l'algorithme YOLO [Red+16] (You Only Look Once) ou son évolution YOLO 9000 [RF16] de Redmon *et al*. qui fournit les boites englobantes de tous les objets d'une scène, mais aussi leur classe sémantique. Récemment, Mordan *et al*. [Mor+17] ont proposé une nouvelle architecture, appelée DP-FCN (Deformable Part-based Fully Convolutional Network), permettant d'améliorer la détection d'objets en apprenant une représentation basée sur les parties déformables des objets.

Bien que les boites englobantes permettent une localisation des objets dans une scène, ces dernières ne permettent pas d'obtenir la délimitation des frontières des objets et manquent donc de précision. Par conséquent, l'étape suivante vers une plus grande précision consiste à prédire la classe sémantique à laquelle appartiennent chacun des pixels de l'image (voir figure 4.1c). C'est l'objectif de la segmentation sémantique qui est la tâche étudiée dans ce chapitre. De la même manière que pour la classification d'image, la segmentation sémantique est une tâche de classification puisque le nombre de classes sémantiques considérées est fini. Néanmoins, on parle ici de classification dense puisque, au lieu d'obtenir un unique vecteur de prédiction, on cherche à obtenir un vecteur de prédiction pour chacun des pixels de l'image d'entrée.

Finalement, on peut pousser la précision encore plus loin avec la segmentation sémantique d'instances. En effet, la segmentation sémantique présente des problèmes de délimitation quand deux objets d'une même classe sémantique se superposent

(voir figure 4.1c). La segmentation sémantique d'instances vise donc à prédire non seulement la classe sémantique de chaque pixel, mais aussi de faire une différence entre les instances, c'est-à-dire que deux objets différents appartenant à la même classe sémantique auront deux valeurs différentes (voir figure 4.1d). Néanmoins, la segmentation d'instance n'est pas le sujet de ce mémoire où nous nous concentrons sur la segmentation sémantique.

4.1.1 Formalisation et fonction de perte

Soit $\mathcal{L} = \{l_1, l_2, ..., l_k\}$, un ensemble de K classes sémantiques (aussi appelées étiquettes). Soit $D = (x_i, y_i)$ un ensemble de paires, composées d'un patch d'entrée x_i (c'est-à-dire d'une image ou d'une sous-partie d'une image) et une classe cible $y_i \in \mathcal{L}$. L'objectif est d'apprendre les paramètres θ d'une fonction non linéaire $\Theta(x_i, \theta) = \hat{y}_i$, où $\hat{y}_i$ est une estimation de la classe cible y_i associée à l'entrée x_i, minimisant un risque empirique $R[\theta, D] = \frac{1}{N}\sum_i^N J(\Theta(x_i, \theta), y_i)$ où $J(\hat{y}_i, y_i)$ est une fonction de perte permettant de calculer une valeur correspondant à l'erreur entre l'estimation $\hat{y}_i$ et la classe cible y_i.

La fonction de perte utilisée pour calculer le risque empirique dépend de la tâche que l'on souhaite effectuer. Par exemple, nous avons vu dans le chapitre 3 que, pour la tâche de constance chromatique, la fonction de perte utilisée est l'erreur angulaire (voir partie 3.1.4). Bien qu'il existe de nombreuses fonctions pour la tâche de classification, la plus couramment utilisée est la fonction d'entropie croisée définie de la manière suivante :

Def. 4.2 *Cross entropy*

Soit p un vecteur de probabilité de taille N et y l'indice de la classe cible. La fonction de perte d'entropie croisée ("cross entropy" en anglais) est obtenue par la formule :

$$J(\mathbf{p}, t) = -\sum_j^N 1_{y=j} log(\mathbf{p}_j) = -log(\mathbf{p}_y) \tag{4.1}$$

Avec $1_{y=j} = 1$ si $y = j$ et 0 sinon.

L'entropie croisée est donc égale au négatif du logarithme de la probabilité prédite pour la classe cible. Cette dernière a pour paramètres un vecteur de probabilités $\mathbf{p}$ ainsi que l'indice de la classe cible t. Afin de l'utiliser lors de l'entrainement des réseaux de neurones convolutifs, il est nécessaire de permettre aux réseaux la prédiction d'un vecteur de probabilités. Pour cela, une fonction appelée *softmax* est ajoutée en sortie du réseau ayant comme objectif la normalisation des valeurs tel que la somme des valeurs soit égale à 1.

La fonction softmax est définie de la manière suivante :

Def. 4.3 *SoftMax* *Soit* $\mathbf{u}$ *un vecteur de taille* N *pour tout* $i \in \{1..N\}$ *:*

$$softmax(\mathbf{u})_i = \frac{e^{u_i}}{\sum_{j=1}^{N} e^{u_j}} \tag{4.2}$$

Si l'on considère l'exponentielle présente dans la fonction de softmax et le logarithme de la fonction d'entropie croisée, il est possible (et mathématiquement juste) de combiner les deux afin d'optimiser les calculs.

Def. 4.4 *Soit* $\mathbf{u}$ *un vecteur de taille* N *pour tout* $i \in \{1..N\}$*, la combinaison d'une fonction de softmax et d'une entropie croisée est égale à :*

$$J(\mathbf{u}, t) = -u_t + log(\sum_{j'=1}^{N} e^{u_{j'}}) \tag{4.3}$$

Cette fonction, applicable directement sur le vecteur produit par un réseau de neurones, permet d'obtenir l'erreur entropique du réseau sur les exemples d'entraînement. C'est cette erreur qui est minimisée par descente de gradient (voir partie 2.2.1) et qui permet l'entraînement des réseaux de neurones convolutifs.

4.1.2 Mesure des performances en segmentation

Dans une tâche de régression, la fonction de perte utilisée pour entraîner le réseau est généralement identique à la mesure d'évaluations des algorithmes. Ce n'est pas le cas pour les problèmes de classification. En effet, un classifieur est entraîné en optimisant une erreur reposant sur les probabilités obtenues sur les exemples d'entraînement, alors que la performance de ce même classifieur repose sur le nombre d'exemples correctement classés. Les mesures détaillées dans cette section sont des mesures de performances qui permettent d'évaluer la qualité d'un classifieur et ne doivent pas être confondue avec les fonctions de pertes vues dans la section précédente 4.1.1 qui servent à entraîner le classifieur.

Matrice de confusion

Pour faciliter la visualisation et la compréhension des différents résultats d'un réseau de neurones (ou d'un classifieur quelconque), on construit généralement une matrice appelée *matrice de confusion*. Cette dernière répertorie et classe les différentes prédictions faites par le réseau en fonction de leurs classes cibles.

Def. 4.5 *matrice de confusion* *Soit* k *le nombre de classes sémantiques à inférer, la matrice de confusion* M *est une matrice de taille* $k \times k$ *dans laquelle est rangé chacun des exemples d'apprentissage de*

manière à ce que chaque ligne représente la classe prédite par un classifieur et chaque colonne la classe cible.

		Vérité terrain		
		Avion	Bateau	Voiture
Prédiction faite par le réseau	Avion	11	6	2
	Bateau	4	12	1
	Voiture	1	0	21

Tab. 4.1.: Exemple de matrice de confusion où la tâche consiste à séparer trois classes sémantiques : *Avion*, *Bateau* et *Voiture*. La case rouge représente les vrais positifs (TP) de la classe *Avion*, les cases vertes l'ensemble de faux positifs (FP) et les cases bleus l'ensemble des faux négatifs (FN).

Le tableau 4.1 montre un exemple de matrice de confusion où la tâche consiste à distinguer les classes *Avion*, *Bateau* et *Voiture*. Dans cet exemple, la base contient 16 images de classe *Avion*, parmi lesquelles 11 ont été correctement classées, 4 ont été classées comme appartenant à la classe *Bateau* et une a été classée comme appartement à la classe *Voiture*. De plus, 6 images de la classe *Bateau* ainsi que 2 images de la classe *Voiture* ont été faussement rangées dans la classe *Avion*.

De manière générale, notons n_{ij} la valeur de la matrice M à la ligne i et à la colonne j. Cette valeur correspond au nombre d'exemples d'entraînement classés dans la classe i et appartenant à la classe j. Pour une classe i donnée, on appelle :

— **TP** (*true positif*) le nombre d'exemples correctement classés, et donc $TP_i = n_{ii}$.

— **FP** (*false positif*) le nombre d'exemples classés comme appartenant à la classe i alors que ce n'est pas le cas. Il est obtenu par la formule $FP_i = (\sum_{j=1}^{k} n_{ij}) - n_{ii}$.

— **FN** (*false negative*) le nombre d'exemples classés comme n'appartenant pas à la classe i alors que c'est le cas. Il est obtenu par la formule $FN_i = (\sum_{j=1}^{k} n_{ji}) - n_{ii}$

Mesure des performances par classe

À partir des vrais positifs (TP), faux positifs (FP) et faux négatifs (FN) décrits ci-dessus, on définit plusieurs mesures de performances. Bien qu'il en existe de nombreuses, nous ne listons ici que les principales utilisées.

La *précision* d'une classe i est le rapport des vrais positifs sur l'ensemble des échantillons rangés dans la classe i :

Def. 4.6
Précision

$$\mathrm{Pre}_i = \frac{TP_i}{TP_i + FP_i} = \frac{n_{ii}}{\sum_{j=1}^{k} n_{ij}} \tag{4.4}$$

La précision d'une classe i correspond à la confiance que l'on peut accorder au classifieur quand ce dernier prédit qu'une image d'entrée appartient à cette classe i. Dans notre exemple (table 4.1), 19 images ont été rangées dans la classe *Avion* mais seulement 11 d'entre elles appartiennent réellement à cette classe. Par conséquent, la précision est de $\frac{11}{19} = 0.57$ ce qui indique que si le réseau prédit qu'une image contient un avion, il y a 57% de chance qu'il ait raison.

Le *Rappel* de la classe i est le rapport des vrais positifs sur l'ensemble des échantillons appartenant réellement à classe :

Def. 4.7
Rappel

$$\mathrm{Rap}_i = \frac{TP_i}{TP_i + FN_i} = \frac{n_{ii}}{\sum_{j=1}^{k} n_{ji}} \tag{4.5}$$

Le rappel de la classe i correspond à la proportion d'exemples de la classe i qui ont été correctement classés. Dans notre exemple, le rappel de la classe *Avion* correspond au pourcentage d'exemples d'entraînement appartenant à la classe *Avion* qui ont été correctement classés par le classifieur. La matrice 4.1 nous indique que, parmi les 16 images d'avions, 11 ont été correctement rangées. Par conséquent, le rappel de la classe avion est de $\frac{11}{16} = 0.68$.

La précision et le rappel sont deux mesures de performances non corrélées, dont l'utilisation dépend généralement de la tâche. Par exemple, si l'on veut détecter des tumeurs on va probablement vouloir privilégier le rappel, afin que toutes les tumeurs soient bien détectées par le réseau de neurones, quitte à avoir des faux positifs et donc une mauvaise précision. Un des problèmes liés à l'absence de corrélation de la précision et du rappel est qu'il peut être difficile de comparer les performances des différents algorithmes (on peut avoir une bonne précision, mais un mauvais rappel). Pour cela, il existe une mesure, appelée *F-mesure* permettant d'obtenir un compris en effectuant une moyenne harmonique entre précision et rappel, en fonction d'un paramètre β :

Def. 4.8
F_β-mesure

$$F_{\beta i} = \frac{(1+\beta^2) \times (\mathrm{Pre}_i \times \mathrm{Rap}_i)}{\beta^2 \times \mathrm{Pre}_i + \mathrm{Rap}_i} = \frac{(\beta^2+1)TP_i}{(\beta^2+1)TP_i + \beta^2 FN_i + FP_i} \tag{4.6}$$

La F-mesure la plus communément utilisée est la F_1-mesure où $\beta = 1$ et qui donne autant d'importance à la précision qu'au rappel :

Def. 4.9
F_1-mesure

$$F_{1i} = 2 \times \frac{\text{Pre}_i \times \text{Rap}_i}{\text{Pre}_i + \text{Rap}_i} = \frac{2 \times TP_i}{2 \times TP_i + FN_i + FP_i} = \frac{2 \times n_{ii}}{\sum_{j=1}^{k} n_{ij} + \sum_{j=1}^{k} n_{ji}} \tag{4.7}$$

Mesures des précisions globales

Les mesures de performances vues dans le paragraphe précédent sont des mesures unitaires. Elles permettent de mesurer les performances d'une unique classe en comparaison aux autres (détection, classification binaire ou "*one versus all*"). Pour mesurer des performances sur un ensemble de plusieurs classes, comme c'est généralement le cas en segmentation sémantique, il faut utiliser des mesures globales. Pour la segmentation sémantique, ces mesures reposent sur le nombre de pixels correctement classés. En effet, les prédictions faites pour chacun des pixels sont considérées et sont rangées de la même manière dans une matrice de confusion.

La mesure la plus simple consiste à étendre la mesure du rappel à l'ensemble des classes. Elle correspond au nombre de pixels (et donc d'exemple) correctement classés, divisé par le nombre total de pixels.

Def. 4.10
Pixel accuracy

Soit k le nombre de classes considérées et n_{ij} le nombre de pixels de la classe i prédit comme appartenant à la classe j. La mesure du rappel global (ou pixel accuracy*) est égale à :*

$$PA = \frac{\sum_{i=1}^{k} n_{ii}}{\sum_{i=1}^{k} \sum_{j=1}^{k} n_{ij}} \tag{4.8}$$

Cette mesure présente un défaut lorsque les classes sont déséquilibrées, c'est-à-dire lorsqu'une classe possède beaucoup plus d'exemples d'entraînement qu'une autre. En effet, imaginons qu'une classe représente à elle seule 90% des exemples contenus dans une base d'entraînement, alors en prédisant cette classe de façon constante, sans même regarder l'entrée, on obtient 90% de rappel global.

Par conséquent, une seconde mesure de performance, appelée *rappel moyen par classe*, consiste à pondérer les exemples d'entraînement de manière inversement proportionnelle au nombre d'échantillons par classe, ce qui revient à donner la même importance à chaque classe.

Def. 4.11 *Mean class accuracy*

Soit k le nombre de classes considérées et n_{ij} le nombre de pixels de la classe i prédits comme appartenant à la classe j. Le rappel moyen par classe (ou mean class accuracy en anglais) est défini par :

$$MCA = \frac{1}{k}\sum_{i=1}^{k}\frac{n_{ii}}{\sum_{j=1}^{k} n_{ij}} \tag{4.9}$$

Un défaut du rappel moyen par classe est que ce dernier a tendance à privilégier une classe faiblement représentée au dépit de sa précision. En effet, dû à la pondération des échantillons, un faux négatif sur un exemple très représenté sera moins pénalisant (en termes de performance) qu'un faux négatif sur un exemple peu représenté. Par conséquent la mesure de performance la plus utilisée aujourd'hui en segmentation sémantique est l'intersection over union (IoU) aussi connu sous le nom d'index de Jaccard, qui cherche à concilier pixel accuracy et rappel moyen par classe.

Def. 4.12 *Intersection over Union*

Soit k le nombre de classes considérées et n_{ij} le nombre de pixels de la classe i prédits comme appartenant à la classe j. L'intersection over union est exprimée par :

$$IoU = \frac{1}{k}\sum_{i=1}^{k}\frac{n_{ii}}{\sum_{j=1}^{k} n_{ij} + \sum_{j=1}^{k} n_{ji} - n_{ii}} \tag{4.10}$$

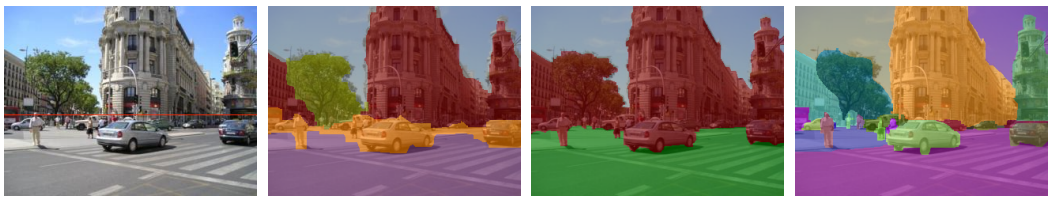
4.1.3 Bases de données pour la segmentation sémantique

Afin d'entrainer et d'évaluer les différents algorithmes il existe plusieurs bases de données conçues pour la segmentation sémantique. Ces bases de données contiennent des images prises sous différentes conditions ainsi que des images de vérités terrains, c'est-à-dire des images où chacun des pixels correspond à l'indice d'une classe sémantique. Cette section détaille une liste non exhaustive des bases les plus populaires actuellement.

Stanford background [Gou+09]

La base de données *Stanford background*[1] [Gou+09] est une des plus anciennes bases fournissant des vérités terrains pour la segmentation sémantique. Elle est composée d'images importées d'autres base de données. Elle contient 715 images d'une résolution de 320×240 pixels ainsi que leurs vérités terrains avec quatre types d'annotations :

1. http ://dags.stanford.edu/projects/scenedataset.html



(a) Horizons (b) Régions (c) Surfaces (d) Couches

Fig. 4.2.: Exemple des différents types d'annotations fournies avec la base de données Stanford Background.

— Horizons (4.2a) : qui donne la position (horizontale) de l'horizon dans chacune des images.

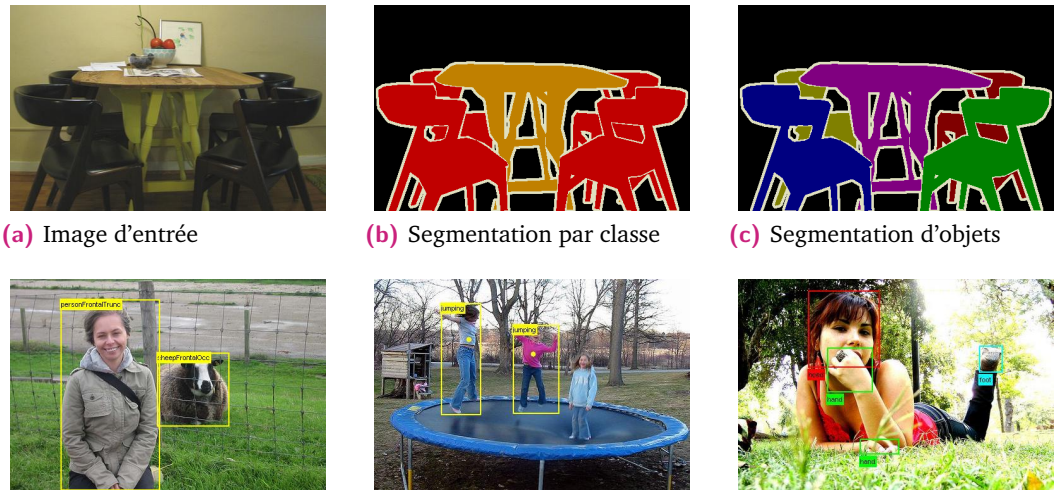
— Régions (4.2b) : qui précise la classe sémantique pour chacun des pixels des différentes images (parmi : *ciel*, *arbre*, *route*, *herbe*, *eau*, *bâtiment*, *montagne*, *objet en premier plan* ou *inconnu*)

— Surfaces (4.2c) : qui détermine le type de surface (horizontale, verticale ou ciel) de chacun des pixels des différentes images.

— Couches (4.2d) : qui différencie les différents objets composant une scène en leur donnant un indice unique.

Les auteurs se sont assurés que chaque image possède au moins un objet au premier plan ainsi que l'horizon visible.

Pascal Visual Object Classes (Pascal VOC) [Eve+10]



(a) Image d'entrée (b) Segmentation par classe (c) Segmentation d'objets

(d) Classification et Détection (e) Classification d'actions (f) Détourage de personnes

Fig. 4.3.: Exemple des différentes tâches réalisables avec la base de données Pascal Voc 2012.

La base de données *Pascal VOC* est probablement la plus populaire dans la communauté de vision par ordinateur. Créée en 2005, la première version de la base n'a

cessé d'évoluer jusqu'en 2012. Aujourd'hui, la base possède des images associées à des vérités terrains permettant cinq tâches différentes :

— La *classification* et *détection* d'objets (figure 4.3d) qui consiste à prédire la présence ou l'absence de 20 types d'objets différents ainsi qu'à les localiser dans une image (en fournissant leurs boites englobantes).

— La *segmentation par classe* (figure 4.3b) qui consiste à annoter chacun des pixels d'une image en les classants dans une des 20 différentes classes.

— La *Segmentation d'objet* (figure 4.3c) qui consiste à délimiter les différents objets présents dans l'image.

— Le *détourage de personne* (figure 4.3f) consiste à prédire la position de différentes parties du corps humain. Pour cette tâche trois parties sont à détecter : la tête, les pieds et les mains.

— La *classification d'action* (figure 4.3e) qui consiste à déterminer l'action réalisée par les personnes présentes dans une image. Pour cette tâche 10 types d'actions différentes plus une catégorie "*Autre*" sont utilisés.

Pour la tâche de segmentation, la base de données contient 1464 images d'entrainement et 1449 images de validation. L'ensemble de test n'est pas fourni, mais un site web [2] permet une évaluation en ligne des méthodes.

SiftFlow [Liu+11]

SiftFlow [3] est une base de données contenant des images d'une résolution de 256×256. La base contient 2.688 images annotées pour la segmentation sémantique en utilisant l'outil d'annotation en ligne LabelMe qui permet d'annoter des images et de créer une base de données de façon collaborative (chacun ajoute ses propres images et annotations augmentant ainsi le nombre d'images total contenues dans la base de données). L'ensemble des images a été aléatoirement séparé en 2.466 images d'entrainement et 200 images de tests. Les annotations contiennent 33 classes sémantiques incluant, entre autres, les classes "*Balcon*", "*Bateau*", "*Herbe*", "*Vache*" etc.

KITTI [Gei+13]

La base de données *KITTI* [4] a été développé en 2012. Les données ont été enregistrées à l'aide d'une voiture, équipée de plusieurs capteurs, roulant dans les rues de

2. http ://host.robots.ox.ac.uk/pascal/VOC/
3. http ://people.csail.mit.edu/celiu/LabelTransfer/
4. http ://www.cvlibs.net/datasets/kitti/

Karlsruhe en Allemagne. Les capteurs incluent des caméras stéréo, un laser Velodine permettant d'obtenir des informations de profondeur, un capteur GPS etc.

Malgré la popularité de la base de données, les vérités terrains pour la tâche de segmentation sémantique ne sont pas fournies. Néanmoins, plusieurs équipes de chercheurs ont créé leurs propres vérités terrains à partir de sous-ensembles de la base de données.

A l'époque de nos travaux, sept équipes avaient créé des vérités terrains de sept sous-parties de la base constituant au total 736 images annotées avec différents ensembles d'étiquettes (voir chapitre 5.4.2).

Microsoft Common Objects in Context (MS Coco) [Lin+14]

(a) Exemple d'entraînement pour la tâche de détection

(b) Exemple d'entraînement pour la tâche d'annotation

(c) Exemple d'entraînement pour la tâche de points clés

(d) Exemple d'entraînement pour la tâche des choses

Fig. 4.4.: Exemple d'entraînement pour les différentes tâches de détection de la base de données MS-Coco.

Microsoft Common Objects in Context [5], abrégée *MS-Coco* est une base de données de plus en plus populaire grâce la précision des vérités terrains et au nombre important d'images qu'elle contient. Les annotations ainsi qu'une évaluation en ligne des résultats offre une compétition évoluant chaque année parmi quatre tâches différentes :

— La *segmentation d'objets* (figure 4.4a), qui demande de segmenter et/ou de distinguer les différentes instances d'objets présents sur une image.

— La détection de *points clés* (figure 4.4c), où la tâche consiste à distinguer les différentes parties des corps humains dans une image.

— La *segmentation de choses* (figure 4.4d), qui ressemble à la segmentation d'objets, mais où la sémantique des classes s'intéresse plutôt aux "choses" tel

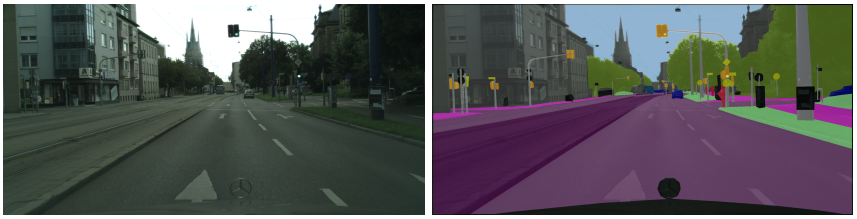
5. http ://cocodataset.org/

que "*Herbe*", "*Mur*", "*Ciel*", plutôt qu'aux objets tels que "*Personne*", "*Voiture*" ou "*Éléphant*".

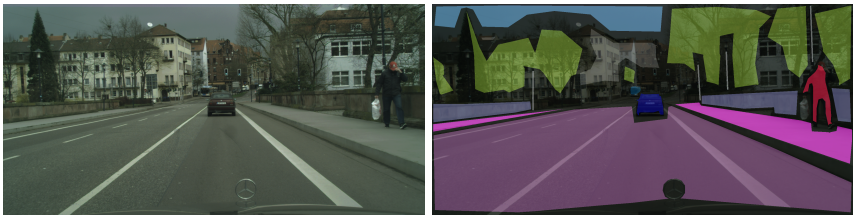
— L'*annotation*, qui consiste à donner une description textuelle des images. La compétition sur cette tâche s'est terminée en 2015 mais les données ainsi que l'évaluation en ligne des résultats restent disponibles.

Pour la segmentation d'objets, la base de données contient plus de 160.000 images dans lesquels apparaissent plus de 80 catégories d'objets différents. L'ensemble d'entrainement est composé de 82.783 images, l'ensemble de validation et de test contiennent respectivement 40.504 et 40.775 images.

Cityscapes [Cor+16]



(a) Exemple d'entraînement avec annotations denses



(b) Exemple d'entraînement avec annotation grossières

Fig. 4.5.: Exemple d'entraînement provenant de la base de données Cityscapes.

La base de données *Cityscapes* [6] contient des images haute résolution (1024 pixels de haut et 2048 pixels de large) prises à l'aide d'une caméra HD montée sur un véhicule circulant dans les rues de 50 villes d'Allemagne et des alentours. D'autres capteurs, tel qu'un GPS, sont aussi utilisés pour avoir des informations sur le déplacement du véhicule.

Cette base de données est spécialement conçue pour la segmentation sémantique. Ainsi elle possède de nombreuses images allant de pair avec des vérités terrains d'une grande précision. La base est séparée en trois ensemble d'entrainement, validation et de test contenant chacun 2975 images, 500 images et 1500 images (pour un total d'environ 5000 images). En plus des images finement annotées (voir figure 4.5a), ils proposent aussi 20.000 images annotées grossièrement, c'est-à-dire avec des délimitations grossières des différentes parties des images (voir figure 4.5b).

6. https ://www.cityscapes-dataset.com/

30 classes sémantiques groupées dans 8 catégories différentes ont été utilisées pour annoter les images. Néanmoins, à cause d'un faible nombre d'exemples présent dans l'ensemble d'entrainement, certaines classes ne sont pas prises en compte pour l'évaluation des performances. Ainsi, sur les 30 classes sémantiques, seulement 19 sont utilisées pour l'évaluation des algorithmes.

Les vérités terrains permettent aussi de faire la distinction entre les différentes instances des éléments d'une scène ce qui permet d'utiliser cette base de données aussi pour la tâche de segmentation d'instance.

Autres bases de données

Dû à la popularité des réseaux de neurones et aux performances croissantes des algorithmes en termes de reconnaissance d'image, de nombreuses autres bases de données ont récemment vu le jour. Parmi elles, on peut noter *Synthia* [Ros+16], une base de données entièrement virtuelle, *CamVid* [Bro+08] une base de données fournissant des courtes séquences vidéo, *Youtube-Objects* [Pre+12] qui ne contient pas d'annotation par pixels, mais dont l'équipe de Jain *et al.* [JG14] ont annoté un sous-ensemble de 126 séquences ou encore *ADE20K* [Zho+17], une base de données récente, mais de plus en plus populaire de par la diversité des classes sémantiques utilisées et du nombre d'images annotées qu'elle contient.

Enfin, il existe aussi des bases de données telles que *Stanford 2D-3D-S* [Arm+17] ou *Sydney Urban Objects Dataset* [Qua14] qui fournissent aussi des informations tridimensionnelles afin d'améliorer la segmentation des images.

4.2 Limitation des réseaux convolutifs classiques

Nous avons déjà présenté dans le chapitre 2 de nombreux réseaux de neurones convolutifs célèbres, parmi lesquels on peut citer AlexNet [Kri+12], GoogleNet [Sze+15a], ResNet [He+16b] ou encore le récent DenseNet [Hua+16b]. Leur célébrité est due principalement à leurs performances sur la compétition *ImageNet Large Scale Visual Recognition Competition* (ILSVRC) et à la possibilité de les spécialiser sur une nouvelle tâche. En effet, il a été montré qu'un réseau de neurones convolutif apprend, sur les premières couches de convolutions, des caractéristiques générales [Yos+14] et que ces dernières sont utiles lorsque l'on change la tâche (ou les classes cibles) du réseau. Ainsi, comme nous l'avons vu dans la partie 2.3.2, il est courant d'utiliser un réseau entraîné sur la base de données ImageNet pour transférer les connaissances apprises sur une autre tâche. Si ce transfert de connaissances permet d'obtenir de

meilleurs résultats, il présente le défaut de contraindre l'architecture du réseau. Or les réseaux présentés précédemment ont été entrainés sur une tâche de classification d'image qui, comme nous l'avons déjà vu, consiste à fournir une unique prédiction par image d'entrée. Il est donc nécessaire d'adapter ces réseaux afin qu'ils fournissent une prédiction dense, c'est-à-dire, une prédiction par pixels de l'image d'entrée.

Dans cette section nous commencerons par étudier les réseaux classiques afin de comprendre pourquoi ils ne donnent qu'une unique prédiction. Pour cela nous verrons les notions de champs réceptif et de taille des cartes de caractéristiques. Ensuite, nous verrons comment utiliser les réseaux classiques naïvement puis plus efficacement sur une tâche de segmentation sémantique, puis finalement nous verrons comment modifier l'architecture des réseaux afin de les spécialiser à la segmentation sémantique d'images.

4.2.1 Champ réceptif et taille des cartes de caractéristiques

Un réseau de neurones convolutif est composé de plusieurs opérateurs dont les opérateurs de convolutions et de sous-échantillonnage (généralement de type *maximum pooling*) vus dans la partie 2.1.4. Dans cette partie nous nous intéressons à ces derniers et nous verrons comment leurs utilisations ont un impact sur la taille des cartes de caractéristiques et sur ce que l'on appelle le champ réceptif d'un réseau de neurones.

Champ réceptif

Def. 4.13 *Le champ réceptif (ou champ de vision) d'un réseau de neurones convolutif est la partie maximale de l'image visible par le réseau.*

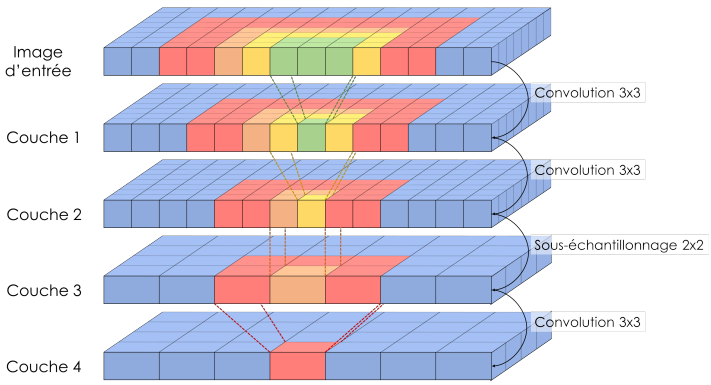


Fig. 4.6.: Illustration du champ réceptif d'un réseau de convolution. Au fil des opérations de convolutions, la zone visible de l'image d'entrée est de plus en plus large.

Si l'on applique une convolution de taille 3×3 à une image, il est facile de comprendre que chaque valeur obtenue sur la carte de caractéristiques en sortie dépend des 3×3 pixels de l'image d'entrée (voir schémas 4.6). Ainsi, le champ réceptif des caractéristiques après une première couche de convolution de masque 3×3 est de 3×3. De la même manière, une deuxième couche de convolution (aussi de

taille 3×3 pour notre exemple) dépendra des 3×3 caractéristiques extraites par la première couche de convolution, qui elles-mêmes dépendent, au total, de 5×5 pixels de l'image d'entrée (voir figure 4.6). Le champ réceptif des caractéristiques présentes à la deuxième couche d'un réseau de neurones convolutifs est donc de 5×5. Ainsi, ajouter une couche de convolution 3×3 augmente le champ réceptif de manière additive de 2×2. De manière générale, s'il n'y a pas eu d'opérations de sous-échantillonnage, ajouter une couche de convolution de taille de masque $S \times S$, augmente le champ réceptif de manière additive de $(S-1) \times (S-1)$.

L'intérêt des opérations de sous-échantillonnage est double : il permet de réduire la taille des cartes de caractéristiques et ainsi de limiter l'espace mémoire utilisé, mais aussi et surtout, il permet, de façon combinée aux opérations de convolutions, d'augmenter de manière significative le champ réceptif. Après l'application d'une opération de sous-échantillonnage de masque de taille 2×2, une valeur dans la carte de caractéristiques de sortie dépendra des 2×2 valeurs de la carte d'entrée. Cela n'augmente pas de manière significative le champ réceptif. Dans notre exemple illustré dans la figure 4.6 on passe d'un champ de vision de 5×5 à la couche 2, à 6×6 à la couche 3. Par contre, la résolution des cartes de caractéristiques étant diminuée de 2×2, l'application d'une couche de convolution de masque de taille 3×3 correspond à un champ réceptif de 6×6 par rapport aux cartes de caractéristiques avant l'opération de sous-échantillonnage, ce qui donne au total un champ réceptif de 10×10 pixels sur l'image d'entrée (voir figure 4.6).

Ainsi, l'enchainement d'autres opérations de convolutions et de sous-échantillonnage augmente de manière significative le champ réceptif du réseau. L'intérêt principal de cette augmentation du champ réceptif est lié à l'objectif général des méthodes issues de l'apprentissage profond, c'est-à-dire l'apprentissage automatique d'une représentation hiérarchique. Pour être plus précis, nous souhaitons obtenir une représentation où les niveaux (les couches) correspondent à des niveaux d'abstraction croissant. Pour cela, il est important que les filtres des niveaux élevés s'appliquent à des régions croissantes. Augmenter la taille des filtres dans les couches élevées augmenterait inutilement les capacités des modèles. Le sous-échantillonnage est une solution naturelle pour pallier à ce problème.

4.3 Les réseaux conçus pour la segmentation sémantique

Les opérations de sous-échantillonnage sont très utiles, car elles réduisent la résolution des cartes de caractéristiques, ce qui a pour effet, en plus de réduire l'espace mémoire utilisé, de permettre aux opérateurs de convolutions d'avoir un champ

réceptif de plus en plus grand (voir partie précédente 4.2.1). Néanmoins, dans le cadre de la segmentation sémantique d'images, la réduction de la taille des cartes de caractéristiques est problématique puisque cela réduit aussi la taille des prédictions alors que l'objectif est d'obtenir une prédiction pour chacun des pixels d'entrée.

Dans cette partie nous verrons comment adapter les architectures des réseaux de neurones convolutifs afin d'obtenir des prédictions dites "*denses*", c'est-à-dire une prédiction par pixel de l'image d'entrée.

4.3.1 Prédiction dense par décalage d'entrées

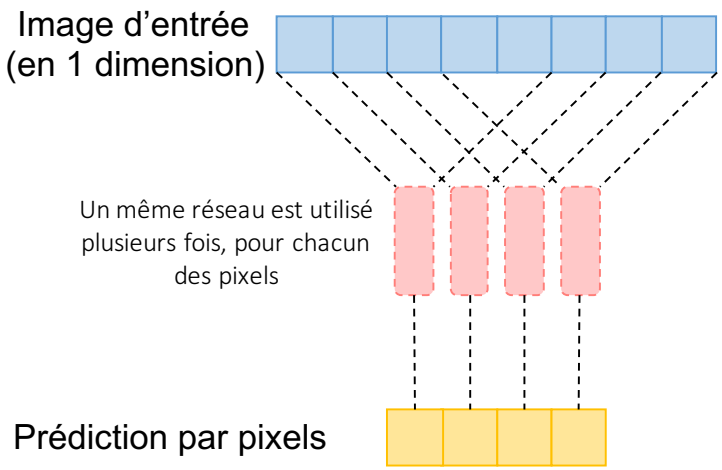


Fig. 4.7.: Méthode naïve pour obtenir une prédiction pour chacun des pixels d'une image d'entrée. Pour des soucis de clarté, l'image (en bleu) n'est représentée en une dimension. Le réseau utilisé, en rouge est le même pour chaque sous-partie de l'image et prend 5 pixels en entrée. Dans cet exemple les pixels au bord de l'image ne sont pas représentés, mais ils peuvent être annotés en ajoutant des pixels avec une valeur de 0 au bord de l'image.

Lorsque l'on a un réseau capable, étant donnée une image d'entrée, de fournir une unique prédiction, la méthode naïve pour annoter l'ensemble des pixels d'une image est d'entraîner ce réseau à annoter le pixel au centre d'une fenêtre d'observation. Puis, pour annoter complètement l'image, on découpe une fenêtre d'observation pour chaque pixel de l'image qui sert d'entrée au réseau. Cette technique (illustrée dans la figure 4.7), bien que facile à mettre en place, est extrêmement couteuse en temps de calculs puisqu'elle nécessite une évaluation complète de chaque fenêtre d'observation par le réseau, créant ainsi beaucoup de redondance dans les calculs.

Sermanet *et al.* [Ser+13] ont créé un réseau appelé *Overfeat* dans lequel ils utilisent une technique permettant d'obtenir une sortie plus dense tout en évitant des calculs redondant. Pour cela, ils modifient la dernière couche de sous-échantillonnage (se

situant à la cinquième couche dans leurs réseau) avant les couches entièrement connectées. Ils opèrent ensuite en cinq étapes :

— (a) Ils récupèrent les cartes de caractéristiques avant l'opérateur de sous-échantillonnage (d'une fenêtre 3×3 dans leurs réseau).

— (b) Les cartes de caractéristiques sont évaluées par l'opérateur de sous-échantillonnage et l'opération est répétée 3×3 fois avec des cartes dont les pixels sont décalés d'un pas (Δ_x, Δ_y) avec $\Delta \in \{0, 1, 2\}$.

— (c) Cela produit un ensemble de cartes de caractéristiques sous-échantillonnées, correspondant aux cartes de caractéristiques (Δ_x, Δ_y) d'entrée.

— (d) Les dernières couches du réseau (6,7,8) évaluent les multiples sorties obtenues.

— (e) Les prédictions finales obtenues pour les différentes cartes de caractéristiques (Δ_x, Δ_y) sont combinées en une unique sortie.

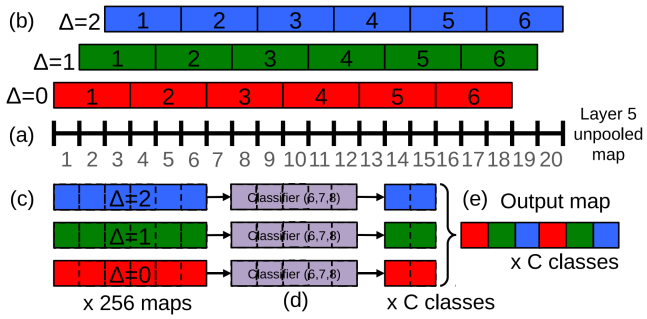


Fig. 4.8.: Illustration en 1 dimension. L'image représentée par le tableau (a) est décalée trois fois (b) pour être passée dans trois opérateurs de sous-échantillonnages (b). Les résultats obtenus (c) sont donnés au classificateur (d). Les résultats finaux sont ensuite ordonnés (e) permettant d'obtenir une sortie plus dense. L'image est tirée du papier d'origine [Ser+13]

Ces opérations, illustrées dans la figure 4.8, permettent d'obtenir une sortie dont la densité des prédictions est équivalente à un réseau sans la dernière couche de sous-échantillonnage. Cela veut dire que, bien que la prédiction du réseau soit plus dense, elle n'a toujours pas la résolution de l'image d'entrée. Il est possible de répéter cette opération autant de fois que d'opérateurs de sous-échantillonnage présents dans le réseau, mais cela est extrêmement compliqué et demande beaucoup d'espace mémoire. Cette technique a donc trouvé son intérêt dans des applications telles que la segmentation pour l'estimation de pose de la main (Neverova *et al.* [Nev+15b]) où les images sont de petites tailles, mais n'est pas applicable à la tâche de segmentation sémantique où les images d'entrée atteignent de plus grandes dimensions.

4.3.2 Réseaux entièrement convolutifs

Plutôt que d'utiliser un réseau conçu pour la classification d'image et de l'adapter à la segmentation d'image, Long *et al.* [Lon+15] montrent comme construire des réseaux de neurones convolutifs spécialisés à la segmentation.

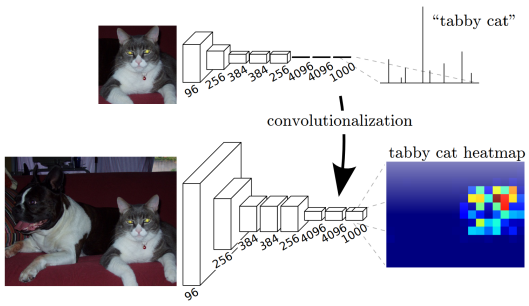


Fig. 4.9.: Les parties entièrement connectées d'un réseau peuvent être transformées en couche convolutives permettant d'utiliser des sorties à différentes tailles. Image tirée du papier [Lon+15].

Pour cela, ils ont conçu un réseau dit entièrement convolutif. Ils ont montré qu'une couche entièrement connectée est équivalente à une couche convolutive dont le masque de convolution est de taille 1×1.

En effet, nous avons vu qu'une carte de caractéristiques est obtenue en faisant la combinaison des convolutions des N cartes de caractéristiques d'entrée f par les masque g de taille $S \times S$ et de demi taille $K \times K$ avec $K = \frac{S-1}{2}$ (voir le détail dans la partie 2.1.3) :

$$(f * g)_j(n,m) = b_j + \sum_{i=1}^{N} \sum_{n'=-K}^{K} \sum_{m'=-K}^{K} f_i(n+n', m+m') \times g_{ij}(n', m')$$

Or si les masque de convolution ont une taille de 1×1 alors $K = \frac{1-1}{2} = 0$ ce qui donne :

$$(f * g)_j(n,m) = b_j + \sum_{i=1}^{N} f_i(n,m) \times g_i(0,0)$$

On retrouve donc la formule d'un perceptron (vue dans la partie 2.1.1) à savoir :

$$y = b + \sum_{i=0}^{N} w_i x_i$$

Avec $w_i = g_i(0,0)$.

L'avantage d'utiliser une couche de convolutions à la place d'une couche entièrement connectée est que les dimensions des images d'entrée ne sont plus contraintes par le réseau (voir figure 4.9). En effet, une couche entièrement connectée prend en entrée

un vecteur de caractéristiques de taille fixe alors qu'une couche de convolution, de par la fenêtre glissante des masques de convolutions, prend en entrée des cartes de caractéristiques de tailles quelconques (mais tout de même plus grande que la taille des masques de convolutions) et produit des sorties dont les tailles dépendent de celles des entrées. Ainsi, un réseau entièrement convolutif prend des tailles d'images quelconque en entrée et produit des sorties dont la taille varie en fonction des dimensions d'entrée.

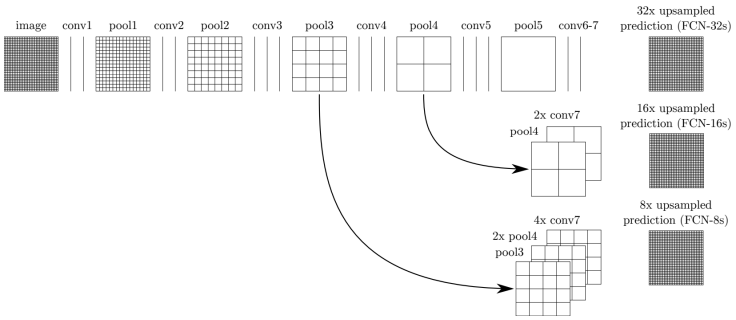


Fig. 4.10.: Des convolutions sont utilisées pour reconstruire les cartes de caractéristiques et augmenter la taille de la prédiction finale. Afin de permettre une meilleure reconstruction, les cartes de caractéristiques extraites dans les parties convolutives sont aussi utilisées. Image tirée du papier [Lon+15].

Néanmoins, à cause des opérateurs de sous-échantillonnage, les dimensions en sortie ne sont toujours pas les mêmes que les dimensions d'entrée. Afin d'augmenter les tailles des sorties du réseau, ils rajoutent, après le réseau convolutif, une partie dite "*déconvolutive*" dont l'objectif est d'augmenter la taille des cartes de caractéristiques. Pour cela, ils utilisent des opérations de sur-échantillonnages (par interpolation des cartes de caractéristiques ou avec des convolutions dont le pas est inférieur à 1) suivies d'opérateurs de convolutions. De plus, pour aider les convolutions à reconstruire les cartes de caractéristiques, ils ajoutent aussi l'information extraite dans la partie convolutive (voir schémas 4.10). Néanmoins, ces opérations de reconstructions sont compliquées et l'apprentissage du réseau est laborieux puisqu'il nécessite plusieurs étapes d'apprentissage (une par opération de reconstruction). C'est pourquoi ils ajoutent les couches une à une lors de l'entraînement et ne parviennent pas à reconstruire complètement les cartes de caractéristiques. Leur prédiction reste à une résolution d'un huitième de l'image d'entrée d'origine.

4.3.3 Réseau conv-déconv

Noh *et al.* [Noh+15] ont repris les travaux de Long *et al.* et améliorent l'apprentissage de la partie déconvolutive. Dans leurs travaux, ils présentent un réseau composé de deux parties, une partie *convolutive* et une partie *déconvolutive* symétrique (voir figure 4.11). La partie convolutive est un réseau de neurones classique composé, entre autres, d'opérateurs de convolution et de sous-échantillonnage. Dans leurs

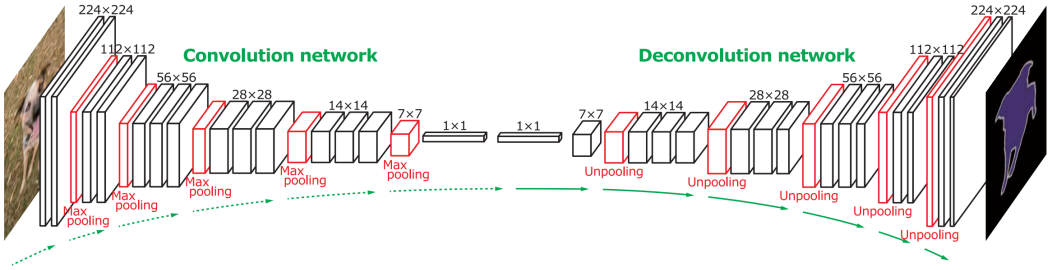
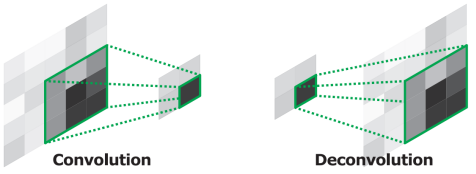
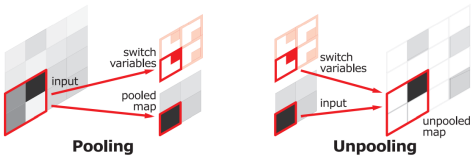


Fig. 4.11.: Réseau Conv-Deconv utilisé par Noh *et al*. Le réseau est composé de deux parties : une partie convolutive (à gauche) et une partie déconvolutive symétrique (à droite). Dans leurs expériences, ils utilisent un réseau de type VGG16 pour la partie convolutive. Image tirée du papier [Noh+15]

travaux, Noh *et al*. ont utilisé un réseau de type VGG16 (voir partie 2.3.3) mais n'importe quelle architecture peut être utilisée. La deuxième partie, *déconvolutive* est symétrique à la première. Elle permet de reconstruire les cartes de caractéristiques grâce à des opérateurs de déconvolutions et de sur-échantillonnage (voir figure 4.12).



(a) Illustration des opérateurs de convolution et déconvolution.



(b) Illustration des opérateurs de maximum pooling et maximum unpooling.

Fig. 4.12.: Opérateurs de convolution et de maximum pooling ainsi que leurs équivalents permettant la reconstruction des cartes de caractéristiques. Image tirée du papier [Noh+15].

L'opérateur de déconvolution (voir figure 4.12a) correspond à l'opération inverse d'un opérateur de convolution. Quand l'opérateur de convolution prend une fenêtre d'entrée afin de calculer une valeur de sortie, l'opérateur de déconvolution prend en entrée une valeur et permet de calculer plusieurs sorties. Cet opérateur peut être vu comme une généralisation d'une augmentation de résolution par interpolation linéaire, où l'interpolation linéaire est remplacée par une interpolation pondérée par des poids appris lors de l'entraînement.

L'opérateur de sur-échantillonnage est très intéressant. Dans leur réseau, Noh *et al*. utilisent des opérateurs appelés "*maximum unpooling*" liés aux opérateurs de sous-échantillonnages de type *maximum pooling* présents dans la partie convolutive. Lors de l'évaluation d'une image, les opérateurs de maximum pooling de la partie convolutive conservent les positions des valeurs maximales observées. Ces positions, liées aux images d'entrée et non pas aux paramètres entraînables du réseau, sont ensuite utilisées par les opérateurs de *maximum unpooling*, présents dans la partie déconvolutive, afin de reconstruire les cartes de caractéristiques (voir figure 4.12b).

Cette technique permet de transmettre une information entre la partie *convolution* et *déconvolutive* permettant à Noh *et al.* de reconstruire entièrement les cartes de caractéristiques et ainsi d'obtenir des sorties à la même résolution que les images d'entrées. De plus, leur réseau en deux parties, utilisant une partie convolutive classique, présente l'avantage de permettre l'utilisation d'un réseau pré-entraîné, ce qui permet une meilleure performance de classification.

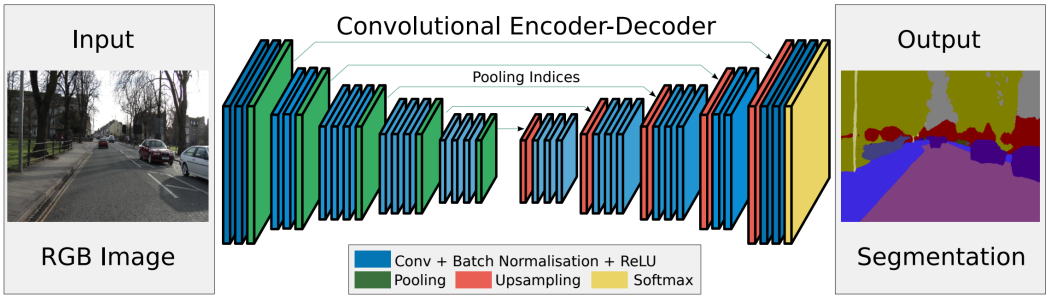


Fig. 4.13.: Schémas du réseau SegNet [Bad+15] développé par Badrinarayanan *et al.* L'architecture de ce réseau est très proche de celle de Noh *et al.* illustrée dans la figure 4.11 mais ne possède pas de partie entièrement connectée. Image tirée du papier [Bad+15].

De manière parallèle au développement du réseau conv-deconv de Noh *et al.*, Badrinarayanan *et al.* ont développé un réseau appelé SegNet [Bad+15] composé lui aussi d'une partie convolutive et une partie déconvolutive symétrique. La principale différence par rapport au réseau de Noh *et al.* est que SegNet ne possède pas de partie entièrement connectée (voir figure 4.11). Cela permet de réduire considérablement le nombre de paramètres du réseau permettant un entraînement plus facile et plus rapide.

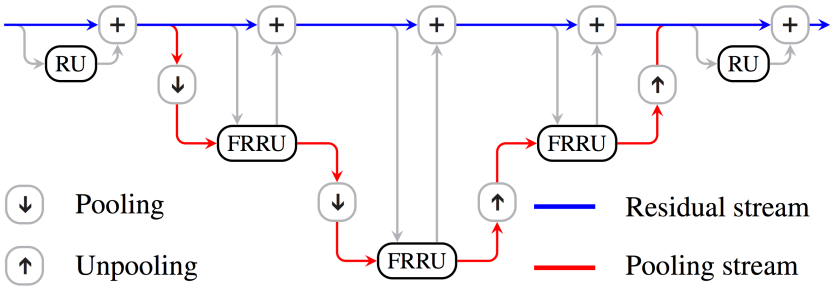


Fig. 4.14.: Schémas du réseau Full Resolution Residual Network (FRRN) de Pohlen *et al.* [Poh+16]. Le réseau est composé de deux réseaux interconnectés, un réseau conv-deconv (en rouge) et un réseau résiduel (en bleu). Des opérateurs de redimensionnement sont utilisés pour connecter les deux (en gris). La figure est tirée du papier d'origine [Poh+16].

Finalement, un problème, lié à l'ajout d'une partie déconvolutive à la suite d'un réseau convolutif traditionnel, est que la profondeur des réseaux augmente forcément. Par conséquent ces derniers retrouvent des difficultés d'entraînement (dû au problème de perte du gradient vue dans la partie 2.3.4). Une technique, permettant à la fois une bonne circulation du gradient et une reconstruction fine des cartes

de caractéristiques, a été développée par Pohlen *et al*. Avec leur Full Resolution Residual Network (FRRN) [Poh+16], ils combinent un réseau conv-déconv avec un réseau résiduel. Leur réseau est composé de deux chemins (un résiduel, un conv-deconv) partageant les mêmes cartes de caractéristiques (voir figure 4.14). Ainsi le réseau conv-deconv extrait des caractéristiques avec une forte sémantique et transmet les informations au réseau résiduel qui conserve une information haute résolution (et permet donc une reconstruction fine des cartes de caractéristiques) tout en permettant une bonne circulation du gradient (qui remonte le long du réseau résiduel jusqu'aux couches du réseau conv-deconv).

4.3.4 U-Network

Dans le réseau conv-déconv de Noh *et al*. une information est transmise entre les parties convolutive et déconvolutive sous la forme des indices des opérateurs de maximum pooling. Néanmoins, cette information est faible et la majorité de l'information permettant la reconstruction des cartes de caractéristiques doit donc passer par le goulot d'étranglement du réseau (c'est-à-dire la partie où les cartes de caractéristiques sont les plus petites, mais aussi les plus nombreuses). Cela pose une difficulté car, en plus d'extraire des informations sémantiques, la partie convolutive se retrouve à devoir aussi extraire des informations spatiales, permettant la reconstruction des cartes de caractéristiques.

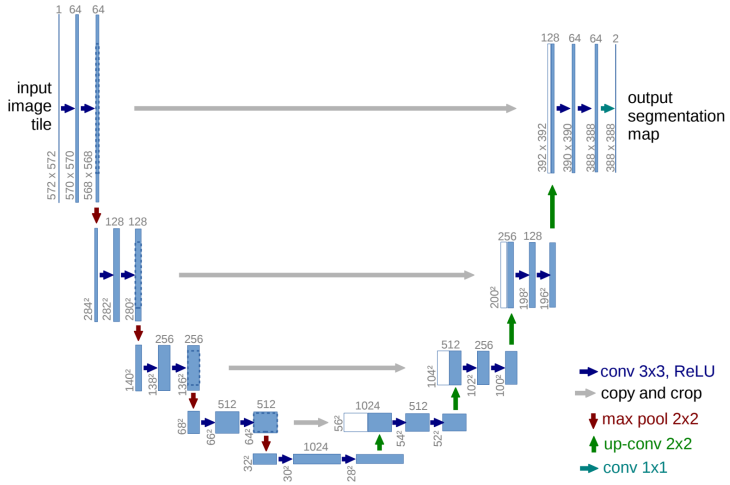


Fig. 4.15.: U-Network. Les cartes de caractéristiques extraites avant les opérateurs de sous-échantillonnage de la partie convolutive sont concaténées aux cartes de caractéristiques extraites après les opérateurs de sur-échantillonnage de la partie déconvolutive. Image tirée du papier [Ron+15].

Afin de permettre une meilleure circulation de l'information spatiale et donc une meilleure reconstruction, Ronneberger *et al*. [Ron+15] ont créé une architecture appelée *U-Network* (U-Net). De manière identique aux réseaux conv-deconv, les U-Networks ont une partie déconvolutive symétrique à la partie convolutive ce qui

permet d'avoir des cartes de caractéristiques dont les tailles sont compatibles (= identiques) entre les parties deux parties du réseau (voir schémas 4.15). Ainsi les cartes de caractéristiques extraites dans la partie convolutive peuvent être concaténées à celle reconstruites dans la partie déconvolutive transférant ainsi une information spatiale plus importante et permettant une meilleure reconstruction. Dans des travaux récents, Newell *et al.* [New+16] ont montré qu'il est intéressant d'utiliser plusieurs U-Net en développant un réseau appelé "Stacked Hourglass Networks". Cette succession de partie convolutive et déconvolutive, utilisée conjointement avec une supervision intermédiaire, permet une meilleure collaboration entre les différentes cartes de caractéristiques et améliore la capacité de reconnaissance et localisation du réseau. Intuitivement on peut imaginer un processus répétitif d'essais et de corrections, avec un premier U-Net qui propose une prédiction et qui est corrigée par le second et ainsi de suite.

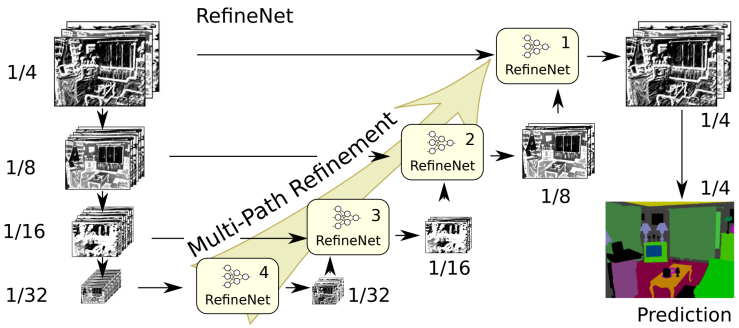


Fig. 4.16.: Schéma du réseau RefineNet [Lin+16b]. Le réseau suit le schéma d'un U-Net mais utilise un opérateur appelé refinenet afin de transformer les cartes de caractéristiques circulant de la partie convolutive à la partie déconvolutive. Image tirée du papier [Lin+16b].

Dans un réseau conv-deconv, la partie convolutive extrait des informations sémantiques sur les différents objets composant une scène quand la partie déconvolutive reconstruit les cartes de caractéristiques avec les informations extraites. Par conséquent, les cartes de caractéristiques n'ont donc pas les mêmes objectifs et donc pas les mêmes sémantiques, ce qui pose un problème dans les U-Net qui concatènent directement les cartes de caractéristiques de la partie convolutive à la partie déconvolutive. Pour régler ce problème, Lin *et al.* ont développé un réseau appelé RefineNet [Lin+16b], qui utilise la même idée que pour les U-Net, mais qui, au lieu de concaténer directement les cartes de caractéristiques, utilise une unité appelée *refineNet*, contenant des unités convolutives résiduelles, des opérateurs de fusion multi-résolutions et un sous-échantillonnage résiduel (voir schémas 4.16). Leur module permet ainsi une transformation des cartes de caractéristiques extraites par la partie convolutive, assurant une meilleure compatibilité avec celles de la partie déconvolutive.

4.3.5 Convolutions dilatées

Les architectures vues dans les parties précédentes utilisent une partie *déconvolutive* afin de reconstruire les cartes de caractéristiques, dont les tailles ont été réduites par les opérateurs de sous-échantillonnage. Une approche différente, étudiée par Yu et Koltun [YK16], consiste à retirer ces opérateurs sous-échantillonnage afin de conserver des cartes de caractéristiques aux dimensions de l'entrée, tout le long du réseau. Néanmoins, comme nous l'avons vu dans la partie 4.2.1, le sous-échantillonnage est nécessaire pour avoir un champ réceptif important. Afin d'augmenter le champ réceptif sans utiliser d'opérateurs de sous-échantillonnage, ils utilisent un opérateur convolutif particulier appelé "*convolution dilatée*" qui correspond à une généralisation des filtres convolutifs de Kronecker [GM16]. Avec une convolution classique, les poids du masque de convolution sont appliqués à des pixels voisins dans l'image d'entrée

$$(f * g)(n) = \sum_{n'=-K}^{K} f(n - n') \times g(n')$$

Une convolution dilatée utilise un facteur de dilatation l correspondant au voisinage des pixels de l'image d'entrée sur lequel les poids de la convolution s'appliqueront :

$$(f *_l g)(n) = \sum_{n'=-K}^{K} f(n - (l \times n')) \times g(n')$$

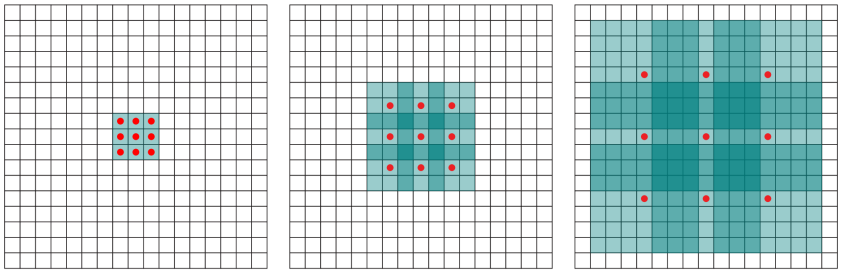


Fig. 4.17.: Champ réceptif des opérateurs de convolutions dilatées utilisant des facteurs de dilatations de 1,2 et 4. Image tirée du papier [YK16].

En commençant avec un facteur de dilatation de 1 et en multipliant ce facteur par 2 à chaque couche convolutive, il est possible d'obtenir un champ réceptif de plus en plus grand (voir figure 4.17).

L'utilisation de convolutions dilatées présente l'avantage d'étendre le champ réceptif des réseaux de neurones convolutifs sans augmenter le nombre de paramètres et surtout sans réduire la taille des cartes de caractéristiques. Il est donc naturel, suite aux travaux de Yu et Koltun [YK16], de retrouver des opérateurs de convolution dilatées dans les travaux de Chen *et al.* avec l'amélioration de leur réseau DeepLab [Che+16] ainsi que dans le réseau ENet [Pas+16] de Paszke *et al.*

Une des difficultés avec les convolutions dilatées, est que cette technique n'est utilisée que pour obtenir une segmentation dense. Par conséquent, il n'existe pas de réseaux traditionnels pré-entrainés sur la base de données ImageNet directement utilisable avec cette technique. Dans leurs travaux, Wu *et al.* [Wu+16] adaptent le populaire ResNet [He+16a], pré-entraîné sur la base de données ImageNet, afin de l'utiliser à la tâche de segmentation sémantique. Wu *et al.* gardent uniquement les premières couches du ResNet pré-entraîné et changent les opérations de convolutions classiques en convolutions dilatées. Pour des problèmes de mémoire, ils gardent aussi trois opérations de sous-échantillonnage (de taille 2×2), ce qui a pour effet de produire une prédiction à une résolution d'un huitième de la résolution de l'image d'entrée. Ils utilisent ensuite une interpolation linéaire afin de retrouver la résolution d'origine.

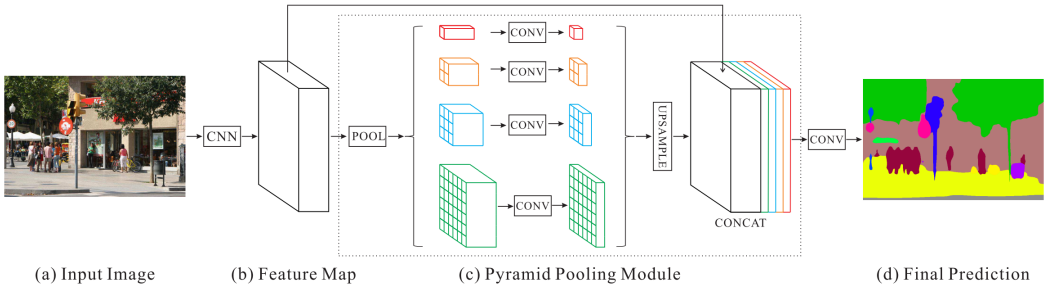


Fig. 4.18.: . Illustration du réseau PSPNet (Pyramid Scene Parsing Network) de Zhao *et al.* [Zha+16]. La sortie d'un ResNet spécialisé à la segmentation sémantique est sous-échantillonnée à plusieurs résolutions avant d'être traitée par plusieurs convolutions. Un opérateur reconstruit ensuite les cartes de caractéristiques leur permettant d'être concaténées et utilisées par une dernière convolution fournissant la prédiction. Cette image est tirée du papier [Zha+16].

Dans leurs travaux, Zhao *et al.* [Zha+16] remplacent l'interpolation linéaire par un module appelé Pyramique Pooling. Ce module est composé de plusieurs opérateurs de sous-échantillonnage de différents facteurs (réduisant plus ou moins les cartes de caractéristiques produites en sortie du réseau), chaque sous-échantillonnages est suivi d'opérations de convolutions puis d'opérateurs de sûr-échantillonnage permettant de retrouver la résolution d'entrée du réseau (voir figure 4.18). Toutes les cartes de caractéristiques obtenues par les différents opérateurs de sous-échantillonnage sont ensuite concaténées avant une dernière opération de convolution permettant d'obtenir la prédiction finale.

4.4 Prise en compte des cohérences spatiales

Lorsqu'un réseau de neurones convolutif fournit des annotations, ces dernières ne prennent pas en compte les cohérences sémantiques et spatiales entre les pixels. En effet, un réseau est entrainé à prédire des annotations par pixels et bien que les vérités terrains possèdent des cohérences spatiales, ces dernières ne sont pas explicitement

modélisées lors de l'apprentissage. Par conséquent, afin d'améliorer les prédictions obtenues à l'aide d'un réseau de neurones convolutif il est possible d'utiliser un post-traitement se présentant parfois sous la forme d'un CRF (*Conditional Random Field*).

4.4.1 Post traitement à l'aide de CRF

Les CRF ou champs aléatoires conditionnels, font partie de la famille des modèles graphiques probabilistes. Ils sont construits en associant un ensemble d'observations $\mathbf{I}$ à un ensemble de variables aléatoires $\mathbf{X}$. Dans le cas de la segmentation sémantique, l'ensemble d'observation $\mathbf{I}$ correspond à une image composée de pixels p_i. Chaque pixel p_i de l'image $\mathbf{I}$ est associé à une variable aléatoire $x_i \in \mathbf{X}$ pouvant prendre une valeur dans l'ensemble d'étiquettes $\mathcal{L} = \{l_1, l_2, ..., l_k\}$.

Dans leur définition des CRF, Lafferty *et al.* [Sut+07] utilisent les modèles graphiques probabilistes afin de modéliser les probabilités conditionnelles entre variables cachées (cibles) et variables observées (images). Ils définissent un graphe $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, qui modélise les interactions entres les variables aléatoires, où les sommets $\mathcal{V}$ (*vertices*) du graphe correspondent aux variables aléatoires $\mathbf{X} = (\mathbf{X}_v)_{v \in \mathcal{V}}$ et les arrêtes $\mathcal{E}$ (*edges*) représentent les interactions entres les variables. La paire $(\mathbf{I}, \mathbf{X})$, formée par les observations (l'image $\mathbf{I}$) et les variables aléatoires $\mathbf{X}$, modélise un CRF caractérisé par la distribution de Gibbs de la forme $P(\mathbf{X} = \mathbf{x}|\mathbf{I}) = \frac{1}{Z(\mathbf{I})} exp(-E(x|\mathbf{I}))$, où $E(\mathbf{x}|\mathbf{I})$ est appelée fonction d'énergie de la configuration $\mathbf{x} \in \mathcal{L}^N$ et $Z(\mathbf{I})$ est une fonction de partition. Autrement dit, $P(\mathbf{X} = \mathbf{x}|\mathbf{I})$ correspond à la probabilité que l'ensemble des variables aléatoires $\mathbf{X}$ prennent pour valeur la configuration $\mathbf{x}$ (c'est-à-dire $\mathbf{X}_i = \mathbf{x}_i \in \mathcal{L}$ pour $i \in [0, N]$) sachant l'image $\mathbf{I}$. En d'autres termes, un CRF définit donc une distribution de probabilités sur les réalisations d'un ensemble de variables aléatoires. Décoder le modèle se fait communément en estimant la réalisation la plus probable (MAP, *maximum a posteriori*), ce qui revient à la minimisation de la fonction d'énergie.

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\operatorname{argmax}} P(\mathbf{X} = \mathbf{x}|\mathbf{I}) = \underset{\mathbf{x}}{\operatorname{argmin}} E(\mathbf{x})$$

La factorisation de la distribution $P(\mathbf{X} = \mathbf{x}|\mathbf{I})$ et donc la décomposition en termes de l'énergie $E(\mathbf{x})$ est déterminée par le graphe $\mathcal{G}$. Les facteurs/termes correspondent aux cliques du graphe.

Krähenbühl et Koltun [KK11] ont défini un CRF entièrement connecté (ou $\mathcal{E}$ contient une arrête pour chaque paire de sommet $(v_i, v_j) \in \mathcal{V}^2$) et ont défini la fonction d'énergie d'annotation des variables $\mathbf{x}$ par

$$E(\mathbf{x}) = \sum_i \upsilon_u(x_i) + \sum_{i<j} \upsilon_p(x_i, x_j)$$

Où $\upsilon_u(x_i)$ est le terme d'attache aux données qui maintient une liaison entre la variable x_i à estimer et les observations I, terme omis dans notre notation pour des raisons de simplicité. Lors de l'utilisation de CRF comme post-traitement d'un réseau de neurones, cette valeur correspond aux prédictions obtenues en sortie de réseau de neurones. La composante énergétique par paire $\upsilon_p(x_i, x_j)$, mesure le coût énergétique d'assigner, de manière simultanée, les étiquettes x_i et x_j aux pixels i et j de l'image. Cette composante assure la cohérence sémantique et spatiale entre les pixels de l'image (en donnant, par exemple, un cout élevé à l'assignation des étiquettes *Vache* et *Plage* à deux pixels quelconque d'une image). Dans le cas de la segmentation d'image, le modèle classiquement utilisé est une il est un modèle composé de noyaux Gaussiens pondérés de la forme suivante :

$$\upsilon_p(x_i, x_j) = \mu(x_i, x_j) \sum_{m=1}^{M} w^{(m)} k_G^{(m)}(\mathbf{f}_i, \mathbf{f}_j)$$

Dans cette formule, $k_G^{(m)}$ avec $m = 1, ..., M$ sont des noyaux Gaussiens pondérés par les poids $w^{(m)}$ et appliqués sur des vecteurs de caractéristiques $\mathbf{f}$ extraits à partir de l'image $\mathbf{I}$. La fonction $\mu(x_i, x_j)$ est appelée fonction de compatibilité des classes puisqu'elle mesure la compatibilité entre les différentes paires d'étiquettes. C'est cette fonction qui donnera, par exemple, un coût élevé à la combinaison (*Vache*, *Plage*) (car il est peu probable d'observer une vache sur une plage), mais un coût faible à la combinaison (*Voiture*, *Route*) plus probable.

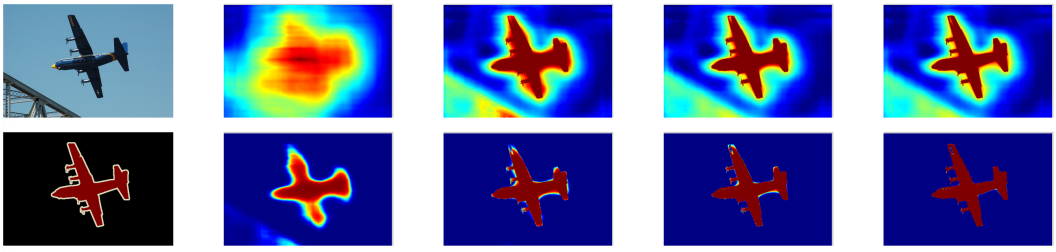


Fig. 4.19.: Exemple d'amélioration obtenue grâce à l'utilisation de CRF. La première ligne montre les valeurs obtenues avant la couche de softmax et la deuxième ligne les valeurs obtenues après le softmax. Image tirée du papier [Che+14a].

L'utilisation des CRFs entièrement connectés peut servir de post-traitement des prédictions obtenues par les réseaux de neurones convolutif, permettant ainsi d'améliorer les résultats. C'est, par exemple, le cas des modèles DeepLab [Che+14a ;

Che+16] de Chen *et al*. où l'utilisation de CRFs leur permet d'améliorer de 0.54% les résultats obtenus sur la base de données PASCAL-Person-Part [Che+14b] (passant de 64.40% à 64.94%). La figure 4.19 montre un exemple de sortie obtenue par un réseau et raffinée à l'aide de CRFs.

L'utilisation de CRFs afin de raffiner les prédictions d'un réseau, permet généralement d'améliorer légèrement les résultats. Néanmoins, l'utilisation de ces techniques ont un coût en termes de mémoire utilisée ainsi qu'en temps de calculs. C'est pourquoi la mise en place, en pratique, est compliqué. Il est nécessaire d'évaluer l'apport de la méthode et le coût (temps/mémoire) qu'elle implique.

Algorithm 3: Mean-field in dense CRFs [KK11], broken down to common CNN operations.

$Q_i(l) \longleftarrow \frac{1}{Z_i} exp(U_i(l))$ for all i ▷ Initialisation

while *pas convergé* **do**

$\tilde{Q}_i^{(m)}(l) \longleftarrow \sum_{j \neq i} k^{(m)}(f_i, f_j) Q_j(l)$ for all m ▷ Message passing

$\check{Q}_i \longleftarrow \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$ ▷ Weighting Filter Outputs

$\hat{Q}_i \longleftarrow \sum_{l' \in \mathcal{L}} \mu(l, l') \check{Q}_i(l')$ ▷ Compatibility Transform

$\breve{Q}_i(l) \longleftarrow U_i(l) - \hat{Q}_i(l)$ ▷ Adding Unary Potentials

$Q_i \longleftarrow \frac{1}{Z_i} exp(\breve{Q}_i(l))$ ▷ Normalizing

end

Dans leurs travaux, Zheng *et al*. [Zhe+15a] vont plus loin que la simple utilisation de CRFs comme post-traitement en montrant comment construire un réseau de neurones récurrent permettant de prendre en compte les cohérences spatiales à la manière d'un CRF entièrement connecté. Pour cela, ils se basent sur l'algorithme de Krähenbühl et Koltun [KK11] détaillé dans la figure 3. Cet algorithme est un algorithme itératif permettant d'obtenir une configuration $\mathbf{x}$ approximant une minimisation la fonction d'énergie $E(\mathbf{x})$ vue précédemment. Dans leur travaux, Zheng *et al*. reprennent les différentes étapes de l'algorithme et montrent comment les transformer en modules utilisables dans un réseau de neurones convolutif. Ils montrent que l'initialisation et la normalisation (voir algorithme 3) peuvent être considérées comme des opérations de softmax, que les étapes "Weighting Filter Outputs" et "Compatibility Transform" peuvent être considérées comme des modules convolutifs prenant en entrée différentes cartes de caractéristique et que l'étape "Adding Unary Potentials" est un simple module de soustraction des cartes de caractéristiques. L'étape la plus compliquée est celle appelée "Message Passing". Pour implémenter cette dernière dans un réseau de neurones convolutif ils se basent encore sur les

travaux de Krähenbühl et Koltun qui décrivent un algorithme permettant de calculer des filtres Gaussiens de grande dimension de façon efficace. Leur réseau de neurones est récursif afin de permettre la convergence de l'algorithme (la boucle "while" de l'algorithme). Ils montrent par la suite l'efficacité de leur réseau CRF-RNN en obtenant (à l'époque) les meilleurs résultats sur les ensembles de tests des années 2010, 2011 et 2012 de base de données Pascal VOC [Eve+10].

4.5 Conclusion

Dans ce chapitre nous avons introduit la tâche de segmentation sémantique. Nous avons vu les fonctions de pertes utilisées ainsi que les différentes métriques permettant de mesurer l'efficacité des algorithmes. Nous avons aussi présenté certaines des nombreuses bases de données existant aujourd'hui et permettant d'entraîner des réseaux de neurones (ou autre modèle) à la tâche de segmentation sémantique. Ces bases de données, de plus en plus nombreuses dû à la popularité des réseaux de neurones et à leur performance, répondent aux besoins de ces derniers en proposant de plus en plus d'images annotées. Finalement, nous avons présenté les limitations des réseaux classiques ainsi que les solutions apportées par la communauté. Nous avons vu comment, avec un réseau entièrement convolutif, il est possible d'obtenir des annotations denses, permettant d'annoter une image complète et comment, l'ajout de post-traitement permet d'améliorer les cohérences spatiales des prédictions.

Dans ce chapitre, on a étudié l'évolution des structures des réseaux, montrant ainsi qu'il est possible de les spécialiser à une tâche précise (ici la segmentation sémantique). Ces derniers temps, de nouvelles techniques ont été utilisées afin de reconstruire les cartes de caractéristiques et permettre des prédictions denses (par pixels). Les réseaux actuels ont dû faire face à de nouveaux problèmes comme, notamment, de garder des informations spatiales précises pour reconstruire correctement les frontières des différents éléments d'une scène tout en ayant des abstractions de haut niveau permettant de reconnaitre les objets. Les structures se sont donc adaptées en échangeant de plus en plus d'informations entre les deux parties des réseaux conv-déconv et la question est donc maintenant de savoir si cette information est suffisante. Nous pensons que des structures encore plus complexes, cassant complètement l'aspect de chemin principal dans un réseau (tel que notre architecture présentée dans le chapitre 6) permettraient aux réseaux d'effectuer des prédictions encore plus précises sans avoir besoin de recourir à des post-traitement tel que les CRF.

Notons également pour finir qu'une autre branche de recherche de plus en plus active s'intéresse à la segmentation d'instances (vue en introduction du chapitre 4.1)

qui demande, en plus de reconnaitre les classes des objets présents, de faire la distinction entre les différentes instances. Cette tâche est d'autant plus complexe que le nombre d'instances présentes dans une image n'est pas connu et est variable d'une image à l'autre. Ainsi, la segmentation d'instances, qui n'est pas abordée dans cette thèse, nécessite une adaptation des réseaux de neurones convolutifs, ce qui pourrait être une suite naturelle et intéressante pour nos travaux.

5 Une fonction de coût sélective

5.1 Introduction

Une des données clés permettant un bon apprentissage des réseaux de neurones convolutifs est le nombre d'exemples d'entraînement disponibles. Un nombre important d'exemples est nécessaire afin d'obtenir des bons résultats et d'empêcher un sur-apprentissage du réseau.

Nous avons déjà vu qu'il existe de nombreux réseaux entraînés sur la base de données ImageNet [Den+09] (voir partie 2.3.1) qui possèdent de nombreuses images d'entrainement (plus d'un million d'images). Nous avons vu aussi qu'il est possible d'utiliser les connaissances apprises par un réseau sur une base pour améliorer l'entraînement sur une autre. Typiquement, plutôt que d'initialiser les poids du réseau aléatoirement, il est plus efficace d'utiliser comme point de départ pour l'entrainement un réseau déjà entrainé sur ImageNet.

Néanmoins, cette technique nécessite un entraînement en deux temps : dans un premier temps un réseau est entrainé sur ImageNet (ou toute autre base possédant ne nombreux exemples) puis le réseau est ré-entrainé sur la base désirée. De plus, si l'on veut prendre en compte d'autres bases de données supplémentaires, afin de tirer parti d'un plus grand nombre d'exemples, il devient alors nécessaire de répéter les étapes d'entraînement autant de fois que de bases de données utilisées. Dans ce chapitre nous nous sommes intéressés à l'utilisation simultanée de plusieurs bases de données afin d'entraîner un unique réseau et ce en une seule étape d'entraînement.

5.2 Définition du problème : apprentissage multi-tâche et multi-domaine

Nous avons déjà donné dans la partie 4.1.1 une formulation des objectifs de la classification. Or ici nous nous intéressons au cas où nous utilisons plusieurs bases de données afin d'entrainer un unique réseau. La formulation du problème est donc différente.

Étant donné un ensemble d'images tirées d'un ensemble de K différentes bases de données, les paires composées d'un patch d'entrée x_i^k et d'une étiquette cible y_i^k sont groupées dans un ensemble $D_k = (x_i^k, y_i^k)$, où $k{=}1 \dots K$ et i indexe les patchs. Une des principales difficultés de l'utilisation de plusieurs bases de données est que la sémantique et le nombre d'étiquettes sont différents pour chacune des bases. Ainsi, chaque cible y_i^k prend une valeur dans un ensemble d'étiquettes $\mathcal{L}^k$ dépendant de la base de données k.

Mise à part cette différence d'ensemble d'étiquettes, le reste de l'objectif est défini de façon classique, c'est-à-dire que notre but est d'apprendre une fonction de correspondance non linéaire $\hat{y} = \Theta(x, \theta)$ exprimée sous forme de réseau de neurones et dont les paramètres θ minimisent un risque $\mathcal{R}(\hat{y}, y)$. Le risque que nous minimisons pour entrainer le réseau de neurones Θ est le risque empirique $\mathcal{R}(\Theta(x, \theta), y) = \frac{1}{N} \sum_{k=1}^{n} J(x, y, \theta)$, où N est le nombre d'exemples d'entrainement et J est la fonction de perte utilisée pour attribuer une valeur d'erreur à chacun des exemples. Nous avons déjà vu dans le chapitre 4.1.1 que, pour les problèmes de classification, il est courant d'utiliser la fonction de perte d'entropie croisée ("cross entropy") définie de la manière suivante : $J(x, y, \Theta) = -\sum_j 1_{y=j} \log \Theta(x, \theta)_j$, où $\Theta(x, \theta)_j$ est la sortie du réseau pour la classe j.

La différence entre les ensembles d'étiquettes $\mathcal{L}^k$ positionne nos travaux dans la catégorie des problèmes multi-tâches. En effet, même si l'objectif est commun (par exemple la segmentation sémantique d'images), l'utilisation de différents ensembles d'étiquettes induit en soit plusieurs tâches. On peut donner l'exemple de deux types de segmentation d'images prises à partir d'un véhicule en mouvement, où un ensemble d'étiquettes serait plutôt orienté sur la sémantique des objets présents sur la route (voiture, piéton, moto) et un autre ensemble plutôt focalisé sur la topologie de l'environnement (où s'arrête la route, où sont les panneaux etc.).

L'utilisation de plusieurs bases de données place aussi nos travaux dans la catégorie des problèmes d'adaptation de domaines. L'adaptation de domaine est une catégorie de problèmes où il existe des différences de distributions entre les données d'entrée (souvent entre les données d'apprentissage et de tests, même si ce n'est pas notre cas ici-présent). Dans notre cas, bien que les entrées soient toutes de même nature (*e.g.* des images RGB), la distribution de ces dernières peut varier entre les bases de données. Par exemple, les images capturées depuis un véhicule, pourraient être obtenues dans deux villes différentes, ou dans des conditions météorologiques différentes ou plus extrême encore, une des bases de données pourrait contenir des images capturées non pas depuis un véhicule mais depuis un drone.

Finalement, on pourrait aussi vouloir considérer notre tâche comme faisant partie des problèmes de classification multi-étiquette, mais ce n'est pas le cas. En effet,

la classification multi-étiquette vise à prédire plusieurs classes sémantiques par entrée (par exemple une voiture est aussi un véhicule). Cela implique donc des recouvrements entre la sémantique des différentes classes d'un ensemble d'étiquettes. Dans notre cas, même si ce recouvrement existe potentiellement entre les différents ensembles d'étiquettes utilisés, ils ne sont pas connus et nos exemples d'entrainement n'ont donc qu'une seule étiquette associée. L'entraînement de notre réseau est donc réalisé de manière à ne prédire qu'une seule étiquette par entrée d'entrainement et uniquement sur l'ensemble d'étiquettes dont il provient.

5.3 Notre fonction de coût sélective

5.3.1 Approche classique

Étant donné K différentes bases de données, possédant chacune son propre ensemble d'étiquettes $\mathcal{L}^k$, l'approche classique (que nous appelons *Baseline*) consiste à entrainer un réseau de neurones Θ^k différent (ou n'importe quelle autre fonction de correspondance) pour chacune des bases de données, chaque réseau produisant une prédiction dans l'ensemble d'étiquettes associé à sa base de données d'entrainement.

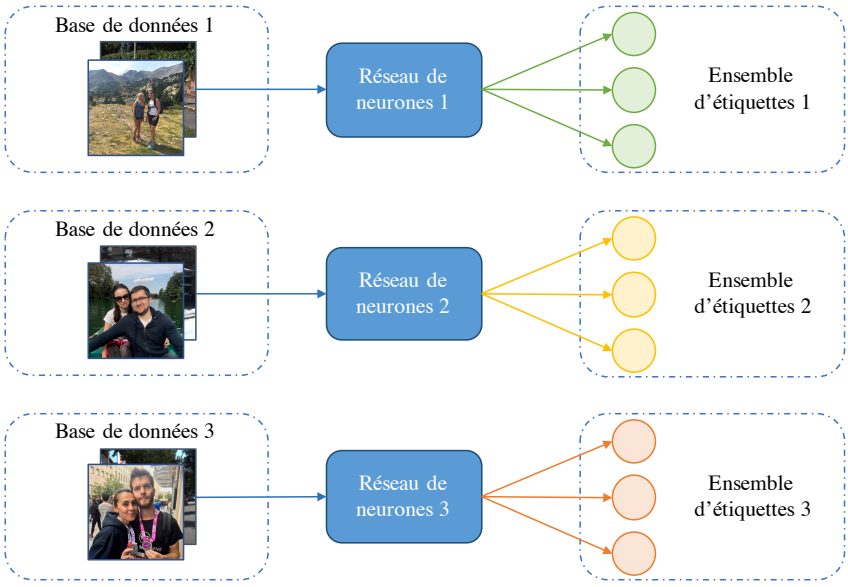


Fig. 5.1.: Stratégie d'apprentissage classique appelée "*Baseline*". Elle consiste à entraîner un réseau de neurones par base de données (et ensemble d'étiquettes) différents.

Cette approche de référence, qui est illustrée dans la figure 5.1, présente deux défauts :

(i) Chaque réseau Θ^k est indépendant et est entrainé avec une base de données D^k associée qui lui est propre. Par conséquent, l'entrainement consiste à optimiser

les paramètres θ^k (propre à chacun des réseaux) de manière indépendante. Or dans le cas des réseaux de neurones convolutifs, l'ensemble des paramètres θ^k est considérable (plus de deux millions de paramètres dans le cas du réseau utilisé pour nos expériences) et nécessite donc un nombre de données importantes. Par conséquent, il pourrait être intéressant d'optimiser un unique ensemble de paramètres θ sur l'ensemble des données disponibles afin de permettre un meilleur apprentissage des représentations.

(ii) Les relations entre les ensembles d'étiquettes ne sont ni modélisées ni exploitées. Par exemple, si deux ensembles d'étiquettes possèdent tous deux une classe sémantique identique il est dommage d'entraîner deux réseaux de neurones de manière indépendante, puisque ces derniers auront à apprendre les mêmes représentations. De plus, les relations, existantes potentiellement entre les classes, peuvent aider l'apprentissage. En effet, si un ensemble d'étiquettes possède, par exemple, une classe véhicule, l'entrainement du réseau sur cette classe pourra aider à distinguer des classes présentes dans un autre ensemble d'étiquettes tel que, par exemple, voiture et camion.

5.3.2 Apprentissage joint avec notre fonction de coût sélective

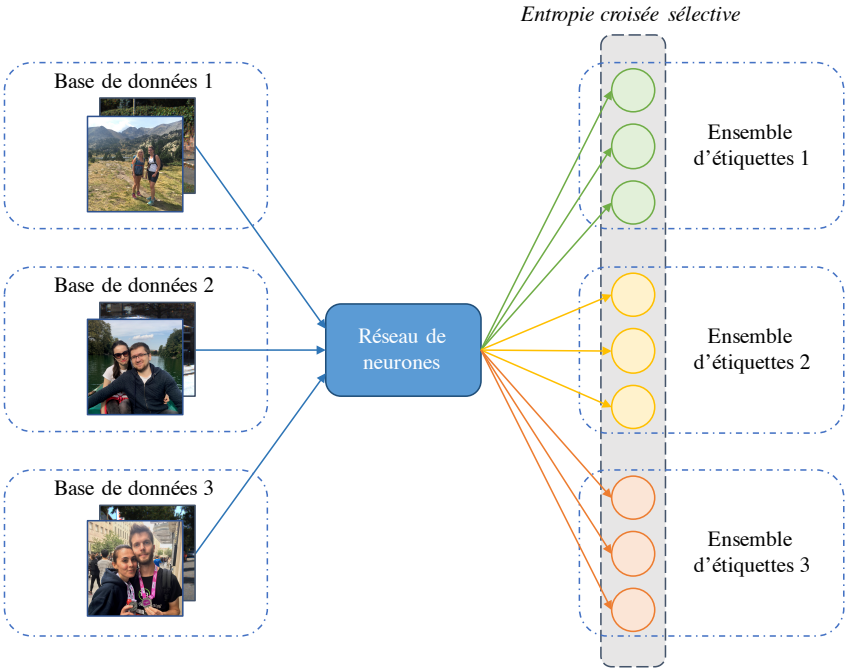


Fig. 5.2.: "*Apprentissage Joint*" (AJ) d'un unique réseau utilisant notre fonction d'entropie croisée sélective.

Afin d'utiliser toutes les données disponibles et donc résoudre le problème (i) présenté dans la partie 5.3.1, nous utilisons un unique réseau de neurones convolutif entrainé sur l'ensemble des bases de données disponibles. Il est connu que les pre-

mières couches des réseaux de neurones convolutifs apprennent des caractéristiques générales et que les couches plus profondes apprennent des caractéristiques de plus en plus complexes et spécifiques à la tâche étudiée [ZF14]. Nous utilisons cette caractéristique hiérarchique des réseaux de neurones convolutifs afin d'entrainer un unique réseau sur l'union de toutes les bases de données. Un plus grand nombre de données permettra aux premières couches de convolutions d'apprendre de meilleures représentations (plus génériques), aidant ainsi les couches plus en profondeur (plus proche de la sortie) et donc plus spécifiques aux tâches étudiées. Cette approche, que nous avons appelé *Apprentissage Joint* (JT), est illustrée dans la figure 5.2.

La stratégie est la suivante : un unique réseau est entrainé avec l'ensemble des bases de données. Il permet une prédiction pour chacune des étiquettes présentes dans l'union des ensembles d'étiquettes $\mathcal{L}^k$ associées aux bases de données k utilisées. Comme aucune connaissance sur les relations entre les étiquettes provenant de différents ensembles n'est disponible, on considère toutes les étiquettes comme étant indépendantes, même si, dans les faits, certaines classes peuvent être communes. On construit donc un ensemble de prédiction dont la cardinalité est égale à la somme de toutes les cardinalités des différents ensembles d'étiquettes. Ainsi, la sortie du réseau est un vecteur de probabilités, où chaque élément correspond à une étiquette donnée dans un ensemble d'étiquettes qui lui est propre. Par conséquent, le réseau est capable de donner une prédiction pour chacune des bases de données utilisées lors de son entraînement.

Nous avons vu dans la partie 4.1.1 que, lors de l'entrainement sur une tâche de classification, il faut adapter la sortie du réseau afin que cette dernière produise un vecteur de probabilités. Pour cela une fonction appelée *softmax* est ajoutée à la sortie du réseau (en remplacement de la fonction de non-linéarité de la dernière couche). Les paramètres du réseau sont ensuite modifiés par descente de gradient (voir partie 2.2.1) afin d'optimiser l'erreur obtenue de façon stochastique sur chaque exemple des bases d'entrainement. Néanmoins, avec K bases de données différentes, il serait inapproprié d'optimiser les probabilités des classes en les considérant toutes indépendantes. En effet, maximiser la probabilité d'une classe cible implique de minimiser les probabilités des autres classes (censées être mutuellement exclusives) et donc, dans notre cas, les classes appartenant aux autres ensembles d'étiquettes aussi. Malheureusement, cette minimisation devient problématique quand il existe une corrélation entre les classes provenant de différents ensembles d'étiquettes, car elle pousse le réseau à pénaliser des classes qui ne devraient pas l'être. Pour donner un exemple concret, lors de l'utilisation des sous-ensembles de la base de données KITTI (voir figure 5.6 où toutes les étiquettes définies sont reportées), la classe *Arbre* du sous-ensemble annoté par l'équipe de He *et al.* [HU13] est corrélée à la classe *Végétation* du sous-ensemble annoté par l'équipe de Kundu *et al.* [Kun+14]. Ainsi, lorsque l'on donne un exemple d'entrainement annoté *Arbre de He* et al., il est erroné

d'entrainer le réseau à ne pas prédire aussi *Végétation de Kundu* et al.. Or c'est ce qui se passe avec une fonction de softmax classique.

Afin de ne pas pénaliser les classes provenant de différents ensembles d'étiquettes et donc afin de prendre en compte les dépendances possibles entre les étiquettes, nous avons donc défini une fonction que nous avons appelée *softmax sélectif* correspondant à une fonction de softmax appliquée sur chacun des ensembles d'étiquettes indépendamment.

Def. 5.1 *Soft-max sélectif*

Pour chaque étiquette $j \in \mathcal{L}^k$ appartenant à un ensemble d'étiquettes k notre fonction de softmax sélectif est définie par :

$$f(j, \Theta(x, \theta)) = \frac{e^{\Theta(x,\theta)_j}}{\sum\limits_{j' \in \mathcal{L}^k} e^{\Theta(x,\theta)_{j'}}} \tag{5.1}$$

La différence entre notre softmax sélectif et un softmax classique réside dans la somme du dénominateur. Cette dernière, au lieu de s'appliquer sur la totalité du vecteur de valeurs produit par le réseau Θ, ne s'applique que sur les éléments appartenant à l'ensemble d'étiquettes k de la classe cible considérée ($j' \in \mathcal{L}^k$).

En pratique, durant l'entraînement, notre softmax sélectif est combiné à la fonction de perte d'entropie croisée pour former ce que l'on a appelé la fonction de perte d'entropie croisée sélective :

Def. 5.2 *Entropie croisée sélective*

$$J'(k, x, y, \theta) = -\Theta(x, \theta)_y + \log(\sum_{j \in \mathcal{L}^k} e^{\Theta(x,\theta)_j}) \tag{5.2}$$

L'entraînement du réseau est réalisé de manière standard, c'est-à-dire en calculant le gradient des paramètres par rétro-propagation (voir partie 2.2.2), puis en les modifiants par descente de gradient afin d'optimiser une fonction objectif. Or on peut constater, dans l'équation 5.2, que l'erreur calculée n'est induite que par les valeurs obtenues pour les classes appartenant au même ensemble d'étiquettes que la classe cible. Ainsi, le calcul du gradient associé à l'erreur n'inclus pas les classes en dehors de l'ensemble d'étiquettes de la classe cible et donc la descente de gradient ne les pénalise pas. Cela est équivalent à utiliser une couche de sortie différente par base de données considérée et d'utiliser les couches intermédiaires du réseau de manière partagée entre les bases de données.

5.3.3 Modéliser les corrélations entre les ensembles d'étiquettes

Nous avons vu dans la partie 5.3.1 qu'il existe deux limitations si l'on utilise une approche classique, à savoir l'entrainement d'autant de réseau de neurones qu'il n'y a de bases de données et l'absence de prise en compte des corrélations entre les étiquettes. La première limitation est adressée avec notre fonction d'entropie croisée sélective vue dans la partie précédente (partie 5.3.2). La deuxième limitation est en partie adressée par l'apprentissage joint expliqué dans la partie précédente. En effet, la nature hiérarchique des réseaux de neurones convolutifs permet à notre réseau de prendre en compte implicitement les corrélations des classes. Les couches cachées (intermédiaires) du réseau étant communes, les représentations apprises par ces dernières seront partagées entre les différents ensembles d'étiquettes. Néanmoins, il est possible aussi de modéliser explicitement cette corrélation afin d'améliorer les capacités de discrimination du réseau.

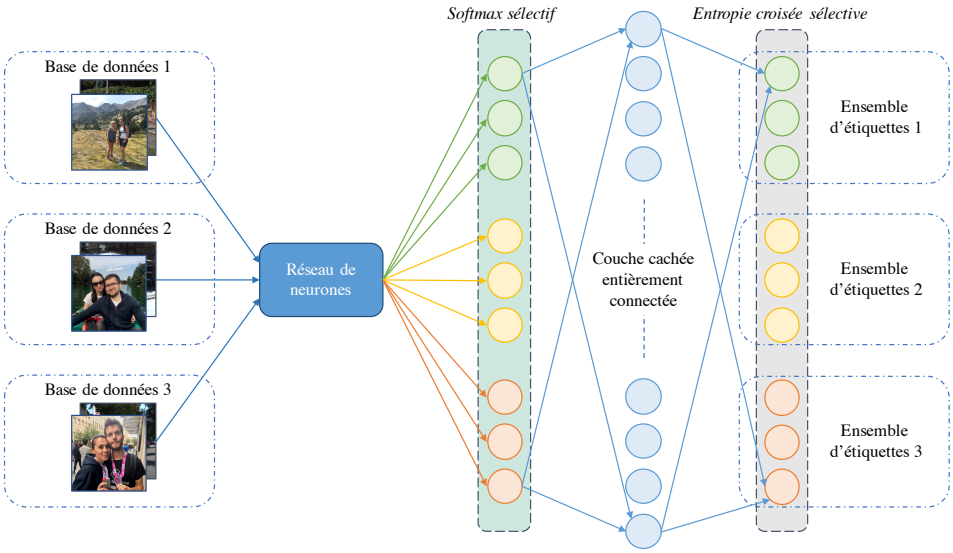


Fig. 5.3.: "*Apprentissage joint avec partage de contexte*" (AJPC). On ajoute une couche entièrement connectée (FC) après la sortie d'un premier réseau, pré-entraîné avec la stratégie d'apprentissage joint (figure 5.2). Une deuxième phase d'apprentissage est ensuite réalisée afin de prendre en compte les corrélations entre les classes sémantiques des différents ensembles d'étiquettes.

Pour cela, nous partons d'un premier réseau $\Theta(x, \theta)$ déjà entrainé sur l'ensemble des bases de données avec notre fonction de perte sélective. Ensuite, une couche entièrement connectée (FC) est ajoutée après notre softmax sélectif (voir figure 5.3) créant ainsi un nouveau réseau $\Theta'(x, \theta')$. Ce dernier est ensuite entrainé à nouveau en utilisant toujours notre fonction d'entropie croisée sélective. Cette approche, appelée *Apprentissage joint avec partage de contexte* (AJPC) est illustrée dans la figure 5.3. L'idée derrière cette stratégie est qu'un réseau déjà entraîné avec notre softmax sélectif extrait des informations importantes, disponibles dans le vecteur de probabilité

de sortie. En effet, le réseau fournissant une prédiction par ensemble d'étiquettes, les corrélations sont présentes dans les probabilités de sortie. L'utilisation d'une couche supplémentaire a pour but d'exploiter ces probabilités afin d'améliorer la prédiction finale.

5.3.4 Inversion de gradient pour l'adaptation de domaine

Jusqu'à présent nous avons vu comment utiliser plusieurs bases de données dont les ensembles d'étiquettes sont "différents". Par contre, nous n'avons pas pris en compte les possibles différences entre les données d'entrées. Les entrées ou les ensembles d'étiquettes sont considérés comme "différents" lorsqu'ils ne possèdent pas les mêmes distributions. Pour les ensembles d'étiquettes, cela correspond, par exemple, à différentes sémantiques dans les étiquettes utilisées. Pour les images d'entrée, les différences se retrouvent dans les changements de résolution, la diversité des lieux photographiés ou même dans les conditions d'acquisition (photographies de scènes d'extérieur ou d'intérieur). Si notre fonction de coût sélective permet de prendre en compte les différences entre les ensembles d'étiquettes utilisés, les différences de distributions entre les images des différentes bases ne sont pas prises en compte. En effet, si l'on utilise de façon jointe les bases de données SiftFlow [Liu+11] et Stanford Background [Gou+09], alors les différences entre les images d'entrée sont conséquentes. Les appareils utilisés ne sont pas les mêmes, la qualité des photos varie et les milieux photographiés sont différents. Afin d'améliorer l'entrainement de notre réseau de neurones, il est important de prendre en compte ces différences quand elles existent. Attention, néanmoins, ces différences ne sont pas systématiques. Par exemple nous verrons dans la partie expériences (partie 5.4.2) que la base de données KITTI (présentée dans la partie 4.1.3) est composée de plusieurs sous-ensembles, annotés par plusieurs équipes (et donc avec des ensembles d'étiquettes différents). Par conséquent, cette base présente la particularité d'avoir des entrées venant de la même distribution, mais des étiquettes relativement différentes, quoique fortement corrélées.

La prise en compte des différences de distribution entre différents domaines s'appelle l'adaptation de domaine. La théorie en adaptation de domaine nous dit que, afin d'obtenir un meilleur entraînement sur des méthodes d'apprentissage, il est nécessaire de rapprocher les distributions des caractéristiques extraites sur les différents domaines [Ben+10 ; Man+09]. Concrètement, cela veut dire que si l'on veut qu'une méthode d'apprentissage fonctionne de façon optimale sur deux domaines dont les entrées sont "différentes", il faut être capable d'en extraire des caractéristiques identiques. L'extraction de caractéristiques projette ainsi les données d'entrée dans un même espace de caractéristiques, permettant ainsi au classifieur de fonctionner de manière identique sur les différents ensembles d'entrée.

Dans les réseaux de neurones convolutifs, les premières couches de convolutions correspondent à une extraction de caractéristiques. C'est sur ces dernières qu'il faut agir afin de forcer un rapprochement de domaines. Pour cela nous nous sommes basé sur les travaux de Ganin et Lempitsky [GL15]. Dans leurs travaux, Ganin et Lempitsky entraînent un réseau de neurones en utilisant un domaine source, c'est-à-dire une base de données dont les étiquettes sont connues, et le force à être suffisamment générique pour fonctionner sur un domaine cible, c'est-à-dire une base de données dont les étiquettes ne sont pas fournies (en réalité les étiquettes existent, mais ne sont utilisées que pour évaluer la méthode et donc pas lors de l'apprentissage). Pour cela, ils entraînent un classificateur de domaine, utilisant les mêmes caractéristiques que celles extraites par leur réseau de neurones et qui a pour objectif de prédire si une image donnée en entrée du réseau provient du domaine source ou du domaine cible. La particularité est que, durant l'entrainement du classificateur de domaine, l'extracteur de caractéristiques est modifié afin de rendre indistinguable les domaines.

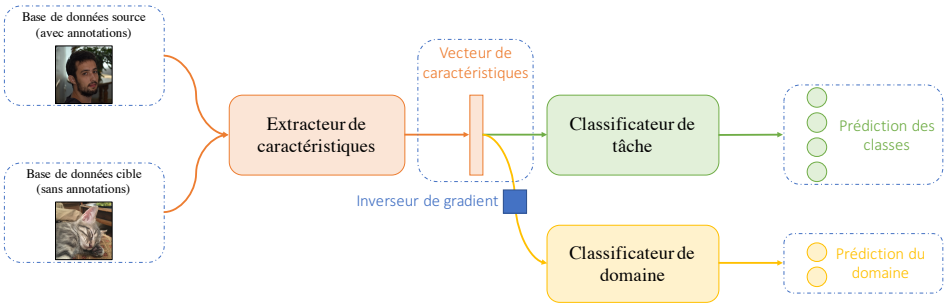


Fig. 5.4.: Illustration de l'inverseur de gradient de Ganin et Lempitsky [GL15]. L'extracteur de caractéristiques (en orange) extrait des caractéristiques utilisées par les deux classificateurs. Le classificateur de tâche (en vert) est entrainé à prédire les classes des images venant de la base de données source. Le classificateur de domaine (en jaune) est entraîné à prédire la provenance des images (base de données source ou base de données cible). L'inverseur de gradient (en bleu) inverse le gradient qui remonte du classificateur de domaine ayant pour effet d'entraîner l'extracteur de caractéristiques à extraire des données invariantes aux bases de données source et cible.

Plus formellement, le réseau de neurones $y = \Theta(x, \theta)$ est divisé en deux parties : un extracteur de caractéristiques $f = \Theta_f(x, \theta_f)$ correspondant aux couches de convolutions et produisant des cartes de caractéristiques notées f et un classificateur de tâches $y = \Theta_t(f, \theta_t)$ correspondant aux couches entièrement connectées et, dans notre cas, incluant notre fonction de perte sélective. Le classificateur de domaine $d = \Theta_d(f, \theta_d)$ est ajouté et réalise la correspondance entre les cartes de caractéristiques f et l'estimation du domaine (source ou cible) d'où proviennent les données (voir figure 5.4). L'objectif est d'optimiser les paramètres θ_d pour minimiser l'erreur faite par le classificateur de domaine Θ_d afin qu'il soit le plus performant possible, tout en modifiant les paramètres θ_f de l'extracteur de caractéristiques Θ_f afin de rendre les domaines indistinguables, c'est-à-dire d'extraire des caractéristiques qui

ne permettent pas au classificateur de domaine d'être efficace. Ainsi, si les cartes de caractéristiques f extraites par l'extracteur Θ_f ne permettent pas au classificateur de domaine Θ_d de distinguer correctement les deux domaines, cela veut dire que les caractéristiques f sont invariantes aux domaines.

Afin d'entrainer les trois modules (l'extracteur de caractéristiques Θ_f, le classificateur de tâches Θ_t et le classificateur de domaines Θ_d) Ganin et Lempitsky utilisent les propriétés de la rétro-propagation. De manière classique le calcul du gradient remonte l'erreur générée par les classificateurs jusqu'à l'extracteur de caractéristiques, tandis qu'un inverseur de gradient est placé entre le classificateur de domaine et l'extracteur de caractéristiques (voir figure 5.4). L'idée étant qu'en inversant le gradient, au lieu de minimiser l'erreur lors de la mise à jour des poids du réseau, cette dernière est maximisée. En pratique, le gradient est multiplié par un hyper-paramètre $-\lambda$ inversant le gradient et évoluant au court de l'apprentissage, permettant ainsi de contrôler l'importance donnée aux classificateurs de tâches et de domaine.

Dans nos expérimentations, décrites dans la section 5.4, nous avons utilisé cette technique afin de rendre les caractéristiques indépendantes vis à vis des différentes bases de données utilisées. Nous verrons dans la partie 5.4.3 que cela aide l'entrainement du réseau lorsque les données d'entrée sont très différentes entre les différentes bases.

5.4 Résultats sur la segmentation sémantique

Comme la totalité des expérimentations décrites dans ce manuscrit, ces dernières ont été effectuées en Lua avec le framework Torch7 [Col+11]. Néanmoins, contrairement aux expérimentations sur la constance chromatique (voir chapitre 3) pour ces dernières nous avons pu utiliser la puissance de GPU (carte graphique) NVIDIA de type Titan X. Ces dernières, de par leur architecture et leur puissance de calcul, permettent d'accélérer considérablement les calculs et donc d'entrainement des réseaux (cela va parfois jusqu'à un facteur 1000 en comparaison avec un entrainement sur CPU).

5.4.1 Détails de l'entrainement

Pour nos expérimentations, nous avons utilisé une architecture du réseau inspirée par Farabet *et al.* [Far+13] et développée pour la segmentation sémantique de scènes. Nous avons modifié cette architecture pour prendre en compte les avancées en matière d'apprentissage profond répertoriées au moment où nous nous sommes

lancés dans ce travail. Notre réseau est composé de trois couches de convolutions suivies par deux couches entièrement connectées.

Le réseau est illustré dans la figure 5.5. Un opérateur de normalisation par batchs [IS15a] (voir partie 2.2.4) est utilisé en entrée du réseau, ce qui permet de ne pas normaliser les images avant de les utiliser dans le réseau (puisque cette normalisation sera apprise par l'opérateur). Les deux premières couches de convolutions (couches 1 et 2 du tableau 5.5) sont composées de filtres de convolution de taille 7×7, suivies par des opérateurs de non linéarité de type ReLU ainsi qu'une opération de sous-échantillonnage de maximum pooling de taille 2×2. Un opérateur de normalisation par batchs a aussi été ajouté après les opérateurs de sous-échantillonnages. La dernière couche de convolutions (couche 3) possède, elle aussi, un masque de taille 7×7 et est aussi suivie d'un opérateur d'une fonction d'activation de type ReLU, mais aucun opérateur de sous-échantillonnage n'a été ajouté. Un opérateur de normalisation par batchs ainsi qu'un opérateur de dropout (voir partie 2.2.3) avec une probabilité d'annulation de 30% ont été utilisés à la sortie de la couche. La première couche entièrement connectée (couche 4) est ensuite suivie pas un opérateur ReLU et la dernière couche utilise notre fonction de softmax sélectif.

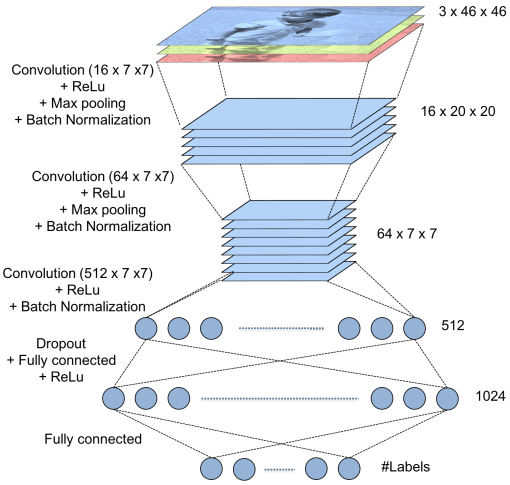
Pour la stratégie "*Apprentissage joint avec partage de contexte*" (AJPC) illustrée dans la figure 5.3, une couche entièrement connectée est ajoutée à la suite du réseau décrit ci-dessus. Notre opérateur de softmax sélectif est aussi utilisé pour la sortie de cette couche.

Afin d'entraîner le réseau, les images RGB (Rouge, Vert, Bleu) sont converties dans un espace de couleur appelé YUV où la composante Y correspond à la luminance et les composantes U et V aux composantes chromatiques. Un exemple d'entraînement est composé d'un patch x_i de taille 46×46, coupé de manière aléatoire dans une image, ainsi que de la base de données k d'où provient l'image et de l'étiquette y_i correspondant à la classe du pixel au centre du patch. L'algorithme de descente de gradient stochastique est utilisé pour mettre à jour les paramètres. La mémoire des GPU nous permet d'utiliser un mini-batch de taille 128. La seule stratégie d'augmentation de données utilisée consiste en un retournement horizontal (avec une rotation sur l'axe vertical) des patches d'entrée.

5.4.2 Détails des bases de données

Pour nos expérimentations nous avons utilisé trois bases de données différentes.

La base de données KITTI possède de nombreuses images extraites de vidéos acquises à l'aide d'un véhicule circulant dans les rues de la ville de Karlsruhe en Allemagne.



(a) Schéma du réseau utilisé pour nos expérimentations

	name	kernel size	output size
Input	input	-	$3 \times 46 \times 46$
	batch-norm-0	-	$3 \times 46 \times 46$
Couche 1	conv-1	7×7	$16 \times 40 \times 40$
	relu-1	-	$16 \times 40 \times 40$
	max-pooling-1	2×2	$16 \times 20 \times 20$
	batch-norm-1	-	$16 \times 20 \times 20$
Couche 2	conv-2	7×7	$64 \times 14 \times 14$
	relu-2	-	$64 \times 14 \times 14$
	max-pooling-2	2×2	$64 \times 7 \times 7$
	batch-norm-2	-	$64 \times 7 \times 7$
Couche 3	conv-3	7×7	$512 \times 1 \times 1$
	relu-3	-	$512 \times 1 \times 1$
	batch-norm-3	-	$512 \times 1 \times 1$
Couche 4	dropout-4	0.3	512
	fully-connected-4	-	1024
	relu-4	-	1024
Output	fully-connected		$\#labels$

(b) Détails des opérateurs

Fig. 5.5.: Schéma et détails du réseau de neurones convolutif utilisé pour nos expérimentations.

De nombreux capteurs ont été utilisés et permettent d'avoir des informations variées, mais aucune annotation pour la tâche de segmentation sémantique n'a été fournie. Heureusement, plusieurs équipes de recherches ont annoté des sous-parties de cette base d'images. Au total, 736 images annotées étaient disponibles à l'époque de nos travaux (d'autres images ont depuis été rajoutées). Nous avons séparé ces images en ensembles d'entrainement, de validation et de test. Nous avons utilisé les mêmes divisions que les auteurs quand ces derniers ont donné cette information, sinon nous avons séparé de manière conventionnelle les données avec un ratio d'environ 70% pour les ensembles d'entrainement et de validation et 30% pour l'ensemble de test.

Les différents sous-ensembles ayant été annotés par différents groupes, les ensembles d'étiquettes utilisés diffèrent d'un sous-ensemble à l'autre. Cette caractéristique de la base de données KITTI fait son originalité et la rend extrêmement bien adaptée à

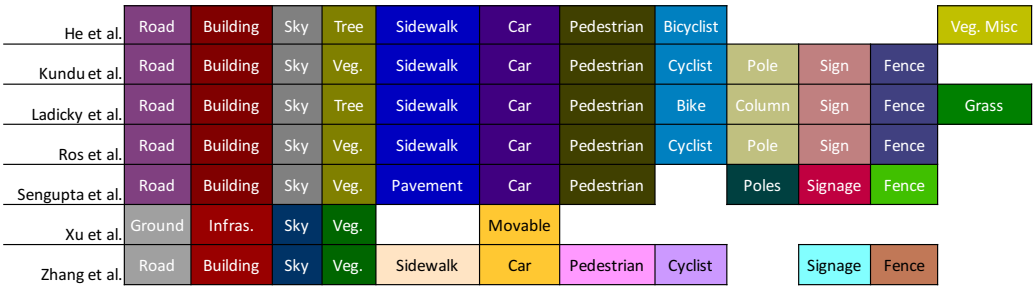


Fig. 5.6.: Les 68 étiquettes utilisées par les différentes équipes pour annoter les sous-parties de la base de données KITTI. Les couleurs correspondent aux couleurs utilisées par les auteurs pour leurs illustrations.

Data	Train	Val	Test	Total
He [HU13]	32	7	12	51
Kundu [Kun+14]	28	7	15	50
Ladicky [Lad+14]	24	6	30	60
Ros [Ros+15]	80	20	46	146
Sengupta [Sen+13]	36	9	25	70
Xu [Xu+13]	56	14	37	107
Zhang [Zha+15]	112	28	112	252
Total	368	91	277	736

Fig. 5.7.: Nombres d'images composant les différentes sous-parties ainsi que leur affectation dans les ensembles d'entraînement, de validation et de test.

notre étude. Les différentes étiquettes utilisées par les auteurs sont résumées dans la figure 5.6. A noter que l'équipe de Xu *et al.* [Xu+13] fournit une hiérarchie d'étiquettes très détaillée, mais nous n'utilisons que le plus haut niveau de la hiérarchie afin d'obtenir un ensemble d'étiquettes plus compatible (en termes de granularité) avec les aux autres ensembles.

La base de données KITTI contient plus de 40 mille images extraites de vidéos (dont la taille totale dépasse les 180 GB) mais nous ne pouvons utiliser que les données annotées, soit 736 images (le détail du nombre d'images par sous-partie est donné dans le tableau 5.7).

Le nombre d'images est faible mais ces dernières sont annotées de manière dense. Par conséquent, chaque pixel peut être considéré comme un échantillon d'entrainement. Ainsi, nous pouvons extraire environ 390 mille patchs de chaque images (dépendant de sa taille) pour un total de plus de 280 millions d'exemples d'entraînement utilisables lors de l'apprentissage du réseau. Bien sûr, il est important de noter que les différents patchs sont souvent très proches les uns des autres et qu'il existe donc une très grande corrélation entre certains exemples d'entrainement.

De plus, comme mentionné précédemment, les étiquettes utilisées par les différentes équipes ne sont pas toujours consistantes. La figure 5.6 montre les différences entre les étiquettes. On peut voir, par exemple, que l'équipe de Ladicky *et al.* [Lad+14] a séparé la classe *Arbre* de la classe *Herbe*. Ces deux classes pourraient correspondre à la classe *Végétation* de l'équipe de Xu *et al.* [Xu+13], mais ces derniers ont aussi une classe *Terrain*. La classe *Herbe* peut-elle aussi faire partie de la classe *Terrain* ? On peut aussi pointer que l'équipe de He *et al.* [HU13] n'utilise pas les étiquettes *Poteaux*, *Panneaux* et *Barrières* utilisées par plusieurs autres équipes. On pourrait donc considérer que ces dernières font partie de la classe *Bâtiment* de He *et al.*, mais dans ce cas cette dernière n'est plus cohérente avec les étiquettes *Bâtiment* des autres équipes. Finalement, certains ensembles d'étiquettes possèdent une étiquette *Vélo* quand d'autres l'étiquette *Cycliste*. Ces deux étiquettes pourraient présenter des points communs, mais dans un cas les équipes s'intéressent à une personne et son vélo comme une unique entité quand d'autres se concentrent uniquement sur le vélo. Ces incompatibilités d'étiquettes ne permettent donc pas de créer un ensemble cohérent pour un apprentissage classique et fait donc de la base de données KITTI un parfait candidat pour notre fonction de perte sélective.

En plus de la base de données KITTI, qui se prête très bien à notre approche, nous avons aussi utilisé deux autres bases de données conçues pour la segmentation de scènes : la base de données Stanford Background [Gou+09] et la base de données SiftFlow [Liu+11]. Comme nous l'avons vue dans la partie 4.1.3 la base de données Stanford Background contient 715 images de scènes extérieures avec une résolution de 320×240 pixels. Nous avons suivi les recommandations d'utilisation de la base de données et avons séparé aléatoirement les images pour en conserver 80% dans l'ensemble d'entrainement/validation et 20% dans l'ensemble de tests. L'ensemble d'étiquettes possède 9 classes : *ciel, arbre, route, herbe, eau, bâtiment, montagne, objets de premier plan* et *inconnu*. La résolution et le nombre d'images nous permet d'extraire environ 40 millions de patchs d'entrainement. La base de données SiftFlow contient 2688 images d'une résolution de 256×256 pixels permettant l'utilisation de plus de 160 millions de patchs d'entrainement. Une séparation de 2488 images d'entrainement/validation et de 200 images de tests est donnée par les auteurs. Les images de la base ont été annotées manuellement dans un ensemble de 33 classes sémantiques.

5.4.3 Résultats des différentes stratégies

Le tableau 5.1 montre nos résultats obtenus avec nos différentes stratégies d'entrainement. Pour un souci de clarté le détail des résultats des différents sous-ensembles de la base de données KITTI n'est pas montré mais une version détaillée des tables est disponible en annexe A. Nous reportons la précision par pixels (précision globale),

	7 KITTI		
Méthodes	Globale	Moyenne	IoU
Baseline	77.35	54.51	41.99
AJ	80.71	58.62	46.21
AJ + EB	80.84	58.89	46.32
AJPC	**81.75**	**59.67**	**47.10**

	Learning with Stanford + SIFTFlow								
	Stanford			SIFTFlow			Précision sur les deux		
Méthode	Glob.	Moy.	IoU	Glob.	Moy.	IoU	Glob.	Moy.	IoU
Baseline	74.80	64.35	51.41	69.79	24.95	19.02	72.66	33.24	25.84
AJ	75.79	67.22	**54.43**	71.55	28.34	21.35	73.99	36.52	28.31
AJ+RG	75.07	67.76	53.80	70.82	25.08	19.06	73.26	34.07	26.37
AJ+EB	75.28	67.37	52.93	71.11	**29.01**	**22.25**	73.51	**37.08**	**28.71**
AJ+RG+EB	**76.04**	67.71	54.05	**71.65**	27.40	20.54	**74.17**	35.89	27.60
AJPC+RG+EB	75.45	**68.33**	53.83	71.39	28.03	21.09	73.73	36.51	27.98

	Learning with 7 KITTI + Stanford								
	7 KITTI			Stanford			Précision sur les deux		
Méthode	Glob.	Moy.	IoU	Glob.	Moy.	IoU	Glob.	Moy.	IoU
Baseline	77.35	54.51	41.99	**74.80**	64.35	51.41	77.09	55.64	43.10
AJ	80.40	58.42	45.93	73.38	63.42	50.51	79.64	58.97	46.43
AJ+RG	80.73	58.33	46.08	72.42	62.79	49.31	79.82	58.82	46.43
AJ+EB	80.76	58.65	46.24	73.93	63.87	51.19	80.02	59.23	46.78
AJ+RG+EB	81.20	58.25	46.28	74.39	**65.31**	**52.04**	80.46	59.02	46.91
AJPC+RG+EB	**81.26**	**60.24**	**47.92**	74.70	63.44	50.88	**81.08**	**60.59**	**48.24**

	Learning with 7 KITTI + SIFTFlow								
	7 KITTI			SIFTFlow			Précision sur les deux		
Méthode	Glob.	Moy.	IoU	Glob.	Moy.	IoU	Glob.	Moy.	IoU
Baseline	77.35	54.51	41.99	69.79	24.95	19.02	76.72	45.18	34.73
AJ	79.86	57.05	44.94	70.67	**28.65**	**21.44**	79.1	48.08	37.52
AJ+RG	80.47	57.10	45.31	70.60	28.56	21.39	79.65	48.08	**37.76**
AJ+EB	78.93	53.94	42.63	**72.15**	27.24	21.12	78.37	45.51	35.84
AJ+RG+EB	80.44	**58.12**	**45.56**	70.88	27.11	20.4	79.65	**48.33**	37.62
AJPC+RG+EB	**82.07**	54.68	44.87	70.89	23.11	17.76	**81.15**	44.71	36.31

Tab. 5.1.: Résultats des précisions par pixels (Globale), par classe (Moyenne) ainsi que l'IoU obtenus sur les 7 sous-parties de la base de données KITTI, ainsi que des différentes combinaisons des bases de données : SIFTFlow + Stanford ; 7 KITTI + Stanford ; 7 KITTI + SIFTFlow. Le tableau présente les résultats obtenus avec les différentes stratégies d'apprentissage vues dans la partie 5.3, à savoir : Baseline (voir la figure 5.1) ; AJ= Apprentissage Joint (voir la figure 5.2) ; EB=Equilibre des bases ; RG=Renversement du gradient ; AJPC=Apprentissage joint avec partage du contexte (voir la figure 5.3). Les meilleurs résultats sont affichés en gras. Pour des questions de clarté, nous ne donnons que les résultats globaux des 7 sous-parties de la base KITTI. Néanmoins, le détail de ces tableaux est disponible dans l'annexe A.

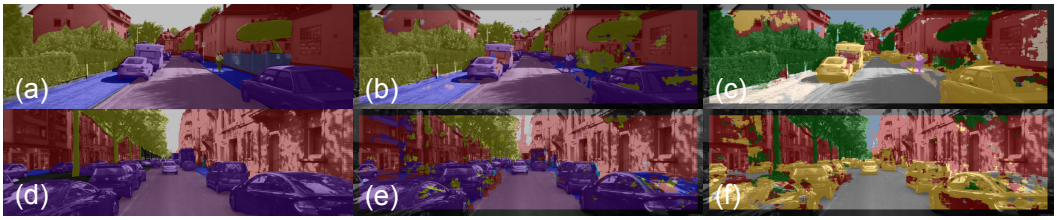


Fig. 5.8.: Exemple de résultats de segmentation sémantique obtenue avec la stratégie AJPC (voir figure 5.1). **(a)** est la vérité terrain venant de la sous-partie annotée par l'équipe de Ros *et al*. **(b)** est le résultat obtenu avec notre stratégie pour l'ensemble d'étiquette de Ros *et al*. et **(c)** est le résultat obtenu simultanément sur l'ensemble d'étiquettes utilisé par l'équipe de Kundu *et al*. De façon identique, **(d)** est un exemple de vérité terrain provenant de la partie annotée par l'équipe de He *et al*., **(e)** est le résultat obtenu par notre stratégie sur l'ensemble d'étiquettes de He *et al*. et **(e)** est le résultat obtenu simultanément sur l'ensemble d'étiquettes de Xu *et al*.

par classes (précision moyenne) ainsi que l'IoU. La précision par pixels (précision globale) correspond aux nombre de pixels correctement classés (True positif) sur le nombre total de pixels. La précision par classe (précision moyenne) correspond à la moyenne des précisions obtenues indépendamment sur chaque classe (nombre de pixels correctement classés). L'IoU (Intersection over Union), aussi appelé Jaccard Index, correspond au ratio $TP/(TP + FP + FN)$ où TP est le nombre de vrais positifs (True Positive), et FP et FN sont respectivement les faux positif et faux négatifs. Ces mesures sont détaillées dans la section 4.1.2. Il est important de noter que la dernière colonne du tableau 5.1 ("précision sur les deux") donne les précisions obtenues sur les deux bases de données considérées et donc cela prend en compte le nombre relatif d'exemples d'entrainement dans chaque base de données. Cette valeur n'est donc pas égale à la moyenne des deux précisions obtenues sur les deux bases.

Baseline

La première stratégie implémentée est la stratégie de base qui consiste à apprendre un réseau différent pour chacune des bases de données. Cette stratégie, illustrée dans la figure 5.1, est expliquée dans la section 5.3.1. C'est notre stratégie de référence et les résultats sont présentés dans la ligne appelée Baseline.

Le tableau 5.2 présente les résultats indiqués par les auteurs sur les sous-parties de la base de données KITTI considérés. Même si l'on indique les résultats de notre Baseline à titre indicatif, ces derniers ne sont pas explicitement comparables avec ceux de l'état de l'art. En effet, les méthodes utilisées par les différents auteurs n'utilisent pas les mêmes données que nous. Certaines étiquettes (difficiles) ne sont pas présente dans leurs études et des techniques de pré-traitement (tel que des super-pixels) ou post-traitement (tel que des CRF) ont été utilisées. De la même manière,

	État de l'art		Notre baseline	
	Globale	Moyenne	Globale	Moyenne
He *et al.* [HU13]	92.77	68.65	76.11	59.66
Kundu *et al.* [Kun+14]	97.20	✗	71.36	53.58
Ladicky *et al.* [Lad+14]	82.4	72.2	73.48	42.81
Ros *et al.* [Ros+15]	51.2	61.6	76.80	48.46
Sengupta *et al.* [Sen+13]	90.9	92.10	73.25	64.25
Xu *et al.* [Xu+13]	✗	61.6	86.07	77.30
Zhang *et al.* [Zha+15]	89.3	65.4	77.30	49.53

Tab. 5.2.: Etat de l'art (au moment de nos travaux sur ces aspects) des différentes sous-parties de la base de données KITTI. Ces résultats ne sont pas directement comparables aux nôtres puisqu'ils utilisent plus d'information et des techniques de prétraitement et post-traitement. X indique que les valeurs n'ont pas été rapportées par les auteurs.

	État de l'art			Notre baseline		
	Globale	Moyenne	IoU	Globale	Moyenne	IoU
Stanford [Gou+09]	82.3	79.1	64.5	74.80	64.35	51.41
SiftFlow [Liu+11]	80.9	39.1	30.8	69.79	24.95	19.02

Tab. 5.3.: Etat de l'art (au moment de nos travaux) des bases de données Stanford et SiftFlow en comparaison avec notre baseline.

les résultats du tableau 5.3, qui montre les résultats (au moment de nos travaux) sur les bases de données Stanford et SiftFlow, ne sont pas non plus explicitement comparables avec notre baseline puisque, pour la base de données Stanford, les étiquettes utilisées n'incluent pas la classe *Inconnue*.

Néanmoins, dans nos travaux nous voulions montrer l'intérêt d'un apprentissage joint par rapport à un apprentissage classique. C'est pourquoi nous pensons que notre baseline est un point de comparaison pertinent. De plus, notre fonction de perte sélective utilisée est compatible avec tous les pré-traitements, post-traitements et autre outils utilisés par les méthodes de l'état de l'art.

Apprentissage joint (JT)

La seconde stratégie étudiée (illustrée dans la figure 5.2) est l'*Apprentissage Joint* (AJ) qui consiste à entrainer un unique réseau avec l'ensemble des bases de données en utilisant notre fonction de perte sélective détaillée dans la section 5.3.2. On peut d'ores et déjà observer que cette stratégie donne de meilleurs résultats (en comparaison avec la stratégie *Baseline*) sur la combinaison de toutes les bases de données. Par exemple, on peut voir dans le tableau 5.1, que l'entrainement avec

toutes les sous-parties de la base KITTI donne, en moyenne, une amélioration de $+3.36$ sur la précision globale, $+4.11$ sur la précision moyenne et $+4.22$ sur l'IoU. Ces résultats montrent que cette stratégie permet d'augmenter efficacement le nombre de données utilisées pour l'entraînement et ce même si les ensembles d'étiquettes sont différents. Cette augmentation de données par l'utilisation de plusieurs bases permet une meilleure généralisation des caractéristiques extraites ainsi qu'une meilleure précision dans les prédictions faites par le réseau. Une seule exception survient quand la base de données Stanford est entraînée de façon jointe aux sept sous-parties de la base de données KITTI. Dans ce cas, toutes les précisions obtenues sur la base Stanford chutent d'environ un pourcent lorsque l'on utilise l'approche jointe. Nous faisons l'hypothèse que cela est dû à la différence trop importante entre les images des sous-parties de KITTI et celles de la base de données Stanford, ainsi qu'à l'écart trop important entre le nombre d'images des bases de données. Par conséquent, le réseau, n'extrait pas des caractéristiques communes aux deux bases de données, mais privilégie les caractéristiques utiles pour les sous-ensembles de la base KITTI.

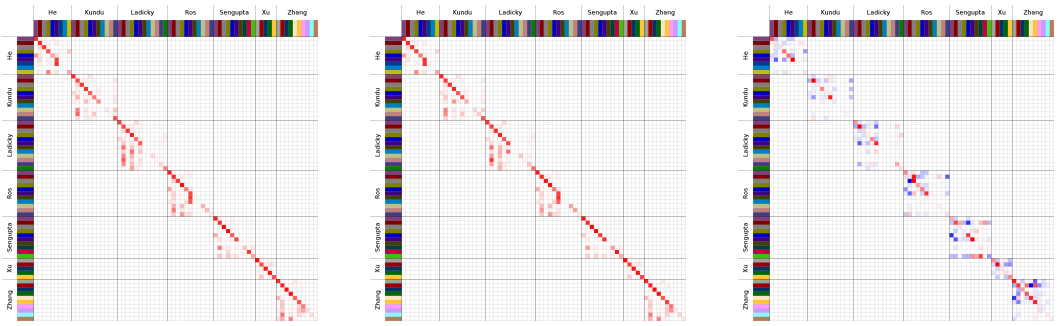
De plus, dans un souci d'exhaustivité, nous avons aussi étudié une approche plus classique qui consiste à pré-entrainer un réseau pour ensuite le spécialiser sur une autre base de données (technique du finetuning vue dans la partie 2.3.2). Les résultats, reportés dans le tableau 5.4, montrent que cette approche améliore la précision obtenue par rapport à la méthode de référence, mais reste moins bonne qu'avec l'utilisation de notre fonction de perte sélective utilisée dans notre apprentissage joint.

Évaluation (spécialisation) sur la base SIFTFlow				
		Globale	Moyenne	IoU
Pré-entrainé sur	Stanford	70.86	23.52	18.05
	7 KITTI	70.33	27.07	19.87
Notre méthode	No Fusion	69.79	24.95	19.02
	Best method	71.11	29.01	22.25
Évaluation (spécialisation) sur la base Stanford				
		Globale	Moyenne	IoU
Pré-entrainé sur	SIFTFlow	74.50	65.65	52.52
	7 KITTI	73.57	63.82	50.81
Notre méthode	No Fusion	74.80	64.35	51.41
	Best method	76.04	67.71	54.05

Tab. 5.4.: Résultats obtenus sur les bases de données SiftFlow (table du haut) et Stanford (table du bas) en pré-entrainant un réseau sur une des bases de données et en le spécialisant ensuite sur la deuxième base. Ces résultats sont comparés à l'*apprentissage joint* utilisant notre fonction de perte sélective, où un réseau est entrainé en même temps sur les deux bases de données.

Apprentissage joint avec partage de contexte (AJPC)

Bien que l'*apprentissage joint* donne de bons résultats, ce dernier n'exploite pas complètement les corrélations possibles entre les étiquettes des différents ensembles d'étiquettes. Notre approche *apprentissage joint avec partage de contexte* (AJPC) illustrée dans la figure 5.3 et expliquée dans la partie 5.3.3 permet une meilleure prise en compte de ces corrélations. Les résultats montrent que cette approche améliore les précisions pour la plupart des sous-parties de la base de données KITTI (voir les tableaux en annexe A) quand ces dernières sont utilisées indépendamment, mais donne des résultats comparables quand le réseau est appris de façon jointe avec d'autres bases de données. Cela s'explique par une plus grande corrélation des étiquettes entre les sous-parties de la base KITTI qui n'est pas (ou moins) présente entre les étiquettes des bases SiftFlow et Stanford.



(a) Apprentissage joint **(b)** Apprentissage joint avec partage du contexte **(c)** Différences

Fig. 5.9.: Matrices de confusion obtenues avec nos stratégies d'apprentissage joint 5.9a et d'apprentissage joint avec partage du contexte 5.9b ainsi que la différence entre les deux 5.9c. Les matrices sont normalisées par lignes et l'intensité des couleurs montre les valeurs les plus élevées. Les valeurs en bleue sont négatives et en rouge positives. Les 68 lignes/colonnes correspondent aux 68 étiquettes détaillées dans le tableau 5.6.

Nous avons aussi étudié les matrices de confusion (voir partie 4.1.2) obtenues avec les approches d'apprentissage joint puis avec l'apprentissage joint avec partage de contexte. En faisant la différence entre les deux matrices on peut observer les différences de précision entre les deux méthodes. La figure 5.9 montre les matrices de confusion des deux approches ainsi que la matrice de différence. Les valeurs en bleu correspondent à des valeurs négatives et les valeurs en rouges à des valeurs positives. L'intensité de la couleur correspond à la valeur absolue des différences. Idéalement les valeurs en bleu doivent être maximales en dehors de la diagonale et les valeurs en rouge maximales dans la diagonale. Cela indiquerait que les erreurs faites par la première approche (apprentissage joint) ont été corrigées par la seconde approche (apprentissage joint avec partage du contexte). On peut voir, sur la matrice de différence, qu'une majorité des erreurs a bien été corrigée.

Renversement du gradient et équilibrage des bases de données

Comme nous l'avons vu dans la section 5.3, nous savons qu'il peut être utile de prendre en compte les différences entre les distributions d'entrées des bases de données (différences entre les images). Pour cela nous avons combiné notre apprentissage joint avec la technique de renversement du gradient de Ganin et Lempitsky [GL15] expliquée dans la partie 5.3.4. Cela a pour objectif de forcer le réseau à apprendre des caractéristiques communes entre les différentes bases de données étudiées. Par contre, un renversement de gradient sur l'apprentissage joint des sous-parties de la base de données KITTI n'a pas de sens puisque les données des sous-parties proviennent d'une même distribution.

Les résultats, donnés dans le tableau 5.1, montrent qu'ajouter ce renversement de gradient n'améliore pas systématiquement les performances. En effet, les résultats de l'apprentissage joint entre les bases SiftFlow et Stanford sont moins bons quand on utilise le renversement de gradient que sans son utilisation. Une perte de performance a tendance à se produire quand il existe une grande différence de données entre les deux bases apprises conjointement. Cela nous a poussé à équilibrer les bases en pondérant le gradient des erreurs en fonction du nombre d'images présentes dans les bases de données. Les résultats montrent l'importance de cet équilibrage qui est utile même pour les sous-parties de la base KITTI. Les meilleurs résultats étant obtenus quand l'entrainement joint est combiné à l'équilibrage des bases de données et au renversement du gradient.

5.5 Application à l'estimation de pose de la main

Afin de démontrer la généralité de notre approche, nous l'avons appliquée à un tout autre problème, un problème de régression : l'estimation de pose de la main.

5.5.1 Description et bases de données

L'estimation de pose de la main est une tâche de régression qui consiste, étant donnée une image d'entrée, d'estimer la position dans l'image des différentes parties d'une main (voir figure 5.10)

Plusieurs bases de données pour l'estimation de pose de la main existent et ces dernières diffèrent généralement sur plusieurs points. Les parties de la main, dont il faut estimer les positions, peuvent être différentes ainsi que leur nombre. Les tailles (et résolutions) des images peuvent aussi varier ainsi que les mains capturées (droite

ou gauche). Pour notre étude, nous nous sommes intéressés à deux des principales bases de données pour cette tâche : la NYU Hand Pose Dataset [Tom+14] et la base de données ICVL Hand Posture [Tan+13].

La base de données NYU est composée de 180 mille images provenant de 10 personnes différentes. Elle est utilisée pour prédire la position de 14 parties différentes d'une main. La base ICVL, quant à elle, contient 70 mille images provenant qu'une seule personne, capturée à partir de trois points de vue différents. Elle est annotée avec 16 points de la main différents, dont il faut estimer les positions. Les points correspondent aux centres des différents segments d'une main et ne sont donc pas, à proprement parlé, des articulations. Pour les deux bases de données, les images utilisées sont des images dites de profondeur, c'est-à-dire, qu'elles ne possèdent pas trois canaux de couleurs (généralement R, G, B) mais un unique canal noté D, dont la valeur correspond à la distance de l'objet (ici la main) observé.

Notre fonction de perte sélective pour l'apprentissage joint des deux bases est donc adaptée puisque le nombre ainsi que les positions (et donc la sémantique) des jointures diffèrent d'une base à l'autre.

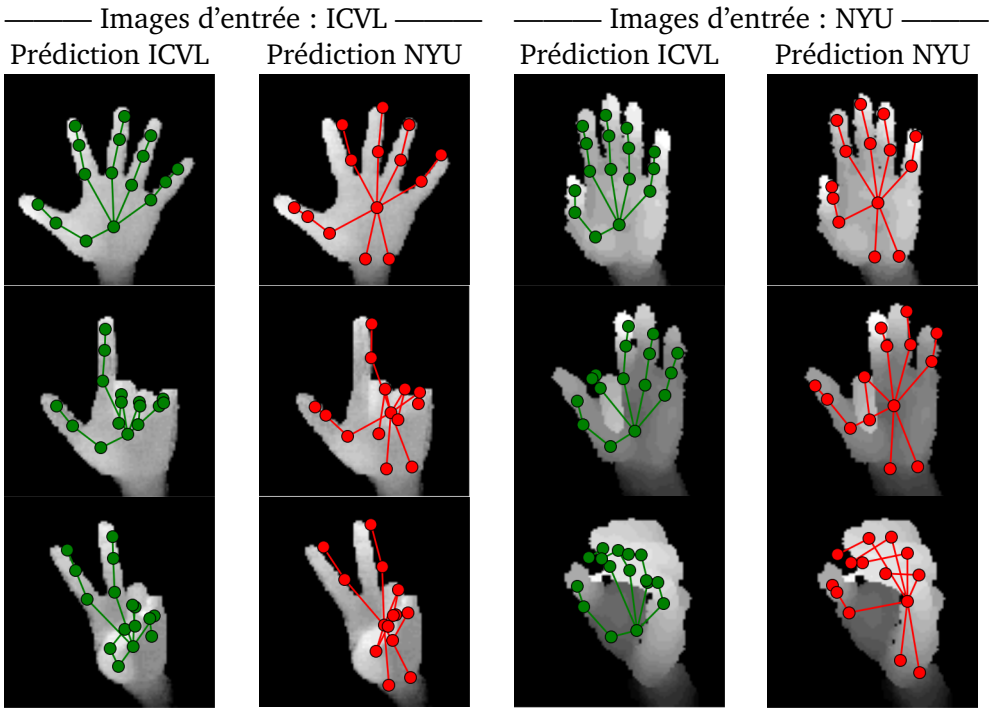


Fig. 5.10.: Résultats de la stratégie *apprentissage joint avec partage de contexte* appliquée à la tâche d'estimation de la pose de la main. Les images des deux colonnes de gauche sont tirées de la base de données ICVL [Tan+13] et les images des deux colonnes de droites proviennent de la base de données NYU [Tom+14].

	Baseline		Apprentissage Joint		AJPC	
	2D, px	3D, mm	2D, px	3D, mm	2D, px	3D, mm
NYU	8.49	17.75	8.16(-3.81%)	17.11(-3.63%)	8.02(-5.47%)	16.80(-5.37%)
ICVL	4.76	9.53	4.62(-2.82%)	9.38(-2.57%)	4.56(-4.13%)	9.16(-3.92%)
Total	N/A	16.42	N/A	15.85(-3.43%)	N/A	**15.56(-5.23%)**

Tab. 5.5.: Résultats de la régression pour la tâche d'estimation de pose de la main, avec un apprentissage joint utilisé sur les deux bases de données : NYU [Tom+14] et ICVL [Tan+13]. Les erreurs reportées correspondent aux distances en pixels pour l'erreur en 2D et en millimètre pour l'erreur en 3D. Plus les valeurs sont petites plus les résultats sont bons, par conséquent, un pourcentage négatif indique une amélioration.

5.5.2 Configurations et résultats

Inspiré par les travaux de Neverova *et al.* [Nev+15a], nous avons utilisé un réseau de neurones convolutif à trois couches de convolution, suivies par trois couches entièrement connectées avec 1200 unités cachées chacune. Un opérateur de dropout avec un facteur d'annulation de 10 % est ajouté à la suite des couches entièrement connectées. Le réseau est entrainé par descente de gradient avec un mini-batch de 100 exemples, composé à chaque fois de 50 images provenant de la base NYU et 50 images de la base ICVL. L'apprentissage joint avec partage du contexte correspond à une couche entièrement connectée supplémentaire, ajoutée à la sortie du réseau et composée de 300 unités cachées.

Les résultats, donnés dans le tableau 5.5, montrent que la fonction de perte sélective est aussi utile pour les problèmes de régressions. Sans partage de contexte, la base la plus complexe (NYU) bénéficie de plus de données d'entrainement ce qui lui permet d'améliorer ses résultats. En ajoutant le partage de contexte, les deux bases bénéficient des améliorations (puisque les positions des jointures sont extrêmement corrélées) réduisant davantage les erreurs faites par le réseau.

Par contre, l'ajout du reversement de gradient n'aide pas. Cela est dû à la nature très similaire des images d'entrées (image de profondeur) qui permettent au réseau d'apprendre des caractéristiques identiques pour les deux bases de données.

Le tableau 5.6 compare les résultats que nous obtenons à ceux de l'état de l'art. Nous pouvons constater que l'erreur 3D de notre méthode est compétitive et donne des performances proches de la méthode [Nev+15a] qui donne les meilleurs résultats, alors que cette dernière utilise des données synthétiques afin d'augmenter la taille de l'ensemble d'entraînement. Les bons résultats de notre méthode s'expliquent facilement avec l'augmentation du nombre de données d'entrainement.

Method	NYU (3D, mm)	ICVL (3D, mm)
Baseline	17.76	9.54
Apprentissage joint	17.11	9.39
AJPC	16.80	9.16
Neverova et al. [Nev+15a]	14.94	-
DeepPrior, Oberweger et al. [Obe+15]	$19.8^{\dagger}$	$9.6^{\dagger}$
Tompson et al. [Tom+14]	$21.0^{\dagger\star}$	-
Multi-Scale, Oberweger et al. [Obe+15]	$27.5^{\dagger}$	$11.7^{\dagger}$
Deep, Oberweger et al. [Obe+15]	$30.5^{\dagger}$	$11.8^{\dagger}$
Shallow, Oberweger et al. [Obe+15]	$34.5^{\dagger}$	$11.7^{\dagger}$
Tang et al. [Tan+13]	-	$12.6^{\dagger\star}$

Tab. 5.6.: Comparaison de nos résultats (en haut) sur la tâche d'estimation de pose de la main avec les méthodes de l'état de l'art sur les bases de données NYU et ICVL. Les valeurs marquées avec le symbole † sont estimées par rapport à une courbe de donnée, car les auteurs de fournissent pas les valeurs numériques. Le symbole * indique que la valeur donnée est celle reportée dans le papier [Obe+15].

La figure 5.10 illustre quelques résultats obtenus par notre méthode sur des images provenant des deux bases de données. On peut distinguer que les jointures utilisées ont une cardinalité et des positions différentes.

5.6 Conclusion sur l'étude des corrélations entre étiquettes

Avec notre stratégie d'apprentissage joint avec partage du contexte, et avec nos études sur la segmentation sémantique et l'estimation de pose de la main, nous avons démontré qu'il est utile de tirer parti des corrélations existantes entre les différents ensembles d'étiquettes.

Nous pouvons aller encore plus loin pour la tâche de classification puisque nous pouvons récupérer les corrélations apprises par le réseau. Pour cela, nous avons construit une matrice de corrélation contenant la totalité des sous-parties de la base de données KITTI. Les lignes de la matrice (montrée dans la figure 5.11) représentent les étiquettes cibles tandis que les colonnes de la matrice correspondent aux prédictions (probabilités) produites par le réseau de neurone. La diagonale de la matrice correspond donc aux étiquettes cibles correctement annotées. Les valeurs élevées, qui ne sont pas sur la diagonale, sont des probabilités importantes prédites par le réseau et qui sont, par conséquent, corrélées avec la classe cible (classe de la ligne correspondante). Chaque bloc de la matrice correspond à un ensemble d'étiquettes différents. Or on peut noter que dans chaque bloc une diagonale est visible. Cela veut dire que les étiquettes de ces diagonales sont toutes corrélées avec les étiquettes de la diagonale principale. Plus concrètement, les cinq premières

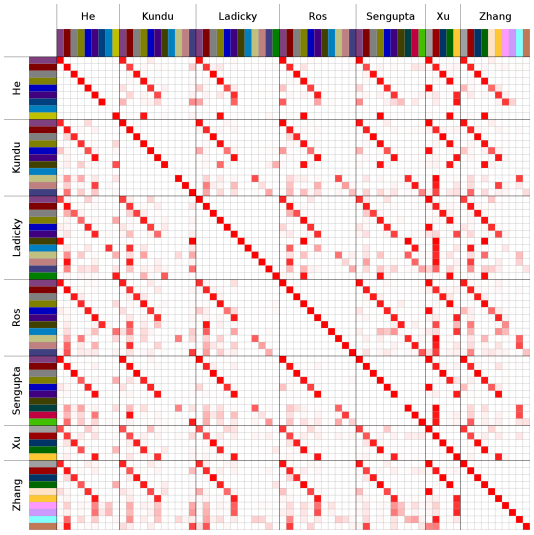


Fig. 5.11.: Matrice de corrélation empirique des étiquettes entre les sous-parties de la base de données KITTI. Chaque ligne correspond à la moyenne des prédictions produites par le réseau pour une étiquette cible donnée. Les 68 lignes/colonnes correspondent aux 68 étiquettes détaillées dans la figure 5.6 (avec les couleurs correspondantes). Une cellule plus foncée indique une probabilité plus élevée. Les cellules qui ne sont pas sur la diagonale correspondent à des classes hautement corrélées avec la classe cible (puisque le réseau leur prédit une grande probabilité).

étiquettes de chacun des ensembles d'étiquettes sont toutes corrélées. Ce résultat est attendu puisque les cinq premières étiquettes ont toutes la même sémantique, à savoir route, bâtiments, ciel, végétation et voiture. Une seconde observation est que la matrice n'est pas symétrique. C'est aussi un résultat attendu puisqu'il arrive que plusieurs étiquettes d'un ensemble soient inclus dans une étiquette définie par un autre. Par exemple, les étiquettes *Bâtiment*, *Poteaux*, *Panneaux* et *Barrière* de l'ensemble d'étiquettes de Sengupta *et al.* sont toutes hautement corrélées avec l'étiquette *Infrastructure* de l'ensemble de Xu *et al.*, indiquant que ces dernières sont probablement incluses dans l'étiquette *Infrastructures* de Xu *et al.* Par contre, l'étiquette *Infrastructure* de Xu *et al.* est très corrélée à l'étiquette *Bâtiment* de Sengputa *et al.* Cela est encore une fois logique puisque le *Bâtiment* de Sengupta *et al.* fait aussi partie de l'*Infrastructure* de Xu *et al.* et, étant beaucoup plus présent que les autres étiquettes, la probabilité qu'un exemple annoté *Infrastructure* chez Xu *et al.* appartiennent à l'étiquette *Bâtiment* de Sengupta *et al.* est plus élevée.

Cette matrice confirme les attentes discutées dans la partie 5.4.2. Elle montre que notre méthode peut aussi être utilisée afin de découvrir de manière automatique des corrélations entre différents ensembles d'étiquettes.

6 GridNet, une architecture spécialisée pour la segmentation sémantique

6.1 Introduction

Dans ce chapitre nous présentons GridNet, un nouveau type d'architecture de réseaux de neurones convolutifs, conçue pour la segmentation sémantique d'images. Les réseaux de neurones convolutifs classiques sont construits avec un seul chemin allant de l'entrée (l'image) jusqu'à la sortie (la carte de prédiction). Ce chemin est généralement composé d'opérations de sous-échantillonnage (généralement du type maximum pooling (voir chapitre 2.1.4)) permettant de réduire la taille des cartes de caractéristiques ("*feature maps*" en anglais) et d'augmenter le champ de vision du réseau pour la prédiction finale (voir chapitre 4.2.1). Néanmoins, pour la segmentation sémantique d'images, où la tâche consiste à donner une prédiction pour chaque pixel d'une image d'entrée, la réduction des cartes de caractéristiques pose un problème, car elle entraine une perte de résolution se répercutant sur la qualité de la prédiction de sortie. Les réseaux basés sur une approche conv-deconv cherchent à corriger cette perte de résolution en ajoutant au réseau une partie dont le rôle est de reconstruire les cartes de caractéristiques. Nous abordons une approche légèrement différente puisque nous ne reconstruisons pas les cartes de caractéristiques afin de corriger un problème de perte de résolutions, mais nous lions à la fois des prédictions hautes résolutions et des niveaux d'abstractions élevés. Pour cela, nous avons développé GridNet, une architecture de réseau de neurones convolutifs en forme de grille 2D, permettant une interconnexion de plusieurs "chemins", travaillant sur des cartes de caractéristiques à différentes résolutions. Dans ce chapitre, nous montrerons que cette architecture généralise plusieurs architectures connues, tels que les réseaux conv-deconv [Lon+15 ; Noh+15], les réseaux résiduels [He+16a ; Wu+16] ou encore les U-Net [Ron+15] que nous avons déjà mentionné dans la partie 4.3. Notre GridNet est entraîné sur la base de données Cityscapes [Cor+16] à partir d'une initialisation aléatoire (sans pré-entraînement des poids du réseau) et obtient des résultats compétitifs par rapport à d'autres réseaux, moins généraux, pré-entraînés sur la base de données ImageNet afin d'améliorer l'initialisation de leurs poids.

6.2 Réseaux entièrement convolutifs

Les architectures traditionnelles de réseaux de neurones convolutifs sont composées d'une séquence d'opérateurs de convolutions suivie de fonctions non linéaires et entremêlées d'opérateurs de sous-échantillonnage. Ces opérateurs de sous-échantillonnage sont responsables de la diminution de la taille des cartes de caractéristiques, et permettent ainsi l'augmentation du champ réceptif des réseaux. Pour la tâche de classification, qui consiste, étant donnée une image, de donner la classe sémantique de l'objet présent sur cette dernière, la réduction de la taille des cartes de caractéristiques est nécessaire. Elle permet d'augmenter le champ de vision du réseau, de limiter l'espace mémoire utilisé et de réduire la taille des couches entièrement connectées généralement utilisées en sortie du réseau (voir détails dans la section 4.2.1). Néanmoins, dans le cas de la segmentation sémantique où une prédiction en haute résolution est attendue, l'opérateur de sous-échantillonnage est néfaste puisqu'il réduit la résolution des cartes de caractéristiques et donc la résolution de la prédiction finale du réseau.

6.2.1 Réseaux Convolutif-Déconvolutif

Nous avons déjà vu dans la partie 4.3 qu'il existe des réseaux spécialisés pour la segmentation sémantique permettant une prédiction en haute résolution. Une des techniques les plus populaires aujourd'hui est celle présentée par Long *et al*. [Lon+15] qui montrent comment transformer les couches entièrement connectées d'un réseau en couches de convolutions permettant d'obtenir un réseau dit "entièrement convolutif" (*Fully convolutions*). Noh *et al*. [Noh+15] ont étendu cette idée en utilisant un opérateur appelé *maximum unpooling* permettant une communication faible (sous la forme des indices des valeurs maximales de la partie convolutive) entre les parties convolutive et déconvolutive (voir détails dans la section 6.2.1).

Ronneberger *et al*. [Ron+15] vont encore plus loin avec les réseaux appelés U-Net, où ils concatènent les cartes de caractéristiques obtenues dans la partie convolutive avec les cartes de caractéristiques de la partie déconvolutive afin de permettre une meilleure reconstruction des images segmentées. Finalement, Lin *et al*. [Lin+16b] utilisent la même idée que pour les U-Net mais au lieu de concaténer directement les cartes de caractéristiques, ils utilisent une unité appelée *refineNet* (voir partie 4.3.4).

Les réseaux présentés ci-dessus (et détaillés dans la 4.3) sont basés sur l'idée que le sous-échantillonnage est important pour augmenter le champ de vision du réseau et essaient de corriger la perte de résolution induite par ce dernier en ajoutant un réseau

de déconvolution. Notre GridNet, quant à lui, est composé de plusieurs chemins travaillant à différentes tailles de cartes de caractéristiques. Des opérateurs de sous-échantillonnage et de sur-échantillonnage sont utilisés pour connecter entre eux les différents chemins. Contrairement aux réseaux décrits ci-dessus, les opérateurs de sur-échantillonnage ne sont pas utilisés pour corriger un effet secondaire dû au sous-échantillonnage, mais ils sont utilisés afin de permettre une prise de décision multi-échelle du réseau. Dans des travaux récents, Newell *et al*. [New+16] utilisent plusieurs U-Net à la suite et montrent que plusieurs étapes successives de sous-échantillonnage et de sur-échantillonnage permettent d'améliorer les performances du réseau. Cette idée est améliorée dans GridNet en ajoutant des connexions fortes entre les différents chemins du réseau.

6.2.2 ResNet et convolutions dilatées

Yu *et al*. [YK16] ont montré que, pour la tâche de segmentation sémantique, les opérateurs de sous-échantillonnage ne sont pas nécessaires. Par conséquent, afin de corriger l'effet secondaire dû au sous-échantillonnage, ils les ont supprimés, permettant ainsi de conserver la résolution des cartes de caractéristiques. Sans sous-échantillonnage et avec des opérations de convolutions classiques, le champ de vision est très petit. Ils corrigent ce problème en utilisant des convolutions dilatées, permettant ainsi d'élargir le champ de vision du réseau. Contrairement aux convolutions classiques, où le masque de convolution est appliqué sur des pixels voisins, les convolutions dilatées possèdent un paramètre de dilatation, utilisé pour appliquer le masque sur des pixels de plus en plus éloignés (voir détails dans la partie 4.3.5).

Dans leurs travaux, Wu *et al*. [Wu+16] ont montré comment adapter le populaire ResNet [He+16a], pré-entraîné sur la base de données ImageNet, à la tâche de segmentation sémantique en utilisant ces convolutions dilatées (voir partie 4.3.5). Néanmoins leur réseau fournit des sorties à $1/8$ de la taille de l'image d'entrée. Par conséquent, ils utilisent une interpolation linéaire des prédictions afin de retrouver la résolution d'origine. Dans leurs travaux, Zhao *et al*. [Zha+16] remplacent l'interpolation linéaire par un module appelé Pyramique Pooling afin d'augmenter la taille des cartes de caractéristiques tout en permettant une décision multi-échelle. Dans GridNet, cette notion multi-échelle est présente directement dans le réseau avec l'utilisation des différents chemins interconnectés et travaillant à différentes résolutions.

Dans leurs travaux, He *et al*. [He+16c] étudient l'importance des unités résiduelles et donnent des résultats détaillés sur les différentes stratégies possibles dans l'utilisation de connexions résiduelles (est-ce que la normalisation par batch doit avoir lieu avant

ou après les convolutions, est-ce que l'opérateur de non-linéarité (ReLU) doit avoir lieu avant ou après l'addition des valeurs des cartes de caractéristiques ? etc.). Comme GridNet utilise aussi des connexions résiduelles, nous avons utilisé leurs travaux afin de choisir la meilleure configuration possible.

Avec leur Full Resolution Residual Network (FRRN) [Poh+16], Pohlen *et al.* combinent un réseau conv-déconv avec un réseau résiduel (voir détails dans la partie 4.3.3). Les deux réseaux sont composés de deux chemins (un résiduel, un conv-deconv) partageant les mêmes cartes de caractéristiques (voir figure 4.14). Grâce à l'utilisation de plusieurs chemins et de connexions résiduelles, GridNet peut être vu comme une généralisation de FRRN.

6.2.3 Architecture 2D

L'idée de réseaux composés de plusieurs chemins n'est pas nouvelle [Zho+15 ; Hua+17 ; SV16]. Zhou *et al.* ont étudié la tâche de reconnaissance de visages avec des réseaux de neurones convolutifs interconnectés. Une image d'entrée est utilisée à différentes résolutions par plusieurs réseaux convolutifs. Les cartes de caractéristiques sont partagées et concaténées pour être utilisées par les différents réseaux. Huand *et al.* [Hua+17] utilisent la même architecture, mais l'entraînent de façon à ce qu'elle soit adaptable aux ressources mémoire disponibles et ce même en phase de test, sans avoir à ré-entraîner le réseau.

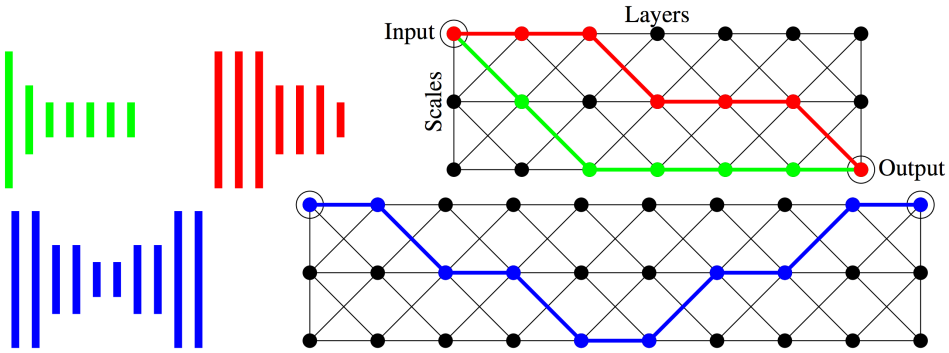


Fig. 6.1.: Schémas des Neural Fabrics de Saxena *et al.* [SV16]. Leurs structures sont très similaires à notre GridNet et permettent de généraliser des réseaux traditionnels, tels que des réseaux convolutifs à 7 couches (en rouge et vert) et des réseaux conv-déconv à 10 couches (en bleu).

Récemment, Saxena *et al.* ont présenté Convolutional Neural Fabrics [SV16], une architecture dont la structure en forme de grille est très similaire à notre GridNet et qui utilise aussi plusieurs chemins à différentes échelles afin de produire une prédiction (voir figure 6.1). Néanmoins, les deux architectures n'ont pas été développées avec les mêmes motivations.

Les Neurals Fabrics ont été développées dans l'objectif d'être très génériques afin de permettre au réseau de choisir la meilleure architecture parmi l'ensemble des chemins disponibles, mais aussi de permettre de combiner toutes architectures. Cette motivation a poussée l'utilisation de nombreux opérateurs de convolutions successifs, correspondant à de nombreuses colonnes dans la grille. Or un nombre élevé de colonne implique une consommation mémoire plus importante (voir partie 6.3.3), limitant l'utilisation des Neurals Fabrics à des tâches de relativement petite taille (en termes de taille d'image et de nombre d'exemples).

Notre GridNet quant à lui, a été développé dans le but d'utiliser des images de grandes résolutions. Nous avons pensé l'architecture comme un réseau conv-déconv échangeant des informations à la fois haute résolution mais aussi avec un fort niveau d'abstraction entre les parties convolutives et déconvolutives. C'est pourquoi notre architecture possède une partie convolutive suivie une partie déconvolutive (voir schéma 6.2) au lieu d'un enchainement d'opérateurs conv-deconv comme dans les Neurals Fabrics (nous avons tout de même essayé une configuration plus proche des Neurals Fabrics enchainant opérations de convolutions et de déconvolutions (voir tableau 6.3)). Nous avons aussi voulu améliorer les échanges et c'est pourquoi nous avons introduit des connexions résiduelles. Ces connexions ont démontré une grande importance dans l'entraînement du réseau. Finalement, afin de permettre une meilleure utilisation de la grille, nous avons développé une méthode forçant le réseau à utiliser tous les chemins de la grille plutôt que de le laisser privilégier un chemin. Cette technique, appelée *dropout par blocs* ("*Total Dropout*" en anglais) a montré empiriquement son efficacité.

6.3 GridNet

Dans cette partie, nous présentons GridNet, notre architecture sous forme de graphe, développée spécialement pour la segmentation sémantique et avec pour objectif de permettre au réseau un fort niveau d'abstraction, tout en conservant une haute résolution dans les prédictions.

L'idée principale de ce concept de grille est de permettre à l'information de circuler librement dans le graphe de calcul. Les chemins dans la grille sont nombreux et non fixés, permettant ainsi au réseau de s'adapter aux données d'entrées. Dans les réseaux traditionnels, les unités de sous-échantillonnage et de sur-échantillonnage sont importantes pour augmenter le champ réceptif du réseau. Nous utilisons donc ces dernières afin de connecter les différentes lignes horizontales de la grille. Par ailleurs, ces lignes utilisent des unités résiduelles, afin de permettre une conservation de la résolution des cartes de caractéristiques et une meilleure circulation de l'information

(et du gradient). Les connexions entre les différents chemins horizontaux permettent donc de récupérer une information à la fois riche en termes de contexte (grâce à un champ de vision large présent les chemins les plus bas du réseau) mais aussi précise (grâce aux premières lignes du réseau qui conserve la résolution d'origine).

6.3.1 Description de l'architecture

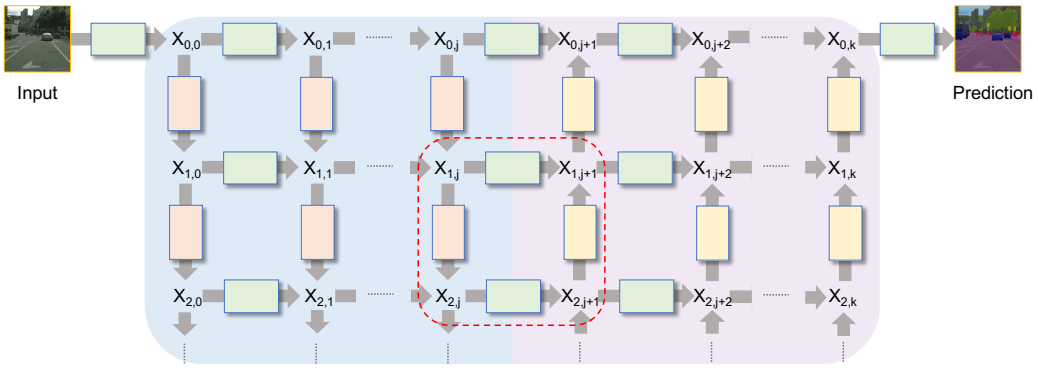


Fig. 6.2.: Schéma de notre réseau GridNet : chaque bloc vert est une unité résiduelle qui ne change ni les résolutions ni le nombre de cartes de caractéristiques d'entrée. Les blocs rouges sont des unités convolutives avec réduction de résolution (sous-échantillonnage) qui doublent le nombre de cartes de caractéristiques. Les blocs jaunes sont des unités déconvolutives qui augmentent la résolution des cartes de caractéristiques et divisent par deux leurs nombres. Un schéma détaillé du carré en pointillé rouge présentant la composition de chacun des blocs est donné dans la figure 6.3

GridNet est un réseau sous forme de graphe de calculs à deux dimensions (voir la figure 6.2). Chaque carte de caractéristiques $X_{i,j}$ est indexée par sa ligne i et sa colonne j indiquant sa position dans la grille. Les cartes sont connectées par des couches de calculs. L'information entre dans la grille par un premier bloc de calcul à la ligne d'indice 0 et sort à travers un dernier bloc à l'autre extrémité de la ligne 0. Entre ces deux points, l'information peut circuler par différents chemins, grâce à des connexions entre les différentes lignes.

Les informations sont traitées par les couches connectant les différents blocs $X_{i,j}$. La principale motivation de notre modèle est la différence entre les couches connectant les cartes de caractéristiques horizontales ou verticales :

Les connexions horizontales (représentées en vert dans le schéma 6.2) sont des unités résiduelles conservant la résolution et le nombre des cartes de caractéristiques. En suivant l'étude sur les connexions résiduelles faite part He *et al.* [He+16c], nous avons composé ces unités de deux opérations de convolutions, précédées d'opérateurs de non-linéarité (de type ReLU) et d'opérateurs de normalisation par batchs (voir détails dans le schéma 6.3).

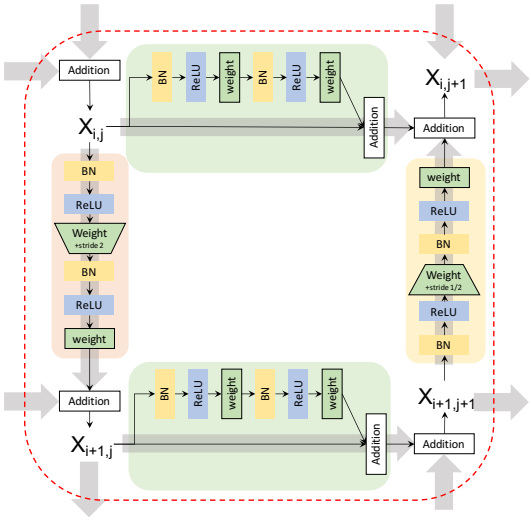


Fig. 6.3: Schéma détaillé des blocs composant notre GridNet. Les unités vertes sont résiduelles et gardes les dimensions des cartes de caractéristiques constantes entre l'entrée et la sortie. Les unités rouges sont convolutives. La partie "weight" en forme de losange est une couche convolutive avec un pas de deux, réduisant ainsi la résolution des cartes de caractéristiques et doublant leurs nombres. Les unités jaunes sont déconvolutives et doublent la résolution des cartes de caractéristiques avec une convolution stridée d'un pas d'un demi. BN = normalisation par batch.

Les unités verticales (blocs rouges et oranges dans le schéma 6.2) sont aussi convolutives, mais ces dernières modifient la taille et le nombre des cartes de caractéristiques. En fonction de leurs positions dans la grille, les dimensions spatiales des cartes de caractéristiques sont soit réduites (sous-échantillonnages) soit augmentées (sur-échantillonnages). Afin de réduire l'espace mémoire utilisé, le changement de résolution n'est pas effectué par des opérateurs de sous-échantillonnage ou sur-échantillonnage, mais est obtenu en modifiant le pas du masque de convolution. En effet, le pas du masque de convolution est généralement de 1, mais en jouant sur ce dernier il est possible de réduire ou augmenter la taille des cartes de caractéristiques obtenues. Par exemple, un pas de 2 produira une sortie pour une valeur sur deux, réduisant la taille des cartes de caractéristiques de 2. De la même manière, utiliser un pas d'une valeur (non entière) de $1/2$ augmentera la résolution des cartes de caractéristiques.

En suivant le schéma classique des réseaux de neurones convolutifs, nous construisons les opérateurs de sous-échantillonnage et de sur-échantillonnage afin que les lignes de la grille possèdent des cartes de caractéristiques dont la taille par rapport à la ligne précédente est diminuée d'un facteur 2 (sur les dimensions horizontale et verticale) et leur nombre multipliés par 2. Plus formellement, si la ligne X_i a pour entrée des cartes de caractéristiques de dimension $(F_i \times W_i \times H_i)$ où F_i est le nombre de cartes de caractéristiques et W_i, H_i sont respectivement leurs largeur et hauteur, alors la ligne X_{i+1} possèdera des cartes de caractéristiques de dimension de $(F_{I+1} \times W_{I+1} \times H_{I+1}) = (2F_i \times W_i/2 \times W_i/2)$.

Mis à part les blocs en bordure de la grille, chaque carte de caractéristiques $X_{i,j}$ dans la grille est le résultat de deux calculs différents : un calcul résiduel horizontal, traitant les données venant de $X_{i,j-1}$ et un calcul vertical traitant les données venant soit de $X_{i-1,j}$ pour les unités de sous-échantillonnage soit de $X_{i+1,j}$ pour les unités

de sur-échantillonnage. Nous nous sommes inspirés des réseaux conv-déconv pour construire GridNet. Par conséquent, nous construisons les différentes unités de notre grille de manière à ce que la première moitié des colonnes contiennent des unités de sous-échantillonnage et la seconde moitié, des unités de sur-échantillonnage. Néanmoins, il est possible de répartir les unités différemment, comme, par exemple, en alternant les colonnes de sous-échantillonnage et de sur-échantillonnage (cette architecture étant plus proche des Convolutional Neural Fabrics [SV16] de Saxena *et al.*). De plus, plusieurs choix peuvent aussi être faits sur la manière des combiner les cartes de caractéristiques venant des deux directions (horizontale et verticale). Nous avons décidé d'additionner les valeurs des cartes de caractéristiques afin de permettre une meilleure circulation du gradient dans la grille lors de l'entraînement et de limiter le nombre de paramètres d'apprentissage. Une autre stratégie, que nous avons testée (voir partie 6.4.1) consiste à concatèner les cartes de caractéristiques. Le problème de cette approche est qu'elle augmente significativement le nombre de paramètres puisque plus de cartes de caractéristiques sont données en entrée des opérations de convolution, ce qui rend l'apprentissage plus complexe (car il nécessite plus de données et plus de temps pour converger).

La définition formelle de notre grille est la suivante : Soit $\Theta^{Res}(.)$, $\Theta^{Sub}(.)$ et $\Theta^{Up}(.)$ les opérateurs de calculs respectifs pour les unités résiduelles (blocs verts dans la figure 6.2), les unités de sous-échantillonnage (blocs rouges) et de sur-échantillonnage (blocs jaunes). Chaque opérateur prend en entrée un tenseur de caractéristiques X et des paramètres θ.

Def. 6.1 *Si la colonne j est une colonne de sous-échantillonnage alors :*

Colonne de sous-échantillonnage

$$X_{i,j} = X_{i,j-1} + \Theta^{Res}(X_{i,j-1}, \theta^{Res}_{i,j}) + \Theta^{Sub}(X_{i-1,j}, \theta^{Sub}_{i,j}) \tag{6.1}$$

Def. 6.2 *Si, j est une colonne de sur-échantillonnage et alors :*

Colonne de sur-échantillonnage

$$X_{i,j} = X_{i,j-1} + \Theta^{Res}(X_{i,j-1}, \theta^{Res}_{i,j}) + \Theta^{Up}(X_{i+1,j}, \theta^{Up}_{i,j}) \tag{6.2}$$

Si i est la première ligne, respectivement la dernière ligne alors $\Theta^{Sub}(.) = 0$ respectivement $\Theta^{Up}(.) = 0$. De la même manière, si $j = 0$ alors $X_{i,j-1} + \Theta^{Res}(X_{i,j-1}, \theta^{Res}_{i,j}) = 0$.

La capacité de notre GridNet est définie par trois hyper-paramètres, N_S, N_{Cs} et N_{Cu} qui sont respectivement le nombre de lignes résiduelles, le nombre de colonnes de sous-échantillonnage et le nombre de colonnes de sur-échantillonnage. Inspirés par

la symétrie des réseaux conv-deconv (voir partie 4.3.3) nous contraignons $N_{Cs}=N_{Cu}$ dans nos expériences, mais cette contrainte peut être levée.

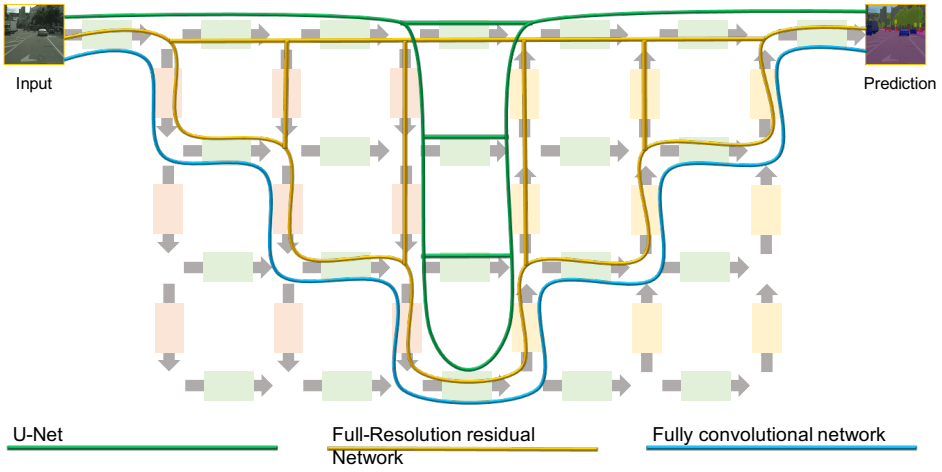


Fig. 6.4.: GridNet généralise plusieurs réseaux classiques, spécialisés pour la segmentation sémantique, tels que les réseaux Conv-Deconv (chemin bleu) (voir 4.3.3), les U-Networks (Chemin Vert) (voir 4.3.4) et le Full Resolution Residual Networks (FRRN) (chemin jaune).

GridNet généralise plusieurs réseaux classiques spécialisés pour la segmentation sémantique. La figure 6.4 montre des exemples de réseaux pouvant être retrouvés dans les différents chemins de la grille. Si l'on ne garde que le chemin bleu dans la figure 6.4, on obtient l'équivalent d'un réseau Conv-Deconv, avec une première partie convolutive et une seconde déconvolutive. Les U-Networks (expliqués dans la partie 4.3.4) sont illustrés dans la figure 6.4 avec le chemin Vert. Finalement, le réseau Full Resolution Residual Networks (FRRN) est illustré par le chemin jaune.

6.3.2 Dropout par blocs

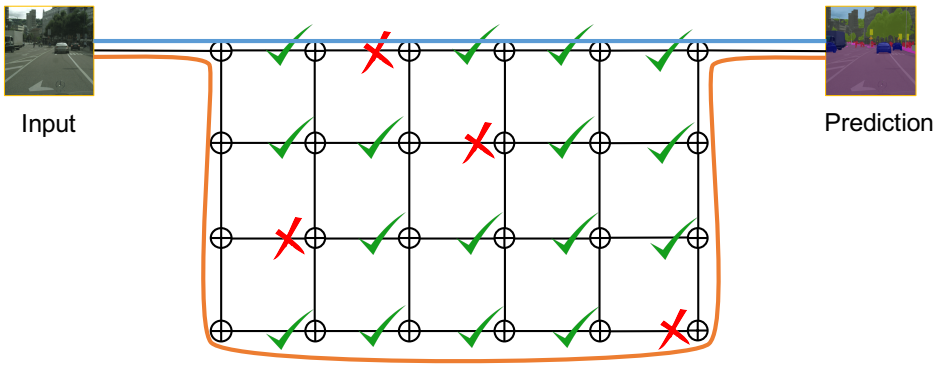


Fig. 6.5.: Le chemin bleu, qui utilise uniquement les connexions hautes résolutions, est plus court que le chemin orange qui utilise un chemin à basse résolution. Afin de forcer le réseau à utiliser tous les chemins de manière équivalente, nous coupons aléatoirement des chemins lors de l'entrainement, tel qu'illustré par les croix rouges.

Un effet secondaire de la topologie 2D de notre grille avec une entrée et une sortie toutes deux situées à la ligne 0 est que le chemin allant de l'entrée à la sortie est plus court en passant par la ligne de haute résolution (chemin bleu dans la figure 6.5) qu'en passant par la ligne de basse résolution (chemin orange dans la figure 6.5). Un chemin plus long dans un réseau de neurones convolutif peut entrainer le problème bien connu de perte du gradient ("*vanishing gradient*", expliqué dans la partie 2.3.4). Par conséquent, les chemins utilisant des lignes de résolution faible mettent plus de temps à converger dans le réseau et sont donc généralement plus difficiles à entrainer. Afin de forcer le réseau à utiliser toutes les lignes disponibles, nous avons développé une technique inspirée par l'opérateur de Dropout (voir partie 2.2.3) que nous avons appelé *dropout par blocs*. Lors de l'entrainement, nous coupons de façon aléatoire des chemins en mettant les valeurs résiduelles obtenues à zéro. Cela force le réseau à utiliser tous les chemins possibles.

Plus formellement, soit $r_{i,j} = Bernoulli(p)$ une variable aléatoire tirée selon une distribution de Bernoulli, qui est égale à 1 avec une probabilité p et égale à 0 autrement. Avec l'utilisation du total dropout, le calcul des cartes de caractéristiques des équations 6.1 et 6.2 devient :

Def. 6.3
Dropout par blocs

$$X_{i,j} = X_{i,j-1} + r_{i,j}(\Theta^{Res}(X_{i,j-1}, \theta^{Res}_{i,j})) + \Theta^{\{Sub;Up\}}(X_{i+1,j}, \theta^{\{Sub;Up\}}_{i,j}) \qquad (6.3)$$

Intuitivement ce dropout par blocs fonctionne de la même manière que l'opérateur de dropout traditionnel (vu dans la partie 2.2.3). L'idée étant que le réseau ne peut pas privilégier un chemin particulier, car celui-ci est régulièrement coupé lors de l'apprentissage. Ainsi, chaque chemin possible doit prendre part dans la décision finale et donc chaque chemin être capable d'apporter une information. De plus, tout comme l'opérateur de dropout traditionnel, cela correspond à utiliser l'ensemble des chemins possibles du réseau, produisant donc une prédiction équivalente à la moyenne des prédictions de l'ensemble des chemins.

6.3.3 Nombre de paramètres et empreinte mémoire

Dans les réseaux de neurones convolutifs, la quantité de mémoire utilisée dépend, à la fois, du nombre d'activations (sortie des différentes couches de calculs) et du nombre de paramètres entraînables. Dans de nombreuses architectures, ces deux paramètres sont fortement corrélés puisque, l'augmentation du nombre de couches utilisées, entraine une augmentation du nombre de paramètres et vice-versa. Bien

que ça soit toujours le cas pour notre GridNet, la structure en grille permet un contrôle plus fin de ces valeurs.

Considérons un GridNet construit selon les principes expliqués dans la section 6.3 : avec N_S lignes, N_{Cs} colonnes de sous-échantillonnage et N_{Cu} colonnes de sur-échantillonnage. Considérons aussi que ce GridNet possède F_0 nombre de cartes de caractéristiques à la ligne 0 d'une taille de $W_0 \times H_0$. Les autres lignes étant obtenues en réduisant la taille des cartes de caractéristiques par deux tout en augmentant leurs nombres d'un facteur 2.

On peut calculer le nombre approximatif du nombre de paramètres nb_{param} et du nombre de valeurs d'activations nb_{activ} de la manière suivante :

Res. 6.1 *GridNet : Nombre de paramètres*

$$nb_{param} \approx 18 \times 2^{2\times(N_s-1)}\, F_0^2\, (2.5N_{Cs} + N_{Cu} - 2) \tag{6.4}$$

De l'équation 6.4 on peut voir que le nombre de paramètre est principalement impacté par le nombre de ligne N_S, puis par le nombre de cartes de caractéristiques (contrôlé par F_0) et finalement par le nombre de colonnes $N_{Cs} + N_{Cu}$.

Res. 6.2 *GridNet : Nombre d'activations*

$$nb_{activ} \approx 6\, H_0\, W_0\, F_0\ (4N_{Cu} + 3N_{Cs} - 2) \tag{6.5}$$

L'équation 6.5 montre que le nombre d'activations dépend principalement de la taille (hauteur et largeur) et du nombre des cartes de caractéristiques de la première ligne et croît de façon linéaire avec l'augmentation du nombre de colonnes.

En pratique, l'empreinte mémoire du réseau lors de l'entrainement ne dépend pas seulement du nombre de paramètres et du nombre d'activations, mais aussi de l'optimiseur utilisé ainsi que de la taille du mini-batch. Le gradient calculé par l'optimiseur prend le même espace en mémoire que les paramètres eux-mêmes et ce dernier peut aussi garder des statistiques sur les paramètres du gradient (par exemple Adam [KB14]). La taille du mini-batch multiplie mécaniquement l'empreinte mémoire puisque les activations sont multipliées afin d'être calculées en parallèle.

Un aspect très intéressant à noter des équations 6.4 et 6.5 est que le nombre de cartes de caractéristiques sur la première ligne ainsi que le nombre de lignes permettent d'influer de manière quasi indépendante la consommation mémoire et le nombre de paramètres (même s'ils sont aussi tous les deux fortement liés aux nombres de colonnes de la grille). Par exemple, il est possible de diminuer la consommation mémoire du réseau en réduisant le nombre de cartes de caractéristiques utilisées sur

la première ligne ou d'augmenter le nombre de paramètres en ajoutant des lignes profondes.

6.4 Résultats

Nous avons testé notre réseau GridNet sur la base de données Cityscapes [Cor+16] (détaillée dans la partie 4.1.3). Cette dernière est composée d'images en haute résolution (1024×2048 pixels) capturées à partir d'un véhicule roulant dans les rues de 50 villes différentes d'Allemagne. 2975 images d'entraînement ainsi que 500 images de test ont été entièrement annotées avec un ensemble de 30 classes sémantiques différentes. Néanmoins, à cause de la rareté de certaines classes, seulement 19 classes sont utilisées pour l'évaluation automatique des algorithmes sur le site internet Cityscapes. Nous n'avons donc utilisé que ces 19 classes pour l'entraînement de notre réseau. Les classes sémantiques sont aussi groupées en 8 catégories différentes, utilisées pour évaluer la précision des algorithmes à un niveau de granularité supplémentaire. Les vérités terrains des exemples de l'ensemble de test ne sont pas fournies, mais l'évaluation en ligne des prédictions est disponible sur le site internet de la base de données [1]. La base donnée contient aussi 19998 images annotées grossièrement (avec des annotations polygonales) mais nous avons choisi de ne pas les utiliser pour l'entraînement, car elles augmentent le déséquilibre (en termes de nombre d'exemples) entre les classes, ce qui pénalise les mesures de performance utilisées.

Les mesures de performances utilisées pour l'évaluation des résultats sur la base Cityscapes sont basées sur l'IoU (vue dans la partie 4.1.2) aussi appelée index de Jaccard. L'*Intersection over Union* (Iou) est obtenue par la formule $\frac{TP}{TP+FP+FN}$ où TP, FP et FN sont, respectivement, le nombre de vrais positifs (True Positive), faux positifs (False Positive) et faux négatifs (False Negative) des pixels classés par le réseau. L'IoU est biaisée par rapport à la taille des instances qui couvrent une grande surface de l'image, par conséquent une mesure appelée instance-level intersection-over-union (iIoU) est aussi utilisée. L'iIoU est calculée en pondérant la contribution de chaque pixel par le ratio de la taille moyenne des instances de la classe, sur la taille de l'instance respective. Finalement, les résultats sont donnés aussi pour les deux granularités sémantiques (classes et catégories) ainsi qu'avec et sans la pondération par instance, ce qui donne quatre mesures de performance distinctes.

Pour nos expérimentations nous avons utilisé un GridNet composé de cinq lignes, chacune produisant séquentiellement des cartes de caractéristiques au nombre de

1. https ://www.cityscapes-dataset.com/

16, 32, 64, 128 et 256. De plus, la grille est composée de 3 colonnes de sous-échantillonnage (parties convolutives), suivie de 3 colonnes de sûr-échantillonnage (parties déconvolutives). Cette configuration, composée de 5 lignes / 6 colonnes, a été mise au point par essais/erreurs (voir table 6.2) et offre un bon compris entre l'utilisation mémoire et le nombre de paramètres du réseau : le réseau est suffisamment profond pour permettre une bonne capacité d'abstraction tout en ayant un nombre de paramètres raisonnables permettant d'éviter les phénomènes de sur-apprentissage. De plus, cette configuration nous permet d'utiliser un GPU avec une mémoire de 12Go et d'entraîner le réseau avec des batchs de quatre images de taille 400×400. Par conséquent, la ligne avec la résolution maximale utilise des cartes de caractéristiques de taille $256 \times 25 \times 25$.

Afin d'entraîner notre GridNet, nous découpons, dans les images d'entraînement, des patchs carrés de tailles aléatoires comprises entre 400×400 et 1024×1024 pixels, à des positions elles aussi aléatoires dans l'image initiale. Les patchs sont ensuite redimensionnés pour qu'ils fassent une taille de 400×400 pixels. Cette stratégie permet d'obtenir, à partir d'une image, plusieurs exemples d'entraînement, correspondant à différentes sous-parties de l'image à plusieurs échelles. Cela permet d'augmenter artificiellement le nombre d'exemples d'entraînement puisque les réseaux de neurones convolutifs ne sont pas invariants aux changements d'échelles (zoom d'une image). Afin d'augmenter encore le nombre d'exemples d'entraînement, nous appliquons aussi aléatoirement un retournement horizontal des images. Nous n'utilisons aucune technique de prétraitement, mais nous avons ajouté une opération de normalisation par batch à l'entrée de la grille. L'algorithme de descente de gradient utilisé est Adam (voir partie 2.2.1) avec un pas d'apprentissage (learning rate) de 0.01, une réduction du pas d'apprentissage (learning rate decay) de 5×10^{-6}, ainsi qu'un β_1 de 0.9, un β_2 de 0.999 et un ϵ de 1×10^{-8}. Après 800 époques, le pas d'apprentissage est réduit à 0.001. Nous arrêtons l'apprentissage après 10 jours d'entraînement, soit à peu près 1900 époques. Finalement, nous utilisons la fonction de perte classique (détaillée dans la partie 4.1.2) d'entropie croisée afin d'entraîner le réseau.

En phase de test, nous évaluons les images redimensionées à plusieurs échelles et nous utilisons un vote de majorité afin d'obtenir la prédiction finale, c'est-à-dire que nous faisons la moyenne des prédictions du réseau à chaque échelle. Pour nos expériences, nous avons redimensionné les images aux échelles $\frac{1}{1}$, $\frac{1}{1.5}$, $\frac{1}{2}$ et $\frac{1}{2.5}$. Cela permet au réseau une évaluation plus juste, mais prend plus de temps pour l'évaluation.

6.4.1 Discussion

Nous avons étudié les effets de chaque composant architectural décrit ci-dessus sur les performances de notre réseau GridNet. Les résultats sont présentés dans les tableaux 6.1 et 6.2.

Le tableau 6.1 analyse l'influence des connexions résiduelles, de l'opérateur de dropout par blocs ainsi qu'une fusion alternative des cartes de caractéristiques (concaténation au lieu d'une somme) sur les performances du réseau. La ligne Somme† est l'architecture que nous avons présentée dans la section 6.3 et que nous avons utilisée pour l'évaluation sur le site officiel de la base de données Cityscapes. Le tableau montre que l'opérateur de dropout par blocs est un élément important pour l'entraînement du réseau puisque ce dernier améliore considérablement les résultats obtenus.

Nous avons aussi testé une version de GridNet entièrement résiduelle. Dans cette version, les connexions verticales (blocs rouges et jaunes dans la figure 6.3) possèdent elles aussi des liens résiduels. Cette résidualité complète de la grille ne semble pas améliorer les résultats. Nous pensons que cela est dû à une trop forte contrainte de la grille (les connexions résiduelles étant censées apprendre une différence par rapport à leur entrée). De plus, l'opérateur de *dropout par blocs* est extrêmement néfaste quand la grille est entièrement résiduelle. Nous pensons qu'une résidualité complète pousse le réseau à n'utiliser qu'un seul chemin dans la grille plutôt que de profiter de l'ensemble des chemins disponibles. Or, le dropout par blocs est conçu pour empêcher le réseau de n'utiliser qu'un seul chemin, ce qui empêche le réseau de converger vers un résultat acceptable.

Finalement, nous avons testé une stratégie de fusion des cartes de caractéristiques différente. Au lieu de fusionner les cartes de caractéristiques horizontales et verticales avec un opérateur d'addition, nous utilisons un opérateur de concaténation. Cette concaténation donne de moins bons résultats que ceux obtenus avec la somme des cartes de caractéristiques. Nous pensons que cela est dû au nombre de paramètres supplémentaires ainsi qu'à la difficulté qu'à le gradient à se propager correctement dans la grille, augmentant les difficultés d'entraînement du réseau. Cette stratégie, théoriquement optimale (puisqu'elle permet au réseau de fusionner les cartes de caractéristiques comme il le souhaite) obtiendrait probablement de meilleurs résultats si elle pouvait bénéficier d'un plus grand nombre d'exemples d'apprentissage, ainsi que d'un entraînement plus poussé (plus d'époques d'entraînement et donc plus de temps d'apprentissage).

Fusion	h-résiduel	v-résiduel	Dropout par blocs	Mesure des performances			
				IoU class	iIoU class	IoU categ.	iIoU categ.
Somme			✓	60.2	34.5	83.9	67.3
Somme	✓			57.2	35.6	83.1	68.4
Somme†	✓		✓	**65.0**	**43.2**	85.6	70.1
Somme	✓	✓		57.6	36.8	**86.0**	**72.6**
Somme	✓	✓	✓	35.6	23.0	62.1	60.3
Concat.	✓			53.9	34.0	82.2	65.2

Tab. 6.1.: Résultats des différentes variantes de GridNet obtenus sur l'ensemble de validation de la base de données Cityscapes. "Fusion" indique les différentes stratégies de fusion des cartes de caractéristiques horizontales et verticales (somme ou concaténation). Les colonnes suivantes indiquent l'utilisation ou non de résidualité horizontale (h-résiduel) ou verticale (v-résiduel) ainsi que l'utilisation de l'opérateur de *dropout par blocs* lors de l'entraînement du réseau. † représente l'architecture utilisée pour l'évaluation finale de notre méthode.

Le tableau 6.2 montre l'impact du nombre de colonnes et de lignes utilisées dans GridNet. Nous avons initié nos expérimentations avec un GridNet composé de huit colonnes (quatre sous-échantillonnages suivi de quatre sur-échantillonnages) et cinq lignes. Au lieu d'utiliser 16 cartes de caractéristiques sur la première ligne, nous n'en utilisons que huit. Ceci réduit l'espace mémoire utilisé, ce qui permet une augmentation plus grande du nombre de colonnes et de lignes afin de mieux étudier leurs impacts. Les réseaux sont entraînés jusqu'à convergence et l'évaluation des performances sur l'ensemble de validation se fait sans augmentation de données (une seule échelle est utilisée, sans vote de majorité).

De la table 6.2, nous pouvons voir qu'augmenter le nombre de colonnes (de 8 à 16) n'améliore pas significativement les performances, mais augmente la complexité d'apprentissage puisque le nombre de paramètre augmentent. Par contre, augmenter le nombre de lignes améliore les performances (de 57.5 à 59.2 pour la performance d'iIoU par classe). Un nombre faible de lignes limite les capacités d'abstraction du réseau et donc pénalise la précision de ce dernier. Augmenter à la fois le nombre de colonnes et le nombre de lignes améliore toutes les performances.

Nb colonnes	Nb cartes de caractéristiques par lignes	Mesure des performances			
		IoU class	iIoU class	IoU categ.	iIoU categ.
8	{8, 16, 32, 64, 128}	57.5	40.0	83.8	**71.8**
16	{8, 16, 32, 64, 128}	56.3	38.6	82.3	70.2
8	{8, 16, 32, 64, 128, 256, 512}	59.2	41.1	83.5	71.0
12	{8, 16, 32, 64, 128, 256, 512}	**59.5**	**41.7**	**84.0**	70.9

Tab. 6.2.: Résultats de l'impact du nombre de colonnes et de lignes dans GridNet. Aucune augmentation de données artificielle (une seule échelle) n'a été utilisée lors de la phase de test.

Name	Sans pré-entraînement	Mesure des performances			
		IoU classe	iIoU classe	IoU categ.	iIoU categ.
FRRN - [Poh+16]	✓	**71.8**	**45.5**	**88.9**	**75.1**
GridNet	✓	69.45	44.06	87.85	71.11
GridNet - Alternative	✓	66.8	38.24	86.55	68.98
RefineNet - [Lin+16b]	✗	**73.6**	47.2	87.9	70.6
Lin *et al.*- [Lin+16a]	✗	71.6	**51.7**	87.3	74.1
LRR - [GF16]	✗	69.7	48	**88.2**	**74.7**
Yu *et al.*- [YK16]	✗	67.1	42	86.5	71.1
DPN - [Liu+15]	✗	66.8	39.1	86	69.1
FCN - [Lon+15]	✗	65.3	41.7	85.7	70.1
Chen *et al.*- [Che+16]	✗	63.1	34.5	81.2	58.7
Szegedy *et al.*- [Sze+15b]	✗	63	38.6	85.8	69.8
Zheng *et al.*- [Zhe+15b]	✗	62.5	34.4	82.7	66

Tab. 6.3.: Résultats des différentes méthodes de l'état de l'art reporté par l'évaluation en ligne (officielle) de la base de données Cityscapes. Nous ne donnons que les résultats qui ont fait l'objet d'une publication et qui utilisent les mêmes données que nous (pas d'annotations grossières ni de données stéréo). "GridNet - Alternative" est une autre structure plus proche de [SV16] où les opérations de sous-échantillonnages et sur-échantillonnages sont alternées.

6.4.2 Comparaison avec l'état de l'art

La figure 6.6 montre les résultats obtenus avec GridNet sur deux exemples provenant de l'ensemble de validation de la base de données Cityscapes. Dans le tableau 6.3, nous comparons les résultats obtenus avec l'évaluation en ligne de la base de données Cityscapes (et donc sur l'ensemble de test) avec les différentes méthodes de l'état de l'art. Nous limitons les comparaisons avec les méthodes qui utilisent les mêmes entrées que nous (sans images annotées grossièrement ni données stéréo). Notre réseau donne des résultats comparables à l'état de l'art et obtient même de meilleurs résultats que la majorité des réseaux pré-entraînés sur la base de données ImageNet ce qui montre le potentiel important de l'approche proposée. Par contre, nos résultats sont légèrement en dessous de ceux obtenus par Pohlen *et al.* et de leur réseau FRRN présenté dans la partie 6.2. Cela peut s'expliquer par plusieurs facteurs comme par exemple une meilleure gestion des hyper-paramètres. Cependant, nous pensons que la principale différence expliquant l'écart des résultats réside dans l'utilisation d'une fonction de perte privilégiant les mesures utilisées. Néanmoins, GridNet est une nouvelle architecture et de nombreuses études et modifications sont encore envisageables pour d'améliorer d'avantage les résultats.

6.5 Conclusion

Dans ce chapitre, nous avons présenté GridNet, une nouvelle architecture conçue spécialement pour la segmentation sémantique. Le modèle généralise plusieurs réseaux connus de l'état de l'art tels que les conv-deconv, U-Net et FRRN. Notre réseau GridNet montre des résultats prometteurs et ce même sans pré-entrainement.

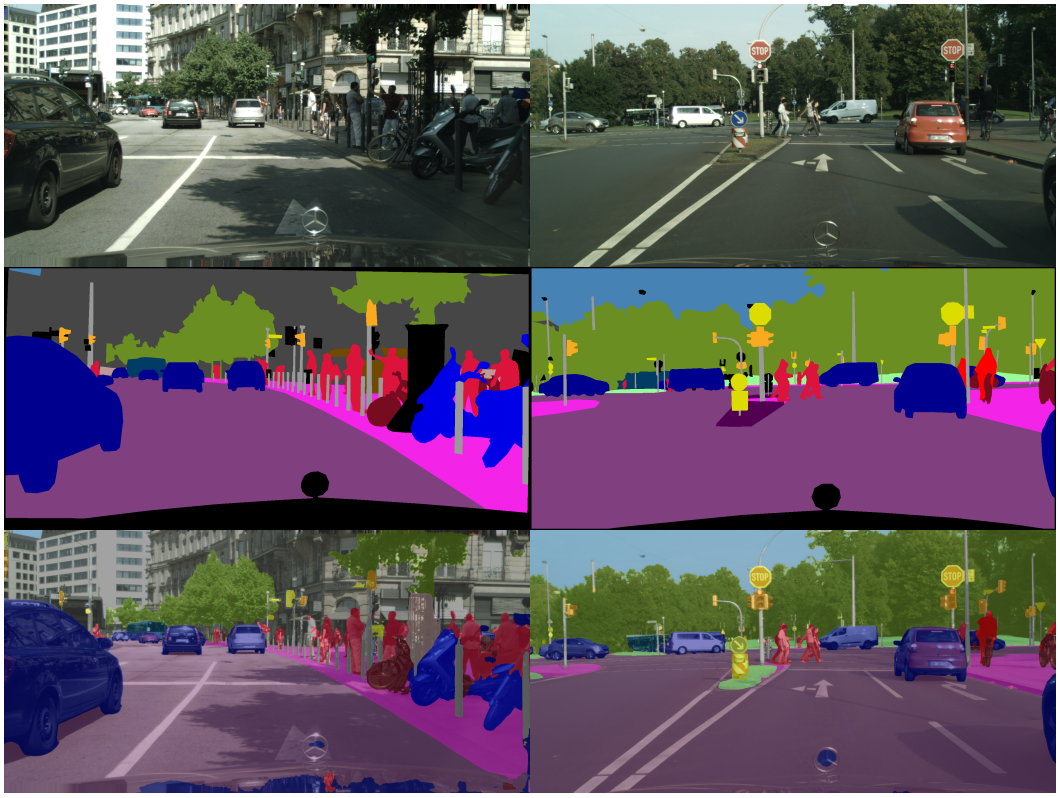


Fig. 6.6.: Résultats de la segmentation sémantique obtenue avec GridNet sur des images de l'ensemble de validation de la base de données Cityscapes. En haut, l'image d'entrée ; au milieu, la vérité terrain et en bas, nos résultats. Il est important de noter que la vérité terrain montre les 33 classes sémantiques de la base de données, alors que nos résultats n'évaluent que les 19 classes utilisées pour mesurer les performances.

Finalement, nous pensons que notre réseau pourrait aussi bénéficier d'une meilleure initialisation des poids, par exemple en le pré-entrainant sûr d'autres bases de données dédiées à la segmentation sémantique, tel que ADE20K [Zho+17].

7 Conclusion et Perspectives

7.1 Conclusions

Dans cette thèse nous avons étudié les différents éléments qui composent les réseaux de neurones convolutifs. Nous avons vu que les architectures des réseaux ont tendance à devenir de plus en plus profondes et de plus en plus complexes, entraînant des difficultés dans l'entraînement des réseaux et nécessitant la mise en œuvre de mécanismes améliorant l'entraînement. Nous avons notamment étudié puis adapté les différents opérateurs des réseaux afin de les spécialiser à différentes tâches.

Nous avons permis la spécialisation des réseaux en adaptant les différents composants. Notamment nous avons développé un opérateur de sous-échantillonnage adapté à la constance chromatique, en montrant, qu'avec ce dernier, un réseau de neurones peut modéliser les algorithmes de base de l'état de l'art. Nous avons pu ainsi développer deux architectures spécialisées à la tâche de constance chromatique et avons fait face à un manque de données d'apprentissages. Ce manque de données nous a poussé à mettre en place des solutions adaptées pour augmenter artificiellement le nombre de données, mais nous a aussi poussé à notre contribution suivante, une fonction de perte sélective.

Au cours de nos travaux nous avons constaté que le nombre de données d'entraînement est un facteur primordial pour le bon apprentissage des réseaux de neurones convolutifs. Bien que de nombreuses bases existent, ces dernières sont souvent limitées sur le nombre de données qu'elles contiennent. Nous avons donc développé une fonction de perte sélective qui permet d'entraîner un réseau sur plusieurs bases de façon simultanée et avons montré que cette dernière permet d'obtenir de meilleurs résultats qu'avec un entraînement séparé (et donc sur moins de données). Notre fonction a prouvé son efficacité sur la tâche d'estimation de la pose de la main, mais est surtout principalement adaptée à la tâche de segmentation sémantique qui souffrait d'un manque crucial de données (depuis nos premiers travaux, de nouvelles bases sont sorties, contenant beaucoup plus de données d'entraînement). Nous avons montré qu'elle permet également de découvrir et de prendre en compte les corrélations existantes entre les étiquettes des différents ensembles d'annotations.

Finalement, nous avons modifié complètement la structure des réseaux afin de permettre une prédiction dense liant à la fois résolution et précision. En effet, nous avons montré que les réseaux de l'état de l'art ont du mal à atteindre des hauts niveaux d'abstraction tout en gardant une prédiction avec une bonne résolution. Le goulot d'étranglement des réseaux conv-deconv permet d'atteindre un bon niveau d'abstraction, mais requiert une reconstruction plus délicate des cartes de caractéristiques créant généralement un manque de précision. Notre réseau GridNet, conçu pour traiter à la fois des hauts niveaux d'abstraction, tout en conservant une bonne résolution, a montré son efficacité et ce même sans pré-entraînement, prouvant qu'une structure linéaire du réseau n'est pas la plus adaptée et ouvrant de nombreuses perspectives sur l'étude des architectures.

7.2 Perspectives

Dans la suite des travaux, il serait intéressant d'étudier l'utilisation d'une structure unique permettant de répondre à plusieurs tâches simultanément. En effet, notre GridNet possède actuellement une unique sortie sur le chemin de plus haut niveau, mais pourrait bénéficier d'autres sorties, pour d'autres tâches, connectées à différents endroits de la grille. On pourrait par exemple ajouter une sortie avec une couche entièrement connectée à la sortie du chemin le plus profond du réseau (chemin portant le plus haut niveau d'abstraction) afin de réaliser une tâche de classification d'objets. Cette technique permettrait d'entraîner un GridNet avec plusieurs bases de données exploitées simultanément, sur des tâches très différentes, ce qui, théoriquement, devrait permettre au réseau d'apprendre des abstractions plus générales et donc plus performantes. De plus, si des structures topologiques sont probablement apprises par les réseaux lors de l'entraînement, ces dernières ne sont pas explicitement modélisées lors de l'apprentissage. En effet, un réseau apprend probablement, par exemple, qu'un piéton marche sur la route et non sur le ciel, mais il doit découvrir cette topologie uniquement en étant guidé à reconnaitre la route et les piétons de façon indépendantes. Notre GridNet, présente l'avantage d'utiliser plusieurs chemins portant des niveaux d'abstraction différents. Par conséquent, il pourrait être naturel d'ajouter, en plus de la prédiction sémantique en sortie du premier chemin, une sortie topologique en sortie d'un chemin de niveau inférieur.

Une seconde approche intéressante pourrait être d'adapter GridNet aux problèmes multi-vues. Les algorithmes multi-vues sont des algorithmes qui utilisent, comme leur nom l'indique, plusieurs vues en entrées. De la même façon qu'il serait intéressant d'utiliser plusieurs sorties à la grille, il pourrait être intéressant de connecter plusieurs entrées. Par exemple, nous avons vu que la base de données KITTI contient plusieurs types d'entrées différentes (des images acquises avec une caméra, mais aussi des

cartes de profondeurs obtenues à l'aide d'un laser ou encore des images stéréo). Il pourrait donc être envisageable d'utiliser toutes ces données simultanément, afin d'ajouter de l'information utilisable par le réseau (la profondeur peut par exemple guider efficacement le réseau à distinguer les frontières des différents éléments d'une scène). La notion de grille du réseau GridNet pourrait aider à apprendre et à extraire des caractéristiques communes ou complémentaires sur chacune des entrées, améliorant probablement les prédictions finales.

Plusieurs perspectives intéressantes sont également envisageables pour des études à plus long terme. Notamment plusieurs études sont actuellement en cours afin d'ajouter de la mémoire aux réseaux de neurones. Les réseaux récurrents tel que les LSTM (*Long Short Term Memory*) permettent actuellement aux réseaux de conserver une mémoire à court terme, mémorisant des informations dans des séquences d'entrées (ces réseaux sont notamment très utilisés dans l'estimation de gestes à partir de vidéos). Il serait intéressant d'étudier ces connexions récurrentes afin de permettre des prédictions à partir de vidéos d'entrées. L'utilisation des vidéos apportant potentiellement des cohérences spatiales et des informations supplémentaires tout en restant extrêmement cohérent dans l'utilisation de notre réseau, notamment dans le contexte du développement des voitures autonomes.

Bibliographie

[Ahm+08] Amr AHMED, Kai YU, Wei XU, Yihong GONG et Eric P. XING. „Training Hierarchical Feed-Forward Visual Recognition Models Using Transfer Learning from Pseudo-Tasks“. In : *Computer Vision - ECCV 2008, 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part III*. 2008, p. 69–82 (cf. p. 29).

[Arm+17] Iro ARMENI, Sasha SAX, Amir R ZAMIR et Silvio SAVARESE. „Joint 2D-3D-Semantic Data for Indoor Scene Understanding“. In : *ArXiv e-prints* (fév. 2017). arXiv : 1702.01105 [cs.CV] (cf. p. 81).

[AZ12] Relja ARANDJELOVIC et Andrew ZISSERMAN. „Three things everyone should know to improve object retrieval“. In : *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*. 2012, p. 2911–2918 (cf. p. 31).

[Bad+15] Vijay BADRINARAYANAN, Ankur HANDA et Roberto CIPOLLA. „SegNet : A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling“. In : *CoRR* abs/1505.07293 (2015) (cf. p. 89).

[Bar15] Jonathan T. BARRON. „Convolutional Color Constancy“. In : *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. 2015, p. 379–387 (cf. p. 67).

[Ben+10] Shai BEN-DAVID, John BLITZER, Koby CRAMMER et al. „A theory of learning from different domains“. In : *Machine Learning* 79.1-2 (2010), p. 151–175 (cf. p. 106).

[Bia+15] Simone BIANCO, Claudio CUSANO et Raimondo SCHETTINI. „Color constancy using CNNs“. In : *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops, Boston, MA, USA, June 7-12, 2015*. 2015, p. 81–89 (cf. p. 55, 65).

[Bro+08] Gabriel J. BROSTOW, Jamie SHOTTON, Julien FAUQUEUR et Roberto CIPOLLA. „Segmentation and Recognition Using Structure from Motion Point Clouds“. In : *ECCV (1)*. 2008, p. 44–57 (cf. p. 81).

[BS14] Pierre BALDI et Peter J. SADOWSKI. „The dropout learning algorithm“. In : *Artif. Intell.* 210 (2014), p. 78–122 (cf. p. 26).

[BT17] Jonathan T BARRON et Yun-Ta TSAI. „Fast Fourier Color Constancy“. In : *CVPR* (2017) (cf. p. 66).

[Buc80] Gershon BUCHSBAUM. „A spatial processor model for object colour perception“. In : *Journal of the Franklin institute* 310.1 (1980), p. 1–26 (cf. p. 48, 65).

[Car+02] Vlad C. CARDEI, Brian FUNT et Kobus BARNARD. „Estimating the scene illumination chromaticity by using a neural network“. In : *J. Opt. Soc. Am. A* 19.12 (2002), p. 2374–2386 (cf. p. 55).

[CF03] Florian CIUREA et Brian V. FUNT. „A Large Image Database for Color Constancy Research“. In : *The Eleventh Color Imaging Conference : Color Science and Engineering Systems, Technologies, Applications, CIC 2003, Scottsdale, Arizona, USA, November 4-7, 2003*. 2003, p. 160–164 (cf. p. 45, 62).

[CF99] Vlad C. CARDEI et Brian V. FUNT. „Committee-Based Color Constancy“. In : *The Seventh Color Imaging Conference : Color Science, Systems, and Applications Putting It All Together, CIC 1999, Scottsdale, Arizona, USA, November 16-19, 1999*. 1999, p. 311–313 (cf. p. 48).

[Cha01] Majed CHAMBAH. „Analyse et traitement de données chromatiques d'images numérisées à haute résolution : application à la restauration numérique des couleurs des films cinématographiques“. Thèse de doct. La Rochelle, 2001 (cf. p. 48).

[Che+14a] Liang-Chieh CHEN, George PAPANDREOU, Iasonas KOKKINOS, Kevin MURPHY et Alan L. YUILLE. „Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs“. In : *CoRR* abs/1412.7062 (2014) (cf. p. 95).

[Che+14b] Xianjie CHEN, Roozbeh MOTTAGHI, Xiaobai LIU et al. „Detect What You Can : Detecting and Representing Objects Using Holistic Models and Body Parts“. In : *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. 2014, p. 1979–1986 (cf. p. 96).

[Che+14c] D. CHENG, D. K. PRASAD et M. S. BROWN. „Illuminant estimation for color constancy : why spatial-domain methods work and the role of the color distribution“. In : *J. Opt. Soc. Am. A* 31.5 (2014), p. 1049–1058 (cf. p. 65).

[Che+16] Liang-Chieh CHEN, George PAPANDREOU, Iasonas KOKKINOS, Kevin MURPHY et Alan L. YUILLE. „DeepLab : Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs“. In : *CoRR* abs/1606.00915 (2016) (cf. p. 92, 96, 138).

[Col+11] Ronan COLLOBERT, Koray KAVUKCUOGLU et Clément FARABET. „Torch7 : A matlab-like environment for machine learning“. In : *BigLearn, NIPS Workshop*. EPFL-CONF-192376. 2011 (cf. p. 62, 108).

[Cor+16] Marius CORDTS, Mohamed OMRAN, Sebastian RAMOS et al. „The Cityscapes Dataset for Semantic Urban Scene Understanding“. In : *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. 2016, p. 3213–3223 (cf. p. 80, 123, 134).

[Dau+15] Yann N. DAUPHIN, Harm de VRIES, Junyoung CHUNG et Yoshua BENGIO. „RMSProp and equilibrated adaptive learning rates for non-convex optimization“. In : *CoRR* abs/1502.04390 (2015) (cf. p. 21).

[Den+09] Jia DENG, Wei DONG, Richard SOCHER et al. „ImageNet : A large-scale hierarchical image database“. In : *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. 2009, p. 248–255 (cf. p. 27, 61, 99).

[Doz16] Timothy DOZAT. „Incorporating nesterov momentum into adam“. In : (2016) (cf. p. 22).

[DT05] Navneet DALAL et Bill TRIGGS. „Histograms of Oriented Gradients for Human Detection“. In : *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*. 2005, p. 886–893 (cf. p. 14).

[Duc+11] John C. DUCHI, Elad HAZAN et Yoram SINGER. „Adaptive Subgradient Methods for Online Learning and Stochastic Optimization“. In : *Journal of Machine Learning Research* 12 (2011), p. 2121–2159 (cf. p. 21).

[Duc+17] Sylvain DUCHÊNE, Carlos ALIAGA, Tania POULI et Patrick PÉREZ. „Mixed illumination analysis in single image for interactive color grading“. In : *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering, NPAR 2017, Los Angeles, CA, USA, July 29-30, 2017*. 2017, 10 :1–10 :10 (cf. p. 67).

[Eve+10] M. EVERINGHAM, L. VAN GOOL, C. K. I. WILLIAMS, J. WINN et A. ZISSERMAN. „The Pascal Visual Object Classes (VOC) Challenge“. In : *International Journal of Computer Vision* 88.2 (juin 2010), p. 303–338 (cf. p. 77, 97).

[Far+13] C. FARABET, C. COUPRIE, L. NAJMAN et Y. LECUN. „Learning hierarchical features for scene labeling“. In : *IEEE TPAMI* 8 (2013) (cf. p. 108).

[Fel05] Christiane FELLBAUM. „WordNet and wordnets“. In : (2005) (cf. p. 27).

[Fin+01] Graham D. FINLAYSON, Steven D. HORDLEY et Paul M. HUBEL. „Color by Correlation : A Simple, Unifying Framework for Color Constancy“. In : *IEEE Trans. Pattern Anal. Mach. Intell.* 23.11 (2001), p. 1209–1221 (cf. p. 53).

[Fin13] Graham D. FINLAYSON. „Corrected-Moment Illuminant Estimation“. In : *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*. 2013, p. 1904–1911 (cf. p. 65).

[For90] David A. FORSYTH. „A novel algorithm for color constancy“. In : *International Journal of Computer Vision* 5.1 (1990), p. 5–35 (cf. p. 53).

[Fou+16a] Damien FOURURE, Rémi EMONET, Élisa FROMONT et al. „Mixed pooling neural networks for color constancy“. In : *2016 IEEE International Conference on Image Processing, ICIP 2016, Phoenix, AZ, USA, September 25-28, 2016*. 2016 (cf. p. 6).

[Fou+16b] Damien FOURURE, Rémi EMONET, Elisa FROMONT et al. „Segmentation de scènes extérieures à partir d'ensembles d'étiquettes à granularité et sémantique variables“. In : *RFIA 2016*. 2016 (cf. p. 6).

[Fou+16c] Damien FOURURE, Rémi EMONET, Élisa FROMONT et al. „Semantic Segmentation via Multi-task, Multi-domain Learning“. In : *Structural, Syntactic, and Statistical Pattern Recognition - Joint IAPR International Workshop, S+SSPR 2016, Mérida, Mexico, November 29 - December 2, 2016, Proceedings*. 2016 (cf. p. 6).

[Fou+17a] Damien FOURURE, Rémi EMONET, Élisa FROMONT et al. „Multi-task, multi-domain learning : Application to semantic segmentation and pose regression“. In : *Neurocomputing* (2017) (cf. p. 6).

[Fou+17b] Damien FOURURE, Rémi EMONET, Elisa FROMONT et al. „Residual Conv-Deconv Grid Network for Semantic Segmentation“. In : *Proceedings of the British Machine Vision Conference, 2017*. 2017 (cf. p. 7).

[FS01] Graham D. FINLAYSON et Gerald SCHAEFER. „Solving for Colour Constancy using a Constrained Dichromatic Reflection Model“. In : *International Journal of Computer Vision* 42.3 (2001), p. 127–144 (cf. p. 52).

[FS10] Brian FUNT et Lilong SHI. „The Rehabilitation of MaxRGB“. In : *18th Color and Imaging Conference, CIC 2010, San Antonio, Texas, USA, November 8-12, 2010*. 2010, p. 256–259 (cf. p. 48, 65).

[FT04] Graham D. FINLAYSON et Elisabetta TREZZI. „Shades of Gray and Colour Constancy“. In : *The Twelfth Color Imaging Conference : Color Science and Engineering Systems, Technologies, Applications, CIC 2004, Scottsdale, Arizona, USA, November 9-12, 2004*. 2004, p. 37–41 (cf. p. 49, 65).

[Gao+14] Shaobing GAO, Wangwang HAN, Kaifu YANG, Chaoyi LI et Yongjie LI. „Efficient Color Constancy with Local Surface Reflectance Statistics“. In : *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part II*. 2014, p. 158–173 (cf. p. 50, 51, 55, 56, 65).

[Geh+08] Peter V. GEHLER, Carsten ROTHER, Andrew BLAKE, Thomas P. MINKA et Toby SHARP. „Bayesian color constancy revisited“. In : *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2008), 24-26 June 2008, Anchorage, Alaska, USA*. 2008 (cf. p. 46).

[Gei+13] Andreas GEIGER, Philip LENZ, Christoph STILLER et Raquel URTASUN. „Vision meets Robotics : The KITTI Dataset“. In : *International Journal of Robotics Research (IJRR)* (2013) (cf. p. 78).

[GF16] Golnaz GHIASI et Charless C. FOWLKES. „Laplacian Pyramid Reconstruction and Refinement for Semantic Segmentation“. In : *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III*. 2016, p. 519–534 (cf. p. 138).

[Gij+10] Arjan GIJSENIJ, Theo GEVERS et Joost van de WEIJER. „Generalized Gamut Mapping using Image Derivative Structures for Color Constancy“. In : *International Journal of Computer Vision* 86.2-3 (2010), p. 127–139 (cf. p. 53).

[GL15] Y. GANIN et V.S. LEMPITSKY. „Unsupervised Domain Adaptation by Backpropagation“. In : *ICML*. 2015 (cf. p. 107, 118).

[Glo+11] Xavier GLOROT, Antoine BORDES et Yoshua BENGIO. „Deep Sparse Rectifier Neural Networks“. In : *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*. 2011, p. 315–323 (cf. p. 13).

[GM16] Roger B. GROSSE et James MARTENS. „A Kronecker-factored approximate Fisher matrix for convolution layers“. In : *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. 2016, p. 573–582 (cf. p. 92).

[Gou+09] S. GOULD, R. FULTON et D. KOLLER. „Decomposing a scene into geometric and semantically consistent regions“. In : *ICCV*. 2009 (cf. p. 76, 106, 112, 115, 158).

[He+15] Kaiming HE, Xiangyu ZHANG, Shaoqing REN et Jian SUN. „Delving Deep into Rectifiers : Surpassing Human-Level Performance on ImageNet Classification“. In : *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. 2015, p. 1026–1034 (cf. p. 14).

[He+16a] Kaiming HE, Xiangyu ZHANG, Shaoqing REN et Jian SUN. „Deep Residual Learning for Image Recognition“. In : *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. 2016, p. 770–778 (cf. p. 35, 93, 123, 125).

[He+16b] Kaiming HE, Xiangyu ZHANG, Shaoqing REN et Jian SUN. „Deep Residual Learning for Image Recognition“. In : *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. 2016, p. 770–778 (cf. p. 81).

[He+16c] Kaiming HE, Xiangyu ZHANG, Shaoqing REN et Jian SUN. „Identity Mappings in Deep Residual Networks“. In : *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*. 2016, p. 630–645 (cf. p. 125, 128).

[Hin+15] Geoffrey E. HINTON, Oriol VINYALS et Jeffrey DEAN. „Distilling the Knowledge in a Neural Network“. In : *CoRR* abs/1503.02531 (2015) (cf. p. 29).

[HK12] Tatsuya HARADA et Yasuo KUNIYOSHI. „Graphical Gaussian Vector for Image Categorization“. In : *Advances in Neural Information Processing Systems 25 : 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. 2012, p. 1556–1564 (cf. p. 31).

[HU13] Hu HE et Ben UPCROFT. „Nonparametric semantic segmentation for 3d street scenes“. In : *Intelligent Robots and Systems (IROS)*. 2013 (cf. p. 103, 111, 112, 115, 157–159).

[Hua+16a] Gao HUANG, Yu SUN, Zhuang LIU, Daniel SEDRA et Kilian Q. WEINBERGER. „Deep Networks with Stochastic Depth“. In : *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*. 2016, p. 646–661 (cf. p. 36).

[Hua+16b] Gao HUANG, Zhuang LIU et Kilian Q. WEINBERGER. „Densely Connected Convolutional Networks“. In : *CoRR* abs/1608.06993 (2016) (cf. p. 37, 81).

[Hua+17] Gao HUANG, Danlu CHEN, Tianhong LI et al. „Multi-Scale Dense Convolutional Networks for Efficient Prediction“. In : *CoRR* abs/1703.09844 (2017) (cf. p. 126).

[IS15a] S. IOFFE et C. SZEGEDY. „Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift“. In : *ICML*. 2015 (cf. p. 109).

[IS15b] Sergey IOFFE et Christian SZEGEDY. „Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift“. In : *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. 2015, p. 448–456 (cf. p. 26, 27, 62).

[JD14] Hamid Reza Vaezi JOZE et Mark S. DREW. „Exemplar-Based Color Constancy and Multiple Illumination“. In : *IEEE Trans. Pattern Anal. Mach. Intell.* 36.5 (2014), p. 860–873 (cf. p. 53, 65, 67).

[JG14] Suyog Dutt JAIN et Kristen GRAUMAN. „Supervoxel-Consistent Foreground Propagation in Video“. In : *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part IV*. 2014, p. 656–671 (cf. p. 81).

[KB14] Diederik P. KINGMA et Jimmy BA. „Adam : A Method for Stochastic Optimization“. In : *CoRR* abs/1412.6980 (2014) (cf. p. 21, 22, 133).

[KK11] Philipp KRÄHENBÜHL et Vladlen KOLTUN. „Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials“. In : *Advances in Neural Information Processing Systems 24 : 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.* 2011, p. 109–117 (cf. p. 95, 96).

[Kri+12] Alex KRIZHEVSKY, Ilya SUTSKEVER et Geoffrey E. HINTON. „ImageNet Classification with Deep Convolutional Neural Networks“. In : *Advances in Neural Information Processing Systems 25 : 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.* 2012, p. 1106–1114 (cf. p. 31, 32, 81).

[Kri70] J. von KRIES. „Influence of adaptation on the effects produced by luminous stimuli“. In : *Sources of color science*. Sous la dir. de D.L. MACADAM. Handbook of Stuff I Care About. Cambridge, MA : MIT Press, 1970, p. 120–126 (cf. p. 43).

[Kun+14] Abhijit KUNDU, Yin LI, Frank DELLAERT, Fuxin LI et James M REHG. „Joint semantic segmentation and 3d reconstruction from monocular video“. In : *ECCV*. 2014 (cf. p. 103, 111, 115, 157–159).

[Lad+14] L. LADICKY, J. SHI et M. POLLEFEYS. „Pulling things out of perspective“. In : *CVPR*. 2014 (cf. p. 111, 112, 115, 157–159).

[LeC+98] Yann LECUN, Léon BOTTOU, Yoshua BENGIO et Patrick HAFFNER. „Gradient-based learning applied to document recognition“. In : *Proceedings of the IEEE* 86.11 (1998), p. 2278–2324 (cf. p. 30).

[Li+14] Bing LI, Weihua XIONG, Weiming HU et Brian V. FUNT. „Evaluating Combinational Illumination Estimation Methods on Real-World Images“. In : *IEEE Trans. Image Processing* 23.3 (2014), p. 1194–1209 (cf. p. 65).

[Lin+13] Min LIN, Qiang CHEN et Shuicheng YAN. „Network in network“. In : *arXiv preprint arXiv :1312.4400* (2013) (cf. p. 34).

[Lin+14] Tsung-Yi LIN, Michael MAIRE, Serge J. BELONGIE et al. „Microsoft COCO : Common Objects in Context“. In : *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*. 2014, p. 740–755 (cf. p. 79).

[Lin+16a] Guosheng LIN, Chunhua SHEN, Anton van den HENGEL et Ian D. REID. „Efficient Piecewise Training of Deep Structured Models for Semantic Segmentation“. In : *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. 2016, p. 3194–3203 (cf. p. 138).

[Lin+16b] Guosheng LIN, Anton MILAN, Chunhua SHEN et Ian D. REID. „RefineNet : Multi-Path Refinement Networks for High-Resolution Semantic Segmentation“. In : *CoRR* abs/1611.06612 (2016) (cf. p. 91, 124, 138).

[Liu+11] Ce LIU, Jenny YUEN et Antonio TORRALBA. „Nonparametric Scene Parsing via Label Transfer“. In : *IEEE TPAMI* 12 (2011) (cf. p. 78, 106, 112, 115, 159).

[Liu+15] Ziwei LIU, Xiaoxiao LI, Ping LUO, Chen Change LOY et Xiaoou TANG. „Semantic Image Segmentation via Deep Parsing Network“. In : *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. 2015, p. 1377–1385 (cf. p. 138).

[Lon+15] Jonathan LONG, Evan SHELHAMER et Trevor DARRELL. „Fully convolutional networks for semantic segmentation“. In : *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. 2015, p. 3431–3440 (cf. p. 86, 87, 123, 124, 138).

[Low99] David G. LOWE. „Object Recognition from Local Scale-Invariant Features“. In : *ICCV*. 1999, p. 1150–1157 (cf. p. 14).

[Lyn+13] Stuart E LYNCH, Mark S DREW et Graham D FINLAYSON. „Colour constancy from both sides of the shadow edge“. In : *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*. IEEE. 2013, p. 899–906 (cf. p. 47).

[Maa+13] Andrew L MAAS, Awni Y HANNUN et Andrew Y NG. „Rectifier nonlinearities improve neural network acoustic models“. In : *Proc. ICML*. T. 30. 1. 2013 (cf. p. 14).

[Man+09] Yishay MANSOUR, Mehryar MOHRI et Afshin ROSTAMIZADEH. „Domain Adaptation : Learning Bounds and Algorithms“. In : *COLT 2009 - The 22nd Conference on Learning Theory, Montreal, Quebec, Canada, June 18-21, 2009*. 2009 (cf. p. 106).

[Men+12] Thomas MENSINK, Jakob J. VERBEEK, Florent PERRONNIN et Gabriela CSURKA. „Metric Learning for Large Scale Image Classification : Generalizing to New Classes at Near-Zero Cost“. In : *Computer Vision - ECCV 2012 - 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part II*. 2012, p. 488–501 (cf. p. 31).

[Mor+17] Taylor MORDAN, Nicolas THOME, Matthieu CORD et Gilles HENAFF. „Deformable Part-based Fully Convolutional Network for Object Detection“. In : (2017) (cf. p. 70).

[Nes83] Yurii NESTEROV. „A method for unconstrained convex minimization problem with the rate of convergence O (1/kˆ 2)“. In : *Doklady AN USSR*. T. 269. 1983, p. 543–547 (cf. p. 21).

[Nev+15a] N. NEVEROVA, C. WOLF, G. TAYLOR et F. NEBOUT. „Hand Pose Estimation through Semi-Supervised and Weakly-Supervised Learning“. In : *arXiv :1511.06728*. 2015 (cf. p. 120, 121).

[Nev+15b] Natalia NEVEROVA, Christian WOLF, Florian NEBOUT et Graham W. TAYLOR. „Hand Pose Estimation through Weakly-Supervised Learning of a Rich Intermediate Representation“. In : *CoRR* abs/1511.06728 (2015) (cf. p. 85).

[New+16] Alejandro NEWELL, Kaiyu YANG et Jia DENG. „Stacked Hourglass Networks for Human Pose Estimation“. In : *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII*. 2016, p. 483–499 (cf. p. 91, 125).

[Noh+15] Hyeonwoo NOH, Seunghoon HONG et Bohyung HAN. „Learning Deconvolution Network for Semantic Segmentation“. In : *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. 2015, p. 1520–1528 (cf. p. 87, 88, 123, 124).

[Obe+15] M. OBERWEGER, P. WOHLHART et V. LEPETIT. „Hands Deep in Deep Learning for Hand Pose Estimation“. In : *CVWW*. 2015 (cf. p. 121).

[Oqu+14] Maxime OQUAB, Léon BOTTOU, Ivan LAPTEV et Josef SIVIC. „Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks“. In : *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. 2014, p. 1717–1724 (cf. p. 29).

[Pas+16] Adam PASZKE, Abhishek CHAURASIA, Sangpil KIM et Eugenio CULURCIELLO. „ENet : A Deep Neural Network Architecture for Real-Time Semantic Segmentation“. In : *CoRR* abs/1606.02147 (2016) (cf. p. 92).

[Per+12] Florent PERRONNIN, Zeynep AKATA, Zaïd HARCHAOUI et Cordelia SCHMID. „Towards good practice in large-scale learning for image classification“. In : *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*. 2012, p. 3482–3489 (cf. p. 31).

[Poh+16] Tobias POHLEN, Alexander HERMANS, Markus MATHIAS et Bastian LEIBE. „Full-Resolution Residual Networks for Semantic Segmentation in Street Scenes“. In : *CoRR* abs/1611.08323 (2016) (cf. p. 89, 90, 126, 138).

[Pre+12] Alessandro PREST, Christian LEISTNER, Javier CIVERA, Cordelia SCHMID et Vittorio FERRARI. „Learning object class detectors from weakly annotated video“. In : *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*. 2012, p. 3282–3289 (cf. p. 81).

[Qia+16] Yanlin QIAN, Ke CHEN, Joni-Kristian KAMARAINEN, Jarno NIKKANEN et Jiri MATAS. „Deep structured-output regression learning for computational color constancy“. In : *23rd International Conference on Pattern Recognition, ICPR 2016, Cancún, Mexico, December 4-8, 2016*. 2016, p. 1899–1904 (cf. p. 66).

[Qia99] Ning QIAN. „On the momentum term in gradient descent learning algorithms“. In : *Neural Networks* 12.1 (1999), p. 145–151 (cf. p. 21).

[Qua14] Alistair James QUADROS. „Representing 3D shape in sparse range images for urban object classification“. Thèse de doct. University of Sydney, Australia, 2014 (cf. p. 81).

[RB93] Martin RIEDMILLER et Heinrich BRAUN. „A direct adaptive method for faster backpropagation learning : The RPROP algorithm“. In : *Neural Networks, 1993., IEEE International Conference on*. IEEE. 1993, p. 586–591 (cf. p. 62).

[Red+16] Joseph REDMON, Santosh Kumar DIVVALA, Ross B. GIRSHICK et Ali FARHADI. „You Only Look Once : Unified, Real-Time Object Detection“. In : *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. 2016, p. 779–788 (cf. p. 70).

[RF16] Joseph REDMON et Ali FARHADI. „YOLO9000 : Better, Faster, Stronger“. In : *CoRR* abs/1612.08242 (2016) (cf. p. 70).

[Ron+15] Olaf RONNEBERGER, Philipp FISCHER et Thomas BROX. „U-Net : Convolutional Networks for Biomedical Image Segmentation“. In : *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III*. 2015, p. 234–241 (cf. p. 90, 123, 124).

[Ros+15] German ROS, Sebastian RAMOS, Manuel GRANADOS et al. „Vision-based offline-online perception paradigm for autonomous driving“. In : *Winter Conference on Applications of Computer Vision (WACV)*. 2015 (cf. p. 111, 115, 157–159).

[Ros+16] German ROS, Laura SELLART, Joanna MATERZYNSKA, David VAZQUEZ et Antonio LOPEZ. „The SYNTHIA Dataset : A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes“. In : *CVPR*. 2016 (cf. p. 81).

[Rus+15] Olga RUSSAKOVSKY, Jia DENG, Hao SU et al. „ImageNet Large Scale Visual Recognition Challenge“. In : *International Journal of Computer Vision (IJCV)* 115.3 (2015), p. 211–252 (cf. p. 3, 28, 31).

[Sch+12] Walter J. SCHEIRER, Neeraj KUMAR, Peter N. BELHUMEUR et Terrance E. BOULT. „Multi-attribute spaces : Calibration for attribute fusion and similarity search“. In : *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*. 2012, p. 2933–2940 (cf. p. 31).

[Sen+13] Sabyasachi SENGUPTA, Eric GREVESON, Ali SHAHROKNI et Philip HS TORR. „Urban 3d semantic modelling using stereo vision“. In : *IEEE ICRA*. 2013 (cf. p. 111, 115, 157–159).

[Ser+13] Pierre SERMANET, David EIGEN, Xiang ZHANG et al. „OverFeat : Integrated Recognition, Localization and Detection using Convolutional Networks“. In : *CoRR* abs/1312.6229 (2013) (cf. p. 85).

[SF10] Lilong SHI et Brian FUNT. „Re-processed version of the gehler color constancy dataset of 568 images“. In : *Simon Fraser University* (2010) (cf. p. 47, 62).

[Shi+16] Wu SHI, Chen Change LOY et Xiaoou TANG. „Deep Specialized Network for Illuminant Estimation“. In : *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*. 2016, p. 371–387 (cf. p. 66).

[SP11] Jorge SÁNCHEZ et Florent PERRONNIN. „High-dimensional signature compression for large-scale image classification“. In : *The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011*. 2011, p. 1665–1672 (cf. p. 31).

[Sri+14] Nitish SRIVASTAVA, Geoffrey E. HINTON, Alex KRIZHEVSKY, Ilya SUTSKEVER et Ruslan SALAKHUTDINOV. „Dropout : a simple way to prevent neural networks from overfitting“. In : *Journal of Machine Learning Research* 15.1 (2014), p. 1929–1958 (cf. p. 25).

[Sri+15] Rupesh Kumar SRIVASTAVA, Klaus GREFF et Jürgen SCHMIDHUBER. „Highway Networks“. In : *CoRR* abs/1505.00387 (2015) (cf. p. 36).

[Sut+07] Charles A. SUTTON, Andrew MCCALLUM et Khashayar ROHANIMANESH. „Dynamic Conditional Random Fields : Factorized Probabilistic Models for Labeling and Segmenting Sequence Data“. In : *Journal of Machine Learning Research* 8 (2007), p. 693–723 (cf. p. 94).

[SV16] Shreyas SAXENA et Jakob VERBEEK. „Convolutional Neural Fabrics“. In : *Advances in Neural Information Processing Systems 29 : Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016, p. 4053–4061 (cf. p. 126, 130, 138).

[Sze+15a] Christian SZEGEDY, Wei LIU, Yangqing JIA et al. „Going deeper with convolutions“. In : *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. 2015, p. 1–9 (cf. p. 33, 34, 37, 81).

[Sze+15b] Christian SZEGEDY, Wei LIU, Yangqing JIA et al. „Going deeper with convolutions“. In : *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. 2015, p. 1–9 (cf. p. 138).

[Sán+12] Jorge SÁNCHEZ, Florent PERRONNIN et Teófilo Emídio de CAMPOS. „Modeling the spatial layout of images beyond spatial pyramids“. In : *Pattern Recognition Letters* 33.16 (2012), p. 2216–2223 (cf. p. 31).

[Tan+04] Robby T TAN, Ko NISHINO et Katsushi IKEUCHI. „Color constancy through inverse-intensity chromaticity space“. In : *JOSA A* 21.3 (2004), p. 321–334 (cf. p. 52).

[Tan+13] D. TANG, T.H. YU et T-K. KIM. „Real-time Articulated Hand Pose Estimation using Semi-supervised Transductive Regression Forests“. In : *ICCV*. 2013 (cf. p. 119–121).

[Tar+16] Sasha TARG, Diogo ALMEIDA et Kevin LYMAN. „Resnet in Resnet : Generalizing Residual Architectures“. In : *CoRR* abs/1603.08029 (2016) (cf. p. 36, 37).

[Tom+14] Jonathan TOMPSON, Murphy STEIN, Yann LECUN et Ken PERLIN. „Real-Time Continuous Pose Recovery of Human Hands Using Convolutional Networks“. In : *ACM Trans. Graph.* 5 (sept. 2014), 169 :1–169 :10 (cf. p. 119–121).

[VHS05] Hermann VON HELMHOLTZ et James Powell Cocke SOUTHALL. *Treatise on physiological optics*. T. 3. Courier Corporation, 2005 (cf. p. 48).

[Wei+07] Joost van de WEIJER, Theo GEVERS et Arjan GIJSENIJ. „Edge-Based Color Constancy“. In : *IEEE Trans. Image Processing* 16.9 (2007), p. 2207–2214 (cf. p. 50, 65).

[Wu+16] Zifeng WU, Chunhua SHEN et Anton van den HENGEL. „Wider or Deeper : Revisiting the ResNet Model for Visual Recognition“. In : *CoRR* abs/1611.10080 (2016) (cf. p. 93, 123, 125).

[Xu+13] Philippe XU, Franck DAVOINE, Jean-Baptiste BORDES, Huijing ZHAO et Thierry DENOEUX. „Information fusion on oversegmented images : An application for urban scene understanding“. In : *IAPR MVA*. 2013 (cf. p. 111, 112, 115, 157–159).

[Yan+15] Kai-Fu YANG, Shao-Bing GAO et Yong-Jie LI. „Efficient illuminant estimation for color constancy using grey pixels“. In : *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. 2015, p. 2254–2263 (cf. p. 65).

[YK16] Fisher YU et Vladlen KOLTUN. „Multi-Scale Context Aggregation by Dilated Convolutions“. In : *International Conference on Learning Representations (ICLR)*. 2016 (cf. p. 92, 125, 138).

[Yos+14] Jason YOSINSKI, Jeff CLUNE, Yoshua BENGIO et Hod LIPSON. „How transferable are features in deep neural networks ?“ In : *Advances in Neural Information Processing Systems 27 : Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. 2014, p. 3320–3328 (cf. p. 29, 81).

[Zei12] Matthew D. ZEILER. „ADADELTA : An Adaptive Learning Rate Method“. In : *CoRR* abs/1212.5701 (2012) (cf. p. 21).

[ZF13] Matthew D. ZEILER et Rob FERGUS. „Stochastic Pooling for Regularization of Deep Convolutional Neural Networks“. In : *CoRR* abs/1301.3557 (2013) (cf. p. 17).

[ZF14] M.D. ZEILER et R. FERGUS. „Visualizing and Understanding Convolutional Networks“. In : *ECCV*. 2014 (cf. p. 28, 103).

[Zha+15] Richard ZHANG, Stefan A CANDRA, Kai VETTER et Avideh ZAKHOR. „Sensor fusion for semantic segmentation of urban scenes“. In : *IEEE ICRA*. 2015 (cf. p. 111, 115, 157–159).

[Zha+16] Hengshuang ZHAO, Jianping SHI, Xiaojuan QI, Xiaogang WANG et Jiaya JIA. „Pyramid Scene Parsing Network“. In : *CoRR* abs/1612.01105 (2016) (cf. p. 93, 125).

[Zhe+15a] Shuai ZHENG, Sadeep JAYASUMANA, Bernardino ROMERA-PAREDES et al. „Conditional Random Fields as Recurrent Neural Networks“. In : *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. 2015, p. 1529–1537 (cf. p. 96).

[Zhe+15b] Shuai ZHENG, Sadeep JAYASUMANA, Bernardino ROMERA-PAREDES et al. „Conditional Random Fields as Recurrent Neural Networks“. In : *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. 2015, p. 1529–1537 (cf. p. 138).

[Zho+15] Yisu ZHOU, Xiaolin HU et Bo ZHANG. „Interlinked Convolutional Neural Networks for Face Parsing“. In : *Advances in Neural Networks - ISNN 2015 - 12th International Symposium on Neural Networks, ISNN 2015, Jeju, South Korea, October 15-18, 2015, Proceedings*. 2015, p. 222–231 (cf. p. 126).

[Zho+17] Bolei ZHOU, Hang ZHAO, Xavier PUIG et al. „Scene Parsing through ADE20K Dataset“. In : *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017 (cf. p. 81, 139).

Annexes fonction sélective

A

Tab. A.1.: Détails des précisions obtenues avec les différentes stratégies d'apprentissage sur les sept sous-ensembles de la base de données KITTI. Les meilleurs résultats sont affichés en gras.

Méthodes	Précisions	Apprentissage avec les 7 sous-ensembles de la bases KITTI							
		He [HU13]	Kundu [Kun+14]	Ladicky [Lad+14]	Ros [Ros+15]	Sengupta [Sen+13]	Xu [Xu+13]	Zhang [Zha+15]	Total
Pas de fusion	Globale	76.11	71.36	73.48	76.80	73.25	86.07	77.30	77.35
	Moyenne	59.66	53.58	42.81	48.46	64.25	81.96	49.53	54.51
	IoU	49.65	39.80	31.74	38.60	44.32	70.83	37.53	41.99
Apprentissage joint	Globale	79.46	76.91	75.89	78.22	78.65	87.94	81.85	80.71
	Moyenne	64.42	60.01	**45.34**	50.23	71.70	84.37	53.10	58.62
	IoU	54.35	**46.01**	**34.08**	40.69	51.27	74.31	41.93	46.21
Apprentissage joint avec équilibrage des bases	Globale	**80.11**	**77.91**	**76.53**	77.43	80.42	87.84	81.77	80.84
	Moyenne	64.33	**59.66**	43.87	50.93	74.14	84.76	**53.90**	58.89
	IoU	55.24	45.65	33.95	39.37	53.24	74.19	42.14	46.32
Apprentissage joint avec partage du contexte	Globale	78.56	77.34	75.56	**78.53**	**81.34**	**88.19**	**84.04**	**81.75**
	Moyenne	**64.99**	59.37	44.92	**52.96**	**75.69**	**84.80**	53.79	**59.67**
	IoU	**57.83**	45.81	34.04	**40.98**	**54.83**	**74.62**	**43.86**	**47.10**

Tab. A.2.: Détails des précisions obtenues avec les différentes stratégies d'apprentissage sur l'ensemble des bases de données Stanford et des sept sous-ensembles de la base de données KITTI. Les meilleurs résultats sont affichés en gras.

Méthodes	Précisions	Apprentissage avec les 7 sous-ensembles de la bases KITTI et la base Stanford								
		He [HU13]	Kundu [Kun+14]	Ladicky [Lad+14]	Ros [Ros+15]	Sengupta [Sen+13]	Xu [Xu+13]	Zhang [Zha+15]	7 KITTI	Stanford [Gou+09]	Total
Pas de fusion	Globale	76.11	71.36	73.48	76.80	73.25	86.07	77.30	77.35	**74.80**	77.09
	Moyenne	59.66	53.58	42.81	48.46	64.25	81.96	49.53	54.51	64.35	55.64
	IoU	49.65	39.80	31.74	38.60	44.32	70.83	37.53	41.99	51.41	43.10
Apprentissage joint	Globale	78.80	77.00	75.92	**78.45**	78.32	88.26	80.83	80.40	73.38	79.64
	Moyenne	63.35	60.38	44.44	50.60	71.37	84.81	53.07	58.42	63.42	58.97
	IoU	53.44	46.38	33.65	41.07	50.37	74.90	41.10	45.93	50.51	46.43
Apprentissage joint avec reversement du gradient	Globale	78.89	77.16	75.62	78.33	79.39	87.99	81.76	80.73	72.42	79.82
	Moyenne	62.66	**60.74**	44.78	**51.16**	71.44	84.90	51.49	58.33	62.79	58.82
	IoU	53.31	**46.66**	33.59	**41.35**	51.37	74.59	40.90	46.08	49.31	46.43
Apprentissage joint avec équilibrage des bases	Globale	80.79	77.77	77.02	77.00	81.45	87.74	81.33	80.76	73.93	80.02
	Moyenne	64.88	60.01	44.68	49.40	74.79	84.72	51.72	58.65	63.87	59.23
	IoU	55.49	45.57	34.42	38.82	54.39	73.85	40.69	46.24	51.19	46.78
Apprentissage joint avec équilibrage des bases et reversement du gradient	Globale	81.41	77.34	77.52	77.02	79.50	88.26	82.65	81.20	74.39	80.46
	Moyenne	64.56	59.01	44.60	49.35	71.12	84.77	53.77	58.25	**65.31**	59.02
	IoU	55.48	44.95	34.60	39.77	51.67	74.79	42.30	46.28	**52.04**	46.91
Apprentissage joint avec partage du contexte	Globale	**82.13**	**78.17**	**77.88**	77.93	**82.91**	**88.38**	**82.82**	**81.26**	74.70	**81.08**
	Moyenne	**67.44**	60.02	**46.05**	50.57	**76.51**	**84.93**	**55.40**	**60.24**	63.44	**60.59**
	IoU	**58.09**	45.93	**35.65**	40.83	**56.19**	**74.98**	**43.31**	**47.92**	50.88	**48.24**

Tab. A.3.: Détails des précisions obtenues avec les différentes stratégies d'apprentissage sur l'ensemble des bases de données SIFTFlow et des sept sous-ensembles de la base de données KITTI. Les meilleurs résultats sont affichés en gras.

		Apprentissage avec les 7 sous-ensembles de la bases KITTI et la base SIFTFlow								
Méthodes	Précisions	He [HU13]	Kundu [Kun+14]	Ladicky [Lad+14]	Ros [Ros+15]	Sengupta [Sen+13]	Xu [Xu+13]	Zhang [Zha+15]	7 KITTI	SIFTFlow [Liu+11]	Total
Pas de fusion	Globale	76.11	71.36	73.48	76.80	73.25	86.07	77.30	77.35	69.79	76.72
	Moyenne	59.66	53.58	42.81	48.46	64.25	81.96	49.53	54.51	24.95	45.18
	IoU	49.65	39.80	31.74	38.60	44.32	70.83	37.53	41.99	19.02	34.73
Apprentissage joint	Globale	80.39	76.97	74.81	76.73	76.92	87.28	81.18	79.86	70.67	79.10
	Moyenne	**63.97**	61.18	43.47	48.27	66.96	84.05	50.89	57.05	**28.65**	48.08
	IoU	**54.31**	**46.43**	32.79	38.81	48.39	73.15	40.06	44.94	**21.44**	37.52
Apprentissage joint avec reversement du gradient	Globale	80.14	76.61	76.08	76.45	80.81	87.60	81.63	80.47	70.60	79.65
	Moyenne	63.82	60.04	43.31	48.37	**70.40**	83.80	49.58	57.10	28.56	48.08
	IoU	54.02	45.44	33.08	38.41	**52.47**	73.54	39.90	45.31	21.39	**37.76**
Apprentissage joint avec équilibrage des bases	Globale	79.60	74.84	74.26	74.24	75.71	86.68	81.05	78.93	**72.15**	78.37
	Moyenne	63.02	55.85	41.88	41.88	61.08	81.71	48.93	53.94	27.24	45.51
	IoU	52.69	41.70	31.05	31.05	47.13	71.08	39.45	42.63	21.12	35.84
Apprentissage joint avec équilibrage des bases et reversement du gradient	Globale	80.25	76.92	76.09	77.25	78.26	87.92	81.50	80.44	70.88	79.65
	Moyenne	63.56	**61.54**	**44.82**	**50.05**	69.26	84.42	**51.96**	**58.12**	27.11	**48.33**
	IoU	53.61	46.25	**33.86**	**39.64**	49.87	74.05	40.89	**45.56**	20.40	37.62
Apprentissage joint avec partage du contexte	Globale	**81.91**	**79.56**	**76.92**	**77.54**	**80.81**	**88.31**	**84.31**	**82.07**	70.89	**81.15**
	Moyenne	63.28	54.98	42.11	45.15	61.59	**84.69**	51.84	54.68	23.11	44.71
	IoU	54.21	43.62	32.92	36.60	49.45	**74.78**	**42.98**	44.87	17.76	36.31

Table des figures

Liste des tableaux

Liste des publications

Publications dans des journaux

Damien Fourure, Remi Emonet, Elisa Fromont, Damien Muselet, Natalia Neverova, Alain Trémeau, Christian Wolf. Multi-task, Multi-domain Learning : application to semantic segmentation and pose regression.
In *Neurocomputing* 2017

Publications dans des conférences internationales

Damien Fourure, Remi Emonet, Elisa Fromont, Damien Muselet, Alain Trémeau, Christian Wolf. Residual Conv-Deconv Grid Network for Semantic Segmentation.
In *British Machine Vision Conference (BMVC)* 2017

Damien Fourure, Remi Emonet, Elisa Fromont, Damien Muselet, Alain Trémeau, Christian Wolf. Semantic Segmentation via Multi-task, Multi-domain Learning.
In *joint IAPR International Workshops on Structural and Syntactic Pattern Recognition (SSPR 2016) and Statistical Techniques in Pattern Recognition (SPR)* 2016

Damien Fourure, Remi Emonet, Elisa Fromont, Damien Muselet, Alain Trémeau, Christian Wolf. Mixed pooling Neural Networks for Color Constancy.
In *International Conference on Image Processing (ICIP)* 2016

Publications dans des conférences nationales

Damien Fourure, Remi Emonet, Elisa Fromont, Damien Muselet, Alain Trémeau, Christian Wolf. Segmentation de scènes extérieures à partir d'ensembles d'étiquettes à granularité et sémantique variables.
In *RFP - Conférence Nationale en Reconnaissance des Formes et Intelligence Artificielle (RFIA)* 2016

Résumé La vision par ordinateur est un domaine interdisciplinaire étudiant la manière dont les ordinateurs peuvent acquérir une compréhension de haut niveau à partir d'images ou de vidéos numériques. En intelligence artificielle, et plus précisément en apprentissage automatique, domaine dans lequel se positionne cette thèse, la vision par ordinateur passe par l'extraction de caractéristiques présentes dans les images puis par la généralisation de concepts liés à ces caractéristiques. Ce domaine de recherche est devenu très populaire ces dernières années, notamment grâce aux résultats des réseaux de neurones convolutifs à la base des méthodes dites d'apprentissage profond. Aujourd'hui les réseaux de neurones permettent, entre autres, de reconnaître les différents objets présents dans une image, de générer des images très réalistes ou même de battre les champions au jeu de Go. Leurs performances ne s'arrêtent d'ailleurs pas au domaine de l'image puisqu'ils sont aussi utilisés dans d'autres domaines tels que le traitement du langage naturel (par exemple en traduction automatique) ou la reconnaissance de son. Dans cette thèse, nous étudions les réseaux de neurones convolutifs afin de développer des architectures et des fonctions de coûts spécialisées à des tâches aussi bien de bas niveau (la constance chromatique) que de haut niveau (la segmentation sémantique d'image). Une première contribution s'intéresse à la tâche de constance chromatique. Chez l'humain, cette dernière est la capacité du système visuel à percevoir des couleurs constantes pour une surface malgré les changements dans le spectre de l'illumination (changement d'éclairage). En vision par ordinateur, l'approche principale consiste à estimer la couleur de l'illuminant puis à supprimer son impact sur la couleur perçue des objets. Nous abordons la tâche de constance chromatique avec l'utilisation des réseaux de neurones en développant une nouvelle architecture composée d'un opérateur de sous-échantillonnage inspiré des méthodes traditionnelles existantes. Les expériences que nous avons menées montrent que notre méthode permet d'obtenir des performances compétitives avec l'état de l'art. Néanmoins, notre architecture requiert une grande quantité de données d'entraînement. Afin de corriger en parti ce problème et d'améliorer l'entraînement des réseaux de neurones, nous présentons plusieurs techniques d'augmentation artificielle de données. Nous apportons également deux contributions sur une problématique de haut niveau : la segmentation sémantique d'image. Cette tâche, qui consiste à attribuer une classe sémantique à chacun des pixels d'une image, constitue un défis en vision par ordinateur de par sa complexité. D'une part, elle requiert de nombreux exemples d'entraînement dont les vérités terrains sont coûteuses à obtenir. D'autre part, elle nécessite l'adaptation des réseaux de neurones convolutifs traditionnels afin d'obtenir une prédiction dite dense, c'est-à-dire, une prédiction pour chacun pixel présent dans l'image d'entrée. Pour résoudre la difficulté liée à l'acquisition de données d'entrainements, nous proposons une approche qui exploite simultanément plusieurs bases de données annotées avec différentes étiquettes. Pour cela, nous définissons une fonction de coût sélective qui a l'avantage de permettre l'entrainement d'un réseau de neurones convolutifs à partir de données issues de multiples bases de données. Nous développons aussi une approche dites d'auto-contexte capturant d'avantage les corrélations existantes entre les étiquettes des différentes bases de données. Finalement, nous présentons notre troisième contribution : une nouvelle architecture de réseau de neurones convolutifs appelée GridNet spécialisée pour la segmentation sémantique d'image. Contrairement aux réseaux traditionnels, implémentés avec un unique chemin allant de l'entrée (l'image) à la sortie (la prédiction), notre architecture est implémentée sous forme de grille 2D permettant à plusieurs flux interconnectés de fonctionner à différentes résolutions. Afin d'exploiter la totalité des chemins de la grille, nous proposons une technique d'entraînement inspirée du dropout. En outre, nous montrons empiriquement que notre architecture généralise de nombreux réseaux bien connus de l'état de l'art. Nous terminons par une analyse des résultats empiriques obtenus avec notre architecture qui, bien qu'entraînée avec une initialisation aléatoire des poids, révèle de très bonnes performances, dépassant les approches populaires souvent pré-entraînés.

Abstract Computer vision is an interdisciplinary field that investigates how computers can gain a high level of understanding from digital images or videos. In artificial intelligence, and more precisely in machine learning, the field in which this thesis is positioned, computer vision involves extracting characteristics from images and then generalizing concepts related to these characteristics. This field of research has become very popular in recent years, particularly thanks to the results of the convolutional neural networks that form the basis of so-called deep learning methods. Today, neural networks make it possible, among other things, to recognize different objects present in an image, to generate very realistic images or even to beat the champions at the Go game. Their performance is not limited to the image domain, since they are also used in other fields such as natural language processing (e. g. machine translation) or sound recognition. In this thesis, we study convolutional neural networks in order to develop specialized architectures and loss functions for low-level tasks (color constancy) as well as high-level tasks (semantic segmentation). Color constancy, is the ability of the human visual system to perceive constant colours for a surface despite changes in the spectrum of illumination (lighting change). In computer vision, the main approach consists in estimating the color of the illuminant and then suppressing its impact on the perceived color of objects. We approach the task of color constancy with the use of neural networks by developing a new architecture composed of a subsampling operator inspired by traditional methods. Our experience shows that our method makes it possible to obtain competitive performances with the state of the art. Nevertheless, our architecture requires a large amount of training data. In order to partially correct this problem and improve the training of neural networks, we present several techniques for artificial data augmentation. We are also making two contributions on a high-level issue : semantic segmentation. This task, which consists of assigning a semantic class to each pixel of an image, is a challenge in computer vision because of its complexity. On the one hand, it requires many examples of training that are costly to obtain. On the other hand, it requires the adaptation of traditional convolutional neural networks in order to obtain a so-called dense prediction, i. e., a prediction for each pixel present in the input image. To solve the difficulty of acquiring training data, we propose an approach that uses several databases annotated with different labels at the same time. To do this, we define a selective loss function that has the advantage of allowing the training of a convolutional neural network from data from multiple databases. We also developed self-context approach that captures the correlations between labels in different databases. Finally, we present our third contribution : a new convolutional neural network architecture called GridNet specialized for semantic segmentation. Unlike traditional networks, implemented with a single path from the input (image) to the output (prediction), our architecture is implemented as a 2D grid allowing several interconnected streams to operate at different resolutions. In order to exploit all the paths of the grid, we propose a technique inspired by dropout. In addition, we empirically demonstrate that our architecture generalize many of well-known state-of-the-art networks. We conclude with an analysis of the empirical results obtained with our architecture which, although trained from scratch, reveals very good performances, exceeding popular approaches often pre-trained.