

Rapport de projet de fin d'études

Effectué chez Hewlett Packard Enterprise

Transformers pour la vision par ordinateur

*Pourquoi devrions-nous utiliser des mécanismes basés sur l'attention
en vision par ordinateur ?*

Hugo CARTIGNY
3^{ème} année en alternance – Double diplôme MSIAM
13 février 2023 – 31 août 2023

HPE
5 Rue Raymond Chanas
38320 Grenoble

Responsable de stage
Bruno MONNET
Tuteur de l'école
Olivier FRANÇOIS

Résumé

L'arrivée des Transformers pour le traitement du langage naturel a permis le développement de modèles génératifs fondamentaux tels que GPT-4, plus connu sous sa version conversationnelle ChatGPT. Le domaine de la vision par ordinateur a lui aussi bénéficié de cette avancée, sous le nom de **vision transformers**, offrant alors une alternative aux couches de convolution, ces dernières ayant déjà fait leurs preuves. Nous nous posons alors une simple question: *pourquoi devrions-nous utiliser des mécanismes basés sur l'attention en vision par ordinateur ?*

Pour y répondre, nous allons nous intéresser au coût énergétique lié à l'entraînement de ces différentes architectures appliqué à la tâche de la **segmentation sémantique**, et verrons qu'il est même possible de les combiner. Nous détaillerons dans un premier temps le protocole mis en place permettant de comparer les entraînements, puis nous étudierons leur scalabilité sur plusieurs nœuds de calcul en utilisant **HPE MLDE**.

(Enfin, si le temps le permet, nous étudierons la performance de MLDE comparé à une pile logicielle **Alpa + Jax + Ray** en terme d'efficacité énergétique).

Dans le cadre fixé par ce projet de fin d'études, nous verrons notamment qu'un modèle basé sur des couches de convolution nécessite moins d'énergie pour atteindre une précision (*accuracy*) cible fixée qu'un modèle basé sur des Transformers, et tenterons d'expliquer ces écarts en formulant des hypothèses.

Remerciements

Je tiens avant tout à remercier Bruno Monnet qui a été mon maître d'apprentissage durant ces 3 années d'alternance à l'Ensimag ; il m'a accompagné dans cette découverte du monde de l'entreprise et m'a introduit au monde fabuleux du calcul hautes performances associé à l'intelligence artificielle en me faisant rejoindre une équipe qui regorge de talent.

Je souhaite exprimer ma gratitude à Olivier François, mon tuteur pédagogique côté école qui m'a suivi durant ce projet, qui m'a suggéré de rédiger ce document comme un rapport scientifique : j'ai trouvé cela très adapté à mon sujet et j'ai pu découvrir une méthode de rédaction que je pourrai sans aucun doute réutiliser dans le futur !

Je souhaite remercier tous les collègues avec lesquels j'ai pu travailler : Hana Malha, Frederic Ciesielski, Jean Pourroy, Sébastien Cabaniols, Philippe Vettori, Scott Johnson ainsi que ma manager Nathalie Viollet, sans oublier mon coach de l'ombre Amar Zitouni.

Je remercie également mes confrères apprentis et stagiaires qui m'ont accompagné durant mon parcours, dans l'ordre : Nicolas Guerroudj, Tom Capelle, Théo Rozier, Tibault Dorn, Arvind Candassamy, Clémence Mayjonade, Arnaud Hincelin, Emilien Tellier, Sari Boussouar et Nicolas Diniz.

Une pensée particulière à mes camarades de promotion : Tanguy Paymal, Gaspard Andrieu, Thomas Vincent, Romain Lopez mais aussi tous les autres que j'oublie (trop de monde !).

Je remercie évidemment ma famille pour leur soutien moral durant ce projet, aussi bien pendant les bonnes que les mauvaises phases : leur présence et leurs encouragements ont été très importants pour moi.

Dans le cadre de ce rapport, je remercie Jean Pourroy et Hana Malha pour leur aide et précieux conseils, ainsi que Jean pour sa thèse dont je me suis inspiré du style !

Enfin, je remercie le projet Typst [Mäd22] d'avoir vu le jour, proposant un fantastique outil en remplacement de L^AT_EX pour la rédaction de cet ouvrage!

Table des matières

Légende	6
Glossaire	7
Acronymes	11
1. Introduction	13
1.1. Qu'est-ce que le deep learning ?	13
1.2. Contexte	14
1.3. Transformers	15
1.4. Couches de convolution	16
1.5. Entreprise	17
1.6. Equipe	18
1.7. Segmentation sémantique	18
1.8. Objectifs	18
1.9. HPE MLDE	19
1.10. Livrables	19
1.11. Plan du document	20
2. Revue de l'état de l'art	21
2.1. Quelques concepts	21
2.2. Modèles basés sur des convolutions	23
2.3. Modèles basés sur des Transformers	26
2.4. Modèles hybrides convolutions/Transformers	28
2.5. Modèles choisis	28
3. Méthodologie	31
3.1. Choix préalables	31
3.2. Facteurs de comparaison	32
3.3. Choix de l'environnement	34
3.4. Challenges	35
3.5. Pour aller plus loin	36
4. Résultats	37
4.1. Contraintes technologiques	37
4.2. Entraînements	39
5. Discussions	43
5.1. Forces de l'approche	43
5.2. Faiblesses de l'approche	43
5.3. Notre approche répond-elle à la problématique ?	44
5.4. Pistes de poursuites	44
5.5. Impact environnemental et sociétal	44
6. Conclusions	47
7. Bibliographie	49

Légende

Ce document utilise les conventions suivantes : en prendre connaissance vous permettra de rendre votre lecture plus agréable !

- **Gras**: Référence cliquable au glossaire ou à la liste d'acronymes
- *Italique*: mot(s) en anglais
- Souligné: lien vers une ressource externe (e.g. un site web) cliquable
- Figure 1: référence à une figure dans le document, cliquable
- [\[Xie+21\]](#): référence à une publication scientifique, cliquable pour accéder à l'entrée de la bibliographie correspondante

Glossaire

computer vision	Vision par ordinateur. Branche de l'intelligence artificielle qui traite de la façon dont les ordinateurs peuvent acquérir une compréhension de haut niveau à partir d'images ou de vidéos numériques. 9 , 14 , 16 , 18 , 19 , 26 , 47
CUDA	CUDA est une technologie permettant de programmer des GPUs en C++ (langage de programmation) pour exécuter des calculs généraux à la place du processeur central (CPU). Elle est développée par NVIDIA pour ses cartes graphiques, et utilise un pilote unifié utilisant une technique de streaming (flux continu). 9 , 37 , 41
benchmark	Code ou ensemble de codes permettant de mesurer la performance d'une solution et d'en vérifier ses fonctionnalités. 18 , 35
bottleneck	Goulot d'étranglement. Dans un code HPC, point limitant le calcul car représentant la partie la plus lente, retardant le reste des composants interagissant avec. 32
cluster	Grappe de serveurs sur un réseau, ferme ou grille de calcul. 19
conteneur	Enveloppe virtuelle permettant de distribuer une application avec tous les éléments dont elle a besoin pour fonctionner (fichiers, environnements, bibliothèques...). Un conteneur peut fonctionner sur la plupart des systèmes utilisant le noyau Linux car il le partage à l'exécution, n'impactant donc pas les performances. Cela permet de reproduire très facilement une expérience sur un autre système, en plus de pouvoir l'isoler du reste du système pendant son exécution. 39
couche de convolution	Type de couche d'un modèle de deep learning , permettant de calculer une sortie en déplaçant un <i>kernel</i> (matrice carrée) sur une matrice d'entrée (i.e. en effectuant une convolution sur l'entrée). A ne pas confondre avec un CNN (<i>Convolutional Neural Network</i> , qui désigne un modèle complet composé, entre autres, de couches de convolution). 8 , 16 , 17 , 28 , 41 , 47
deep learning	<u>Apprentissage profond</u> . Sous-domaine de l'intelligence artificielle qui utilise des réseaux neuronaux pour résoudre des tâches complexes, grâce à des architectures articulées de différentes transformations non linéaires. 7 , 8 , 9 , 11 , 13 , 22 , 37 , 43

Determined.AI	Rachetée par HPE en 2021, entreprise développant une solution open-source du même nom permettant de paralléliser des entraînements de modèles sur un cluster de calcul. 19 , 39
entraînement	En deep learning , l'entraînement consiste à estimer un modèle à-partir d'observations en nombre fini (un jeu de données) dans le but de résoudre une tâche. Cette phase est également appelée « apprentissage », et est réalisée avant la phase d' inférence qui consiste en son exploitation. 13 , 20
IA générative	Ensemble des techniques d'intelligence artificielle permettant de générer du contenu, le plus souvent du texte, des images ou des médias à-partir de prompt . 13 , 14
inférence	La phase d'inférence représente l'exécution d'un modèle une fois qu'il a été entraîné. Pour cela, il est possible d'appliquer de nombreuses optimisations afin d'augmenter les performances lors de la mise en production, comme de la quantification qui consiste à réduire le nombre de bits sur lesquels les paramètres du modèle sont stockés afin de pouvoir en traiter plus en parallèle. 8 , 16 , 28
max-pooling	En deep-learning, couche fonctionnant de la même manière qu'une couche de convolution mais sans paramètres ; son <i>kernel</i> $n \times n$ effectue l'opération $\max()$ sur les pixels qu'il traite : cela a alors pour effet d'effectuer une « moyenne » de l'image en gardant les valeurs les plus fortes par région. Cette opération est souvent employée pour réduire la taille de l'image/d'un tenseur en réduisant le coup calculatoire. 25
modèle	En deep learning, également appelé réseau de neurones; formé de couches parallèles et/ou successives contenant des paramètres entraînables. 7 , 8 , 9 , 11 , 13 , 14 , 19 , 24
neurone	Représentation mathématique simplifiée d'un neurone physique, représenté par $y = \text{activation}(w \cdot x + b)$, où w et b sont des paramètres entraînables, et activation est une fonction non-linéaire (par exemple $\text{ReLU}(x) = \max(0, x)$).
prompt	Invite sous forme de texte, phrase ou instruction donnée à un modèle . 8 , 27
PyTorch	Bibliothèque logicielle basée sur le langage de programmation Python permettant de réaliser des calculs tensoriels nécessaires notamment pour le deep learning , en utilisant le CPU et le GPU, supportant

CUDA. Initialement développé par Facebook (Meta) et confié fin 2022 à la fondation Linux. [9](#), [34](#), [37](#), [38](#)

segmentation sémantique	Sous-domaine de la computer vision , consistant à attribuer une classe (chien, chat, route, voiture...) à chaque pixel d'une image, sans pour autant pouvoir différencier plusieurs instances d'une même classe. 11 , 18 , 28 , 31
tenseur	Objet mathématique. Un tenseur est un tableau multidimensionnel qui est une sorte de généralisation du concept de matrice ou de vecteur ; il possède une ou plusieurs dimensions (une « forme ») et les données qu'il contient ont toutes le même type (entiers, flottants...). 22
TensorFlow	Bibliothèque logicielle développée par Google, similaire et concurrente à PyTorch . Elle propose aussi des composants plus haut niveau (plus d'abstraction) et orientée objet, ce qui la rend plus accessible mais permet à PyTorch d'être généralement préférée pour le monde de la recherche. 38
Transformer	Type de couche d'un modèle de deep learning , basé sur un mécanisme dit de « <i>self-attention</i> » permettant de calculer une sortie en associant un contexte à une entrée (e.g. un mot). Plus d'informations dans l'introduction aux Transformers. 14 , 15 , 16 , 17 , 26 , 28 , 41 , 44 , 47

Acronymes

FLOP	<i>Floating Point Operation</i> ; opération à virgule flottante 14 , 44
FLOPS	<i>Floating Point Operation per Second</i> ; nombre d'opérations à virgule flottante par seconde (FLOP/s). ⚠ Ne pas confondre avec FLOPs (plusieurs FLOP)
GPU	<i>Graphics Processing Unit</i> ; processeur graphique. Carte physique de calcul massivement parallèle, particulièrement utile pour accélérer les entraînement de modèles en deep learning , ces derniers étant principalement constitués de multiplications matricielles 14 , 19 , 28 , 32 , 33 , 37 , 41
HPC	<i>High Performance Computing</i> ; calcul haute performance 36 , 43 , 47
HPE	Hewlett Packard Enterprise 17
LLM	<i>Large Language Model(s)</i> ; modèle(s) de langage de grande taille. Généralement de plusieurs dizaines voire centaines de milliards de paramètres 13 , 14 , 17
mIoU	<i>mean Intersection over Union</i> ; intersection sur l'union moyenne. Aussi connue sous le nom d' <u>indice de Jaccard</u> , cette métrique permet de mesurer la distance entre deux répartitions de classes au sein d'une image en segmentation sémantique 22 , 23 , 39
MLDE	Machine Learning Development Environment. Nom commercial de la solution open-source <u>Determined.AI</u> (voir glossaire) 19 , 35 , 38 , 43 , 44
NLP	<i>Natural Language Processing</i> ; traitement automatique du langage naturel 14 , 16 , 27 , 33 , 47
PFE	projet de fin d'études 18
SOTA	<i>State-of-the-art</i> ; état de l'art 14

1. Introduction

L'arrivée récente des **modèle(s) de langage de grande taille** (LLM) et de l'**IA générative** dans le grand public a bouleversé nos repères : en quelques semaines seulement, une grande partie de la population s'est émerveillée devant les réponses impressionnantes de ChatGPT (OpenAI) et l'art, parfois même photoréaliste, généré avec Midjourney, Stable Diffusion ou encore Dall-E 2, à-partir d'une simple phrase.

Ces technologies, désormais populaires, ont un point commun : ce sont des **modèles** de **deep learning** qui ont été entraînés sur des gigantesques jeux de données.

1.1. Qu'est-ce que le deep learning ?

Le deep learning est un domaine où l'on essaye de résoudre un problème en « entraînant » un modèle. Le modèle est une structure comportant plusieurs couches parallèles et/ou successives, formant une suite d'opérations mathématiques, semblable à une multiplication de plusieurs matrices comportant chacune des « paramètres ». Ces paramètres viennent transformer l'entrée (numérique) du modèle, donnant alors une représentation intermédiaire qui sera passée en entrée à la couche d'après. La dernière couche fournit alors un résultat interprétable. En fonction de l'exactitude du résultat, une rétropropagation du gradient de l'erreur va être faite sur les paramètres du modèle, par itérations, afin de tendre vers un meilleur résultat la fois suivante : c'est ce que l'on appelle l'**entraînement**.

Lors de l'entraînement, les réponses du modèle sont confrontées à celles pré-établies du jeu de données. Un jeu de données associe un lot d'entrées à un lot de sorties correspondantes.

Il est également possible d'imaginer le deep learning comme suit : lorsque le processus associant une sortie à une entrée est connu, il est possible de le programmer, avec du code par exemple. Si cependant le processus n'est pas connu mais qu'il est possible d'établir un jeu de données listant des entrées et leurs sorties, en suffisamment grand nombre, alors il est possible de recourir à des méthodes comme du machine learning ou du deep learning.

Le deep learning est un sous-ensemble du machine learning qui repose, comme vu ci-dessus, sur l'apprentissage de représentations de données par opposition à des algorithmes qui sont, eux, plus spécifiques à une tâche.

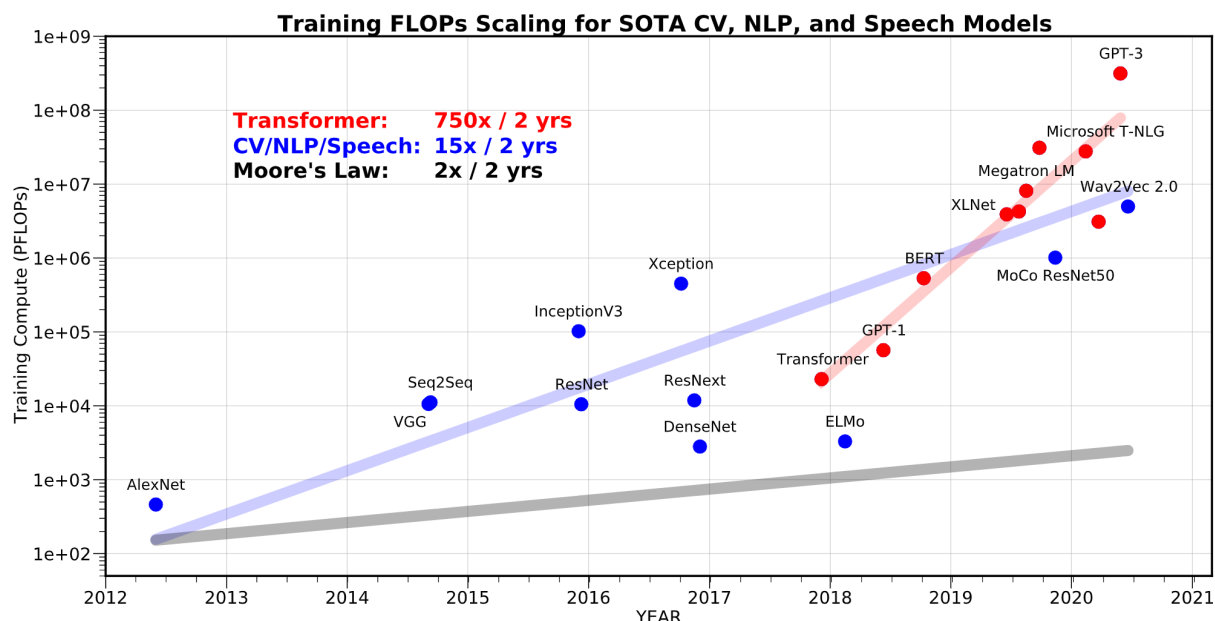


Figure 1: Evolution du nombre de **FLOPs** (Floating Point Operations) requis pour l'entraînement de **modèles de l'état de l'art** (SOTA) en **computer vision**, **traitement automatique du langage naturel** (NLP) et **speech** (synthèse de voix) ([source](#))

1.2. Contexte

Depuis l'arrivée des transformers, la taille des **LLMs** n'a cessé d'augmenter, de manière globalement exponentielle mais très largement supérieure à la loi de Moore¹. Cette dernière suit approximativement l'évolution de la puissance de calcul disponible (nombre de transistors sur un microprocesseur) et montre que celle-ci double environ tous les 2 ans. Comme visible sur la Figure 1 les LLMs, eux, voient leur puissance calculatoire requise multipliée par plus de 10 chaque année depuis 2018 pour un total d'environ 750 en 2 ans, à titre comparatif. Cela montre bien une chose, outre le fait qu'une limite va être atteinte tôt ou tard : les moyens de calculs requis deviennent de plus en plus coûteux et l'on va, dans ce contexte, vite devoir parler de supercalculateur lors de l'entraînement d'un (très) grand modèle.

Nous avons ici pris l'exemple des **LLMs** afin de mettre en avant leurs tailles spectaculaires (parfois jusqu'à plus d'1T (10^{12} , mille milliards) de paramètres !), mais cela est en fait applicable à tout le domaine de l'**IA générative**. Ceci résulte en des modèles dont l'entraînement nécessite toujours plus de **processeurs graphiques (GPUs)**.

Cette politique de « toujours plus » est permise notamment grâce à une technologie qui se met très bien à l'échelle lorsqu'elle est parallélisée : le **Transformer**.

¹Nous ne nous intéressons pas ici à l'exactitude de la loi de Moore mais à la différence d'ordre de grandeur dans les évolutions

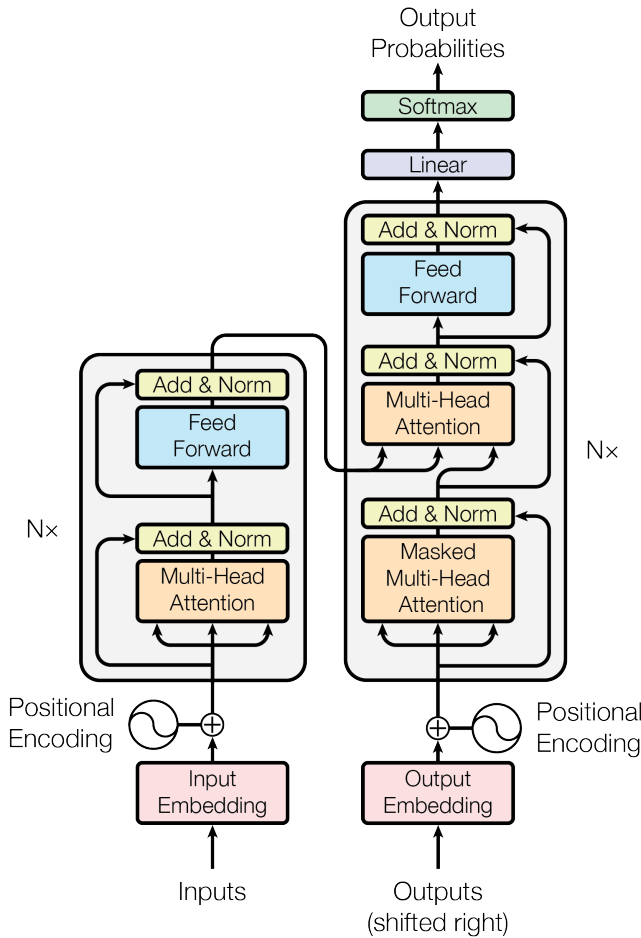


Figure 2: Architecture d'un modèle
Transformer
(source: [Vas+23])

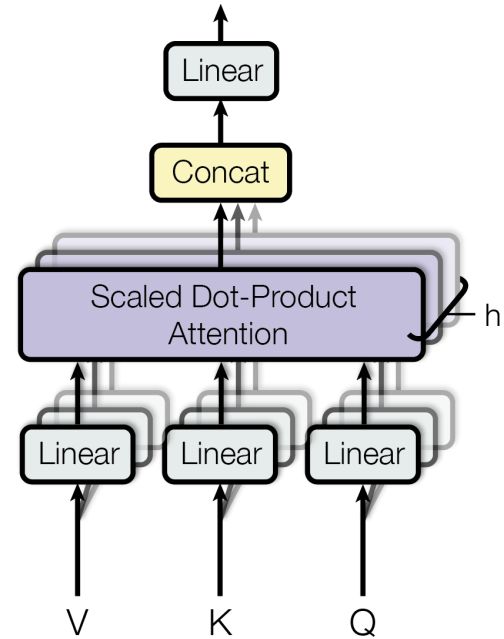


Figure 3: Architecture du mécanisme
de *multi-head attention*
(source: [Vas+23])

1.3. Transformers

Les Transformers ont été introduits et popularisés par le papier *Attention Is All You Need* [Vas+23]. Ils sont inspirés de travaux précédents sur l'attention [BCB16] [Kim+17] [Par+16] et comportent plusieurs mécanismes intéressants ; une des notions principales est la *multi-head attention* (attention multi-têtes) qui est basée sur le mécanisme de *self-attention* (auto-attention) : l'idée est de permettre au modèle de formuler des « requêtes » d'un mot par rapport aux autres dans la phrase, et de former ainsi des vecteurs de contexte. Par exemple, dans la phrase « Cette pastèque est sucrée », le vecteur de contexte de **sucrée** donnerait beaucoup d'importance à **pastèque**, mot auquel il se rapporte.

La Figure 2 illustre l'architecture d'un modèle Transformer, où l'on peut isoler 2 types de blocs (en gris) : à gauche l'*encoder* qui va, dans un contexte de traduction de français vers anglais par exemple, venir « encoder » la phrase originale en français ; à droite, le *decoder* qui va recevoir les informations de l'*encoder* tout en se référant à la portion de phrase déjà traduite en anglais, dans le but de prédire quel sera le prochain mot : on parle alors de « *masked* » *multi-head attention*.

Comme illustré dans la Figure 3, le mécanisme de *multi-head attention* est constitué de 3 couches denses mises en parallèle, dans le but d'imiter un système similaire à une base de données, permettant d'effectuer des « requêtes » entre les différents mots de la phrase.

En effet, les Transformers ont la particularité d'être parallélisables à l'entraînement, c'est à dire que l'on va pouvoir prédire n mots à la fois en masquant la partie de la phrase que le modèle n'est pas sensé connaître dans chaque partie concurrente. Durant la phase d'**inférence** cependant, c'est-à-dire la phase où l'on va tester le modèle afin de l'utiliser sur des nouvelles données, l'exécution sera forcément séquentielle car il faudra générer le mot $n - 1$ pour générer le mot n .

En pratique, les mots ne sont pas générés ou utilisés tels quels directement, ils sont découpés en *tokens*. Par exemple **bonjour**, étant lui-même un mot peu commun, pourrait être découpé en **bon ##jour** : **bon** et **jour** sont deux radicaux potentiellement plus fréquents (le préfixe **##** désigne la suite d'un mot). Cela permet par exemple d'avoir **jour**, puis **jour ##s** une fois au pluriel !

A la base présenté comme un outil dans le domaine du **NLP**, la communauté scientifique a essayé (et essaye encore, à l'heure où je rédige ces mots) d'appliquer les **Transformers** à d'autres domaines que le texte comme la **vision**, pourtant à l'origine dominée par des **couches de convolution** ou dérivé(e)s.

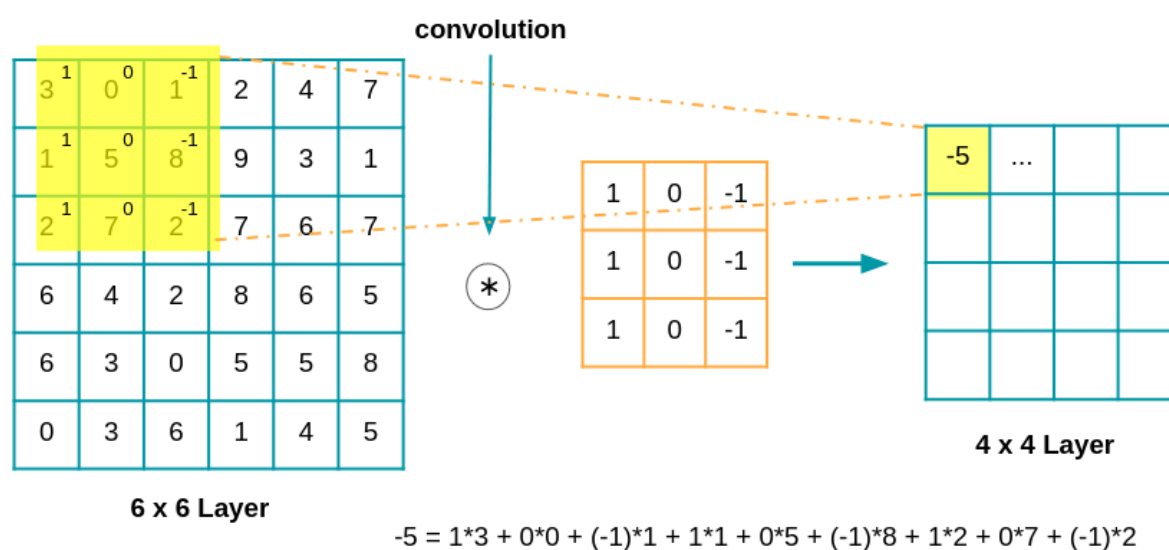


Figure 4: Convolution sur une matrice 6x6 avec un kernel de taille 3x3 ([source](#))

1.4. Couches de convolution

Les **couches de convolution** effectuent une convolution sur l'entrée qui leur est fournie, comme illustré ci-dessus dans la Figure 4. Pour cela un *kernel*, de taille arbitraire, est appliqué en haut à gauche, puis chacun de ses paramètres est multiplié par la valeur qu'il recouvre ; chacun des 9 résultats sont en suite additionnés ensemble pour donner, ici, un résultat de -5 . Le kernel est ensuite déplacé vers la droite et vers le bas de sorte

à ce que toutes les cases de la matrice de résultat aient été remplies. En déplaçant un kernel de 3x3 sur une matrice 6x6, nous avons une matrice 4x4 en résultat, à cause du kernel ne pouvant pas être centré sur la case en haut à gauche. Pour prévenir cela, il est généralement appliqué un *padding*, qui peut par exemple être constitué d'une bande de zéro autour de la matrice d'entrée, permettant ainsi de garder une matrice 6x6 en sortie.

Les couches de convolution sont particulièrement adaptées au traitement d'images car elles permettent de traiter des pixels dans leur contexte. En regardant les couches de convolution d'un modèle après entraînement, on s'aperçoit qu'elles détectent des motifs, plutôt simples dans les premières couches, mais qui deviennent de plus en plus complexes au fur-et-à-mesure qu'elles s'enchaînent. Comme d'habitude en deep learning, les séries de transformations non-linéaires vont permettre de détecter des motifs complexes en se combinant.

Nous aurons l'occasion de reparler de certains détails concernant les **Transformers** et les **couches de convolution** par la suite, le but était ici de donner l'intuition de leur fonctionnement.

Ayant vu ces éléments, nous pouvons d'ores et déjà nous poser la question suivante : **devons-nous utiliser les Transformers, les couches de convolution ou un mix des deux ?** Nous allons voir ci-après d'autres éléments nous permettant de préciser cette question.

1.5. Entreprise

Hewlett Packard Enterprise (HPE), née de la séparation d'HP en 2 entités en 2015, a hérité de la partie services aux entreprises avec notamment la vente de supercalculateurs, qui apparaissent d'ailleurs très régulièrement au Top 500. La performance est le cœur de métier d'HPE, et c'est là que l'expertise de l'entreprise vient croiser celle de ses clients : en effet, lors de la livraison d'un supercalculateur une *acceptance* doit être passée, venant démontrer la conformité de la machine ainsi livrée, en termes de performance brute sur des tâches choisie par le client comme critères.

Etant donné que nous parlons de supercalculateurs, cela signifie que les charges de travail sont à grande échelle et doivent donc être optimisées. L'entraînement d'un **LLM** émet l'équivalent carbone (CO₂eq) de 25 à 50 allers-retours Paris-New York en avion pour 1 passager², selon le pays³. D'autre part, celui-ci coûte cher en électricité : de 400 à 1500 MWh en moyenne [LVL22], soit de 3 à 10 millions de kilomètres parcourus en Tesla Model 3⁴ !

²Source : <https://eco-calculateur.dta.aviation-civile.gouv.fr>

³Source : <https://app.electricitymaps.com/zone/FR>

⁴Source : https://www.tesla.com/fr_ch/support/european-union-energy-label

1.6. Equipe

J’ai eu la chance de réaliser mon alternance ainsi que ce projet de fin d’études dans l’équipe HPC & AI (*High Performance Computing and Artificial Intelligence*, Calcul Haute Performances et Intelligence Artificielle), qui se situe dans le centre de compétences de Grenoble qui possède une expertise mise à disposition de la zone EMEA (*Europe, Middle East and Africa*).

Le site de Grenoble est l’un des rares sites à disposer d’un lab, c’est-à-dire d’un petit datacenter dans lequel du matériel est constamment testé (avec des **benchmarks**) et mis à disposition de clients, nous permettant ainsi d’effectuer des tests à grande échelle.

L’équipe est plutôt orientée HPC, mais travaille aussi bien sur les parties entraînement qu’inférence côté IA. De mon côté, ce **projet de fin d’études** (PFE) s’articule principalement sur la partie entraînement IA car c’est dans cette phase que les calculs massivement parallélisés interviennent et nécessitent des supercalculateurs.

Nous avons choisi la **segmentation sémantique** car il s’agit d’un domaine pour lequel nous avons déjà un modèle de référence. Ce domaine a le potentiel d’apporter des connaissances pouvant servir pour diverses applications par la suite, comme dans la conduite autonome ou tout simplement en tant que **benchmark** pour l’équipe, afin de profiler un système pour une tâche clé dans le futur.

1.7. Segmentation sémantique

La segmentation sémantique est un sous-domaine de la **computer vision**, consistant à attribuer une classe (chien, chat, route, voiture...) à chaque pixel d’une image, formant ainsi des sortes de masques à la fin étant donné que les pixels proches ont tendance à avoir la même classe. La segmentation sémantique ne permet pas de différencier deux chiens sur une image mais les identifiera tous deux comme « du chien »⁵, contrairement à la segmentation d’instances qui, elle, saura différencier les deux chiens mais sans savoir qu’ils sont tous les deux des chiens ; seulement qu’ils sont différents : on dit qu’ils représentent différentes instances.

La combinaison de la segmentation sémantique et d’instances existe : la segmentation panoptique, qui attribue une classe et une instance à chaque pixel.

Comme vu précédemment, le modèle que nous possédons permet de faire de la segmentation sémantique, c’est pourquoi c’est la tâche que nous choisissons.

1.8. Objectifs

Le Top500 classe deux fois par an les 500 supercalculateurs les plus puissants de la planète. Le Green500 classe les supercalculateurs du Top500 selon un critère d’efficacité énergétique : dans la liste de juin 2023, les premiers développent plus de 60 GFLOPs/Watt. On constate un rapport de 4 entre le 1^{er} et le 50^{ème} de la liste et de 14 entre le 1^{er} et

⁵Citation d’un grand philosophe, Frederic Ciesielski

le 100^{ème}. D'autre part, les 50 premiers systèmes comportent des accélérateurs (comme des **GPUs**). Dans ce contexte, il est alors intéressant de vouloir réduire sa consommation, une fois la machine déployée en phase de production. Le pays d'installation a alors de l'importance quant au type d'énergies utilisées pour fabriquer de l'électricité, comme le charbon en Allemagne. Même si des progrès peuvent probablement être faits du côté de la fabrication du matériel, cette étude s'adresse à la partie logicielle qui a sans doute un fort impact durant la vie du supercalculateur.

Notre objectif est ainsi de comparer l'efficacité énergétique des deux architectures de modèles, Transformers et convolutions, pour un problème donné de **computer vision**. Cela nous permettra par exemple de déterminer l'impact du mécanisme d'attention (*self-attention*) des Transformers. Pour cela, des critères communs doivent également être définis : je choisis, entre autres, la précision (*accuracy*) du modèle, et le jeu de données (nous détaillerons les points communs dans la méthodologie). Ensuite, des modèles représentatifs de l'état de l'art devront être choisis.

Nous pourrions profiter de la scalabilité (mise à l'échelle) des modèles basés sur les Transformers ou les convolutions quand ceux-ci sont parallélisés sur plusieurs nœuds avec **HPE Machine Learning Development Environment** (MLDE).

1.9. HPE MLDE

Hewlett Packard Enterprise Machine Learning Development Environment est une plateforme développée par **Determined.AI**, déployée sur un **cluster** de calcul, permettant de faire de l'entraînement distribué de **modèles**, c'est-à-dire sur plusieurs unités de calcul (ici des **GPUs**) en parallèle. Paralléliser un entraînement est nécessaire dans certains cas si le modèle est trop gros pour un seul **GPU**, et permet d'une manière générale d'accélérer l'entraînement. L'entraînement d'un modèle peut être parallélisé de plusieurs manières : par exemple en disposant d'une copie du modèle par accélérateur (GPU) mais en fournissant à chaque instance des données différentes, ou en séparant les couches du modèle en plusieurs parties puis en les plaçant chacune sur un GPU différent. Cette dernière méthode introduit des communications supplémentaires et est utile notamment quand le modèle est trop grand pour la mémoire d'un seul accélérateur. MLDE propose également de l'*hyperparameter tuning*, soit de l'optimisation d'hyperparamètres : ces derniers sont des paramètres globaux du modèle (nombre de couches, leur taille, etc.) afin de trouver la configuration optimale ; l'idée est de représenter les différentes valeurs des hyperparamètres comme un espace n -dimensionnel et de le parcourir avec une méthode efficace [Li+20], limitant les calculs inutiles.

Enfin, pour conclure cette liste des objectifs, si le temps l'avait permis j'aurais aimé étudier les différences de performances entre **MLDE** et un pile logicielle constituée d'Alpa, JAX et Ray, dont nous discuterons brièvement dans les pistes de poursuites.

1.10. Livrables

Nous souhaitons qu'il ressorte de ce projet de fin d'études plusieurs livrables. Premièrement, ce document lui-même comportera une méthodologie détaillant comment estimer

l'énergie consommée par l'**entraînement** d'un modèle. Deuxièmement, une seconde méthodologie viendra compléter la première en montrant comment étudier la scalabilité (mise à l'échelle) en étudiant l'entraînement d'un modèle. Troisièmement, les recettes d'entraînement pour chacun des modèles seront fournies, compatibles directement avec HPE MLDE si possible.

1.11. Plan du document

Dans un premier temps nous passerons en revue l'état de l'art dans le contexte de cette étude, puis nous introduirons la méthodologie. Nous présenterons ensuite nos résultats en les analysant, et effectueront une critique de ce projet de fin d'études. Enfin, nous concluerons.

2. Revue de l'état de l'art

2.1. Quelques concepts

Cette partie présente des outils du domaine, utiles pour comprendre l'état de l'art avec plus de détails.

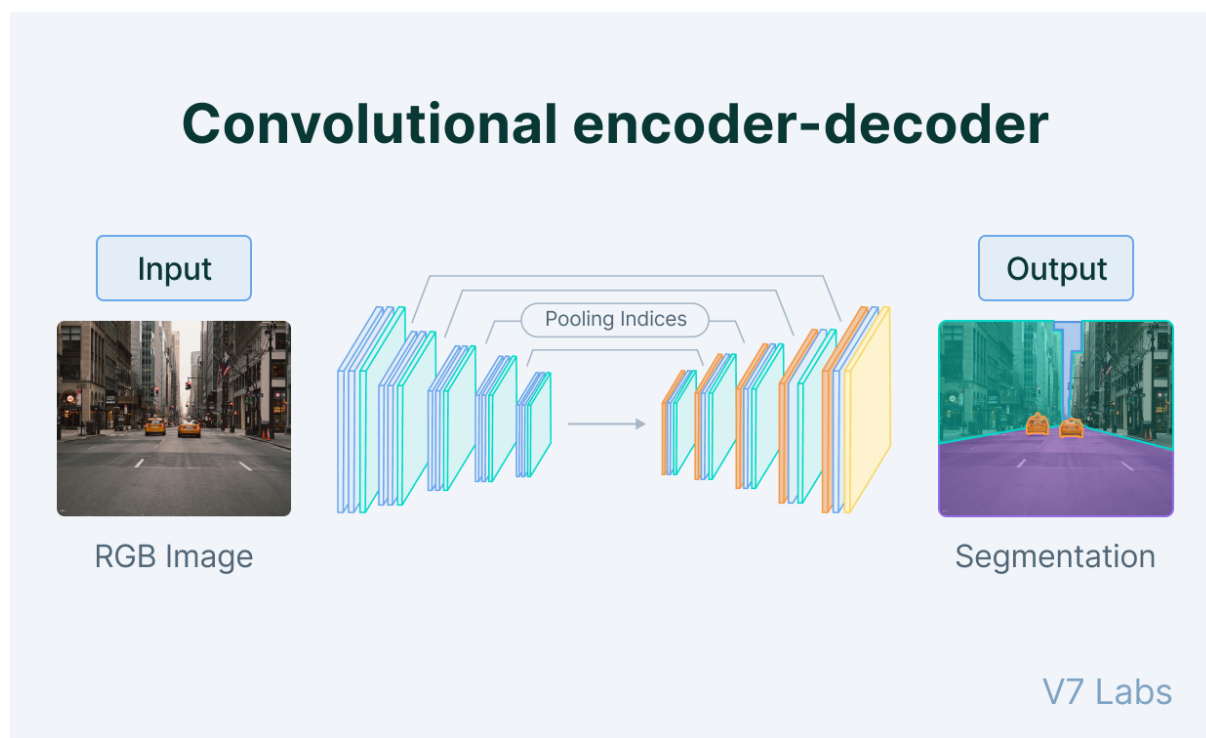


Figure 5: Architecture *encoder-decoder* (encodeur-décodeur) d'un CNN (réseau neuronal de convolutions), c'est-à-dire contenant des couches de convolution successives ([source](#))

Habituellement, la classification d'images s'effectue globalement en faisant se succéder des couches de convolution, que l'on connecte ensuite à une ou plusieurs couches denses (neurones interconnectés). Le processus permet d'établir une nouvelle représentation de l'image en extrayant ses composantes principales, puis en les matérialisant en classes dans les dernières couches. Ainsi, une architecture basique qui était utilisée en segmentation sémantique est telle que représentée dans la Figure 5 : des couches de convolution successives avec une taille de *kernel* de plus en plus petite afin d'extraire les informations globales de l'image dans un premier temps, puis de plus en plus précises dans un deuxième temps ; le processus est ensuite répété à l'inverse avec des couches de « déconvolution » (le processus inverse d'une convolution, c'est-à-dire qu'une image plus grande est construite à partir de l'entrée en déplaçant le *kernel*).

Cette approche possède un inconvénient : les premières couches de déconvolution se basent sur la (au mieux, les) dernière(s) couche(s) de convolution, c'est-à-dire que le résultat des premières couches du modèle (les informations plus globales de l'image) se perdent au fur-et-à-mesure.

Concernant la sortie du modèle, comme vu dans la partie introduction de ce document, chaque pixel se voit assigner une classe. Si l'image d'entrée est de taille $n \times n$ et il y a y classes différentes, alors la sortie du modèle sera un **tenseur** $n \times n \times y$; le vecteur correspondant est dit *one-hot encoded*.

2.1.1. Encodage one-hot

L'encodage one-hot, aussi appelé encodage 1 parmi n , consiste à encoder une variable à n états sur n bits dont un seul prend la valeur 1, le numéro du bit valant 1 étant le numéro de l'état pris par la variable⁶. Ainsi, en ayant les classes chien, chat et pastèque, soit $\begin{pmatrix} \text{chien} \\ \text{chat} \\ \text{pastèque} \end{pmatrix}$, la classe pastèque sera représentée par $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ en encodage one-hot. Cette méthode s'avère particulièrement utile dans le domaine du **deep learning** (voire du machine learning en général), car il est admis et facilement constatable que l'usage de valeurs sur les espaces $[0, 1]$ ou $[-1, 1]$ (flottants, $x \in \mathbb{D}$) est plus stable numériquement que des valeurs entières ($x \in \mathbb{Z}$) choisies arbitrairement de 0 à n .

2.1.2. Comment mesurer les performances ?

La précision, aussi appelée *accuracy* en anglais, représente les performances du modèle sur une tâche donnée, souvent exprimée en pourcentage. Une fois une classe assignée à chaque pixel de l'image d'entrée, donc à la sortie du modèle, la question se pose alors du protocole à suivre pour attribuer un score (une précision/*accuracy*) à la répartition des classes proposée. Une méthode simple serait de comparer la classe attribuée à chaque pixel puis de la comparer avec la valeur attendue, puis en retenant 1 pour une valeur correcte, et 0 sinon ; enfin, la moyenne de tous les scores des pixels serait faite, donnant le score global de la répartition proposée. Cette approche est toutefois inadaptée car elle ne tiendrait pas compte du déséquilibre entre les classes. Concrètement, dans un jeu de données, une ou plusieurs classes sont souvent sur-représentées ; par exemple, si la plupart des photos incluent du ciel visible, alors il est probable que le ciel soit surreprésenté, c'est-à-dire que la classe « ciel » correspondante sera beaucoup plus fréquente en nombre de pixels sur l'ensemble du jeu.

La métrique la plus utilisée afin d'exprimer la précision (*accuracy*) en segmentation sémantique est l'**intersection sur l'union moyenne** (mIoU), en anglais *mean Intersection over Union*. L'*Intersection over Union* (IoU), pour une classe donnée, correspond à l'espace où les pixels de la classe en question prédits chevauchent ceux attendus divisé par l'union des espaces prédits et attendus, comme illustré dans la Figure 6. En d'autres termes, cette métrique exprime la quantité de surface correctement prédite divisé par l'union de la surface correctement prédite, de la surface incorrectement prédite et de la surface attendue non prédite.

⁶Citation tirée de https://fr.wikipedia.org/wiki/Encodage_one-hot


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Figure 6: Formule représentant l'*Intersection over Union* ([source](#))

D'autres métriques existent également, comme le *dice coefficient* (*F1 score*), mais sont moins utilisées dans l'état de l'art du domaine de la segmentation sémantique. Sans s'attarder sur les détails, elle est corrélée au **mIoU** et diffère en ce qu'elle porte plutôt sur la performance moyenne tandis que le **mIoU** exprime plus la performance dans le pire cas.

Cette introduction permettant de présenter les outils standards du domaine, intéressons-nous maintenant aux papiers et modèles étudiés de le cadre de cette revue de l'état de l'art. En deep learning, l'état de l'art est extrêmement riche et il est impossible d'en faire le tour, c'est pourquoi je présente ici une sélection des papiers que j'ai trouvé les plus pertinents, qui sont à la fois en rapport direct avec le sujet ainsi que d'autres qui permettent de connecter d'autres domaines. Le domaine de la segmentation sémantique est plutôt réduit par rapport à certains autres comme la classification et il est donc important de rester ouvert aux avancées pouvant venir d'ailleurs.

2.2. Modèles basés sur des convolutions

2.2.1. U-Net

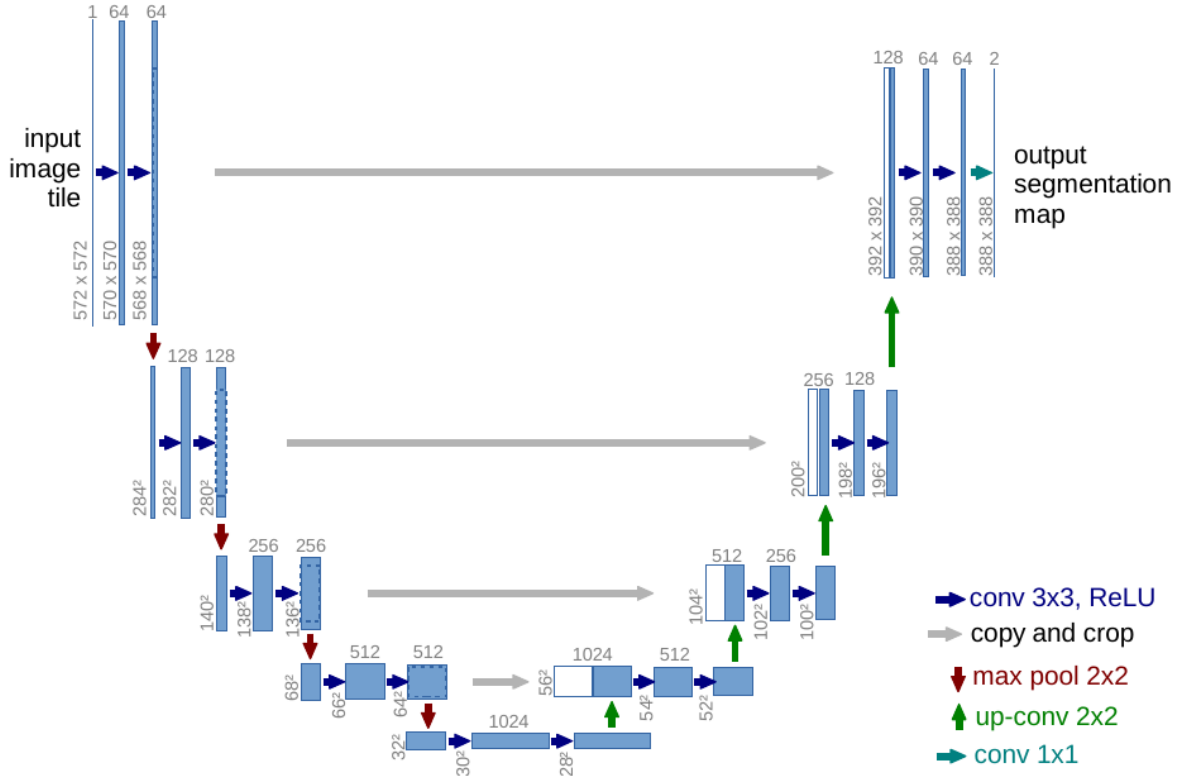


Figure 7: Architecture du **modèle** U-Net (source: [RFB15])

Le modèle U-Net [RFB15] est historiquement une référence car il a introduit les *skip-connections* (sauts de connexions), comme visible sur la Figure 7 : ce sont des connexions allant du décodeur vers le décodeur prévenant la perte d'information comme vu précédemment (Chapitre 2.1).

Dans ce modèle, l'encodeur est découpé en petits blocs, qui sont associés avec leurs équivalent côté décodeur ; l'idée est de combiner les caractéristiques plus haut-niveau (abstraites, globales à l'image) avec des plus basses (plus précises) sans perte d'information.

2.2.2. PSPNet

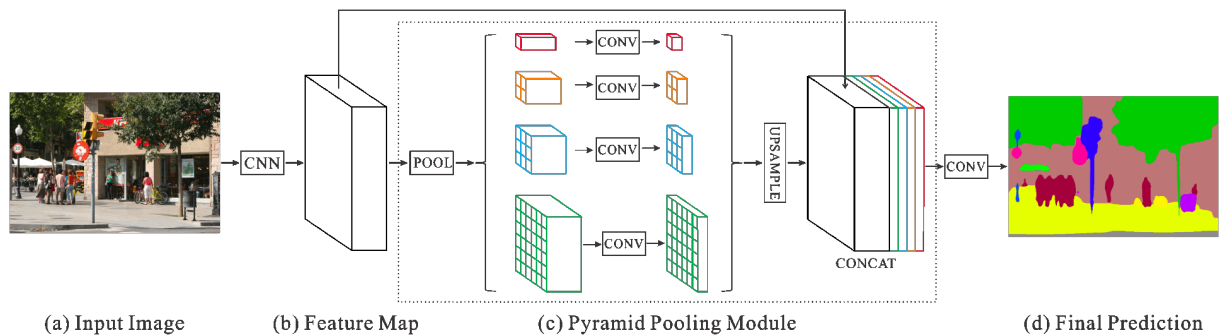


Figure 8: Vue globale de l'architecture de PSPNet et son *Pyramid Pooling Module* (source: [Zha+17])

PSPNet [Zha+17] introduit le *Pyramid Pooling Module* (module de *pooling* pyramidal) ; le terme *pooling* (en français : regroupement ou mise en commun) signifie une réduction

de la dimension spatiale des données. Comme visible dans la partie (c) de la Figure 8, ce module effectue 4 {convolutions, suréchantillonnages} de tailles de *kernel* différentes en parallèle, puis les concatène dans un CNN afin de produire les prédictions finales en (d).

Ce concept est très intéressant puisqu'il permet d'effectuer une extraction de caractéristiques en parallèle, précédemment effectué de manière séquentielle.

2.2.3. DeepLab

Le modèle DeepLab existe sous 4 versions, introduites par 4 papiers dont les auteurs principaux travaillent chez Google. Nous discutons ici quelques points intéressants apportés par les différentes versions.

DeepLab v1 (2014) [Che+16]. Cette version, à l'origine du modèle DeepLab, présente les *atrous (dilated) convolutions*, soit convolutions dilatées en français. Un *kernel* d'une convolution dilatée est similaire à celui d'une convolution régulière, sauf qu'un paramètre r est ajouté à sa diagonale, afin de venir « étirer » les pixels sur une plus grande zone, en les espaçant effectivement. Comme illustré sur la Figure 9, un *kernel* 3×3 dilaté avec $D = 2$ voit chaque pixel qu'il contient séparé de ses voisins de $D = 2$ pixels.

Cela implique qu'un *kernel* 3×3 verrait approximativement la même chose qu'un *kernel* 5×5 , mais avec seulement 9 paramètres au lieu de 25 en utilisant un taux de dilation de $D = 2$. Dans les faits, plus d'informations pourront être capturées avec (potentiellement) le même coût calculatoire. Il est alors possible de remplacer un ensemble convolution + **max-pooling** par une convolution dilatée !

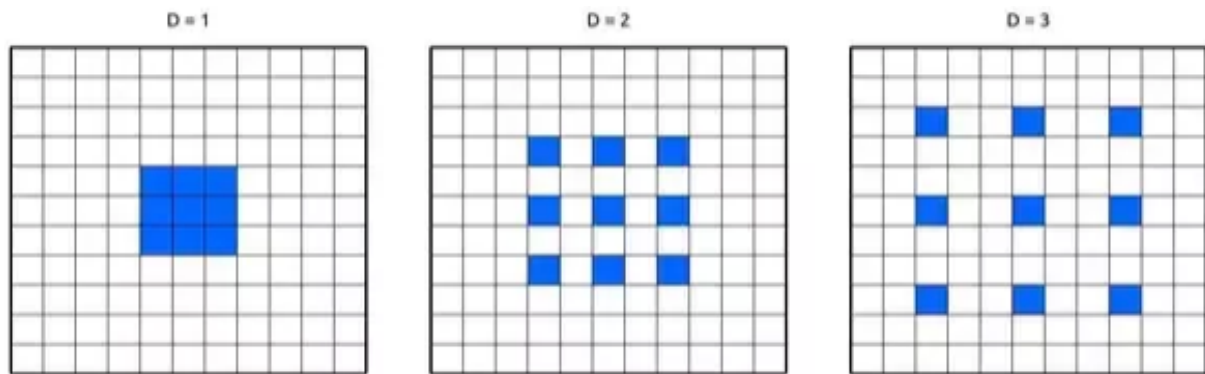


Figure 9: Un *kernel* 3×3 de :

(a) convolution régulière ou de dilatation $D = 1$

(b) dilatation $D = 2$ (c) dilatation $D = 3$

(source: [HV20])

La version 1 utilise également un CRF (*Conditional Random Field*) dans les couches denses finales du modèle, qui a pour effet de « lisser » les contours des objets en éliminant les artéfacts visuels. Pour cela, des connexions aléatoires sont effectuées en basant les décisions sur celles des voisins les plus proches.

DeepLab v2 (2016) [Che+17a]. La version 2 intègre notamment l'ASPP (*Atrous Spatial Pyramid Pooling*) présenté par PSPNet [Zha+17].

DeepLab v3 (2017) [Che+17b]. Cette version présente des convolutions dilatées mises en cascade et/ou en parallèle en faisant varier le taux de dilatation afin d'améliorer l'ASPP de la version 2.

DeepLab v3+ (2018) [Che+18]. L'ASPP des précédentes versions est combinée avec l'architecture encodeur-décodeur (par exemple de U-Net) ; le résultat est alors un encodeur-décodeur contenant un *pyramid pooling module* à la fin de l'encodeur.

2.2.4. UPerNet

UPerNet [Xia+18] est un article scientifique qui présente une méthodologie, soit un « framework » (et non un réseau/modèle) permettant d'achever plusieurs tâches de **computer vision**. C'est un connecteur générique de *backbones* (un *backbone* est un modèle de classification dont on se sert comme réseau pour extraire les composantes principales des images avant de les segmenter) pour du multi-tâches. Il ne résout pas un problème mais se branche sur plusieurs *backbones* : c'est la généralisation d'un décodeur en quelque sorte.

L'intuition proposée, pour introduire ce modèle, est que les convolutions ne permettent pas, de par leur nature et malgré leur biais cognitif (un pixel est toujours traité dans son contexte), d'accéder au contexte global de l'image.

Beaucoup d'équipes qui entraînent un nouveau modèle de classification donnent ensuite leur score sur la tâche de segmentation sémantique en utilisant UPerNet. Même si cette approche est intéressante, il en ressort que les modèles en question sont généralement gros (beaucoup de paramètres) et en ajoutant le coût de l'entraînement d'UPerNet, la tâche serait trop coûteuse.

Cependant, la question sous-jacente reste importante : nous étudions le coût de l'entraînement de modèles mais faut-il autoriser les *backbones* (parties de modèles) pré-entraînés par autrui pour notre entraînement ?

L'ensemble des modèles vus ci-dessus (U-Net, PSPNet, DeepLab et UPerNet) utilisent tous des *backbones* comme ResNet (un modèle de référence dans la classification d'images) : nous verrons les raisons et répondrons précisément à ce point dans la section méthodologie un peu plus loin.

2.3. Modèles basés sur des Transformers

2.3.1. ViT (Vision Transformers)

Le modèle Vision Transformers [Dos+21] doit être présenté ici car il est l'un des premiers à proposer une adaptation des **Transformers** pour la **computer vision**. C'est un modèle de classification d'images, et son fonctionnement repose sur le découpage de l'image en « patches » (parcelles) puis en y appliquant le mécanisme d'attention.

2.3.2. FocalNets

Les FocalNets (Focal Modulation Networks) [Yan+22] sont une série de modèles proposant une modification du mécanisme d'attention en effectuant une agrégation plus rapide dans la partie *query*, *key* et *value*. Ils utilisent UPerNet pour la segmentation sémantique car c'est un modèle de classification.

2.3.3. SegFormer

SegFormer [Xie+21] propose une approche utilisant quasiment exclusivement des Transformers et leur mécanisme d'attention (modifié). Dans l'encodeur (nommé MiT pour *Mix Transformer*), l'image d'entrée est découpée en patches de 4×4 , puis en reproduisant un mécanisme similaire à l'ASPP, c'est-à-dire en générant des représentations multi-niveaux à $\{\frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}\}$ de la résolution de l'image d'origine. Côté décodeur, une simple série de couches denses est utilisée car l'encodeur est considéré comme suffisamment « puissant » ; le coût calculatoire est alors très intéressant.

2.3.4. Segmenter

Contrairement à SegFormer, le modèle Segmenter [Str+21] repose sur un *backbone* comme ViT [Dos+21] et traite le problème de segmentation sémantique comme de la séquence vers séquence, similaire au **NLP**. Le but est de modéliser le problème de la computer vision par un problème rapportable à du NLP, là où SegFormer souhaite plus radicalement adapter l'architecture à la tâche de la vision. Il se décline en deux versions : une version où le décodeur est un MLP (c'est-à-dire un ensemble de couches denses, où tous les neurones sont reliés), et une version où un décodeur est basé sur les Transformers.

L'intuition proposée, pour introduire ce modèle, est que les convolutions ne permettent pas, de par leur nature et malgré leur biais cognitif (un pixel est toujours traité dans son contexte), d'accéder facilement au contexte global de l'image. Plusieurs procédés (empiriques) tentent d'y remédier mais complexifient le modèle inutilement. L'idée est d'utiliser les Transformers, connus dans le NLP comme étant capables d'exploiter des informations globales, malgré la complexité quadratique de leur module d'attention.

2.3.5. Segment Anything Model (SAM)

Cette publication scientifique [Kir+23], proposée par une équipe de Meta AI Research durant ce projet de fin d'études, a très vite été populaire au sein de la communauté scientifique.

Il remet en question le domaine même de la segmentation sémantique en introduisant une nouvelle tâche, appelée *promptable segmentation*, dont le concept est de produire un résultat similaire à de la segmentation panoptique selon plusieurs entrées possibles : des points sur l'image, un **prompt**, etc. afin de segmenter ce que l'utilisateur a demandé. En effet, celui-ci consiste en deux innovations :

1. Un jeu de données. Habituellement, les jeux de données en segmentation sémantique se font rares. En effet, l'un des plus connus est ADE20K qui contient plus de 20 000 images, mais cela est souvent considéré comme trop faible ; les modèles sont partiellement entraînés sur des jeux plus gros comme le populaire ImageNet (classification) avant d'être affinés sur ADE20K. Ce papier introduit SA-1B, un nouveau jeu de données conçu en collaboration entre des annotateurs humains et un modèle de segmentation, pour un total de plus d'1 milliard de masques et 11 millions d'images.
2. Un modèle, SAM (*Segment Anything Model*), qui possède une particularité intéressante : lors du chargement d'une image, la première partie de l'**inférence** est exécutée sur **GPU** (dans le cloud), permettant d'extraire les composants principales de l'image, puis la seconde partie est exécutée directement sur l'appareil de l'utilisateur, comme son navigateur ! Le tout permettant d'effectuer divers « requêtes » à l'image après son traitement initial.

Ce papier ne s'adresse donc pas à de la segmentation sémantique mais propose une approche très enrichissante car les applications qui peuvent en découler sont multiples. Par exemple, il est assez évident que Meta vise le domaine de la réalité virtuelle (VR) afin de tendre vers l'AR (*augmented reality*, réalité augmentée) à terme.

2.4. Modèles hybrides convolutions/Transformers

Je n'ai pas trouvé de modèle spécifique à de la segmentation sémantique étant un hybride convolutions + Transformers. Cependant, les nouveaux modèles de classification basés sur des Transformers proposent des résultats en segmentation sémantique en utilisant des frameworks comme UPerNet étant eux-mêmes constitués de couches de convolution, ce qui est donc une sorte d'hybride.

Toutefois, ces modèles sont souvent gros et ne mettent pas en avant une réelle « collaboration » entre **convolutions** et **Transformers**. Également, par choix, cette partie n'aura pas été traitée pour se concentrer sur le reste ; la classification est un domaine plus actif que la segmentation sémantique, c'est pourquoi les innovations de ce premier devraient graduellement atteindre les autres domaines.

2.5. Modèles choisis

Suite à cette étude de l'état de l'art, j'ai choisi 4 modèles afin de le représenter au mieux dans le cadre de ce projet.

U-Net [RFB15]. Modèle déjà disponible au sein de l'équipe, une référence du domaine avec une architecture encodeur-décodeur utilisant des **convolutions**.

DeepLab v3+ [Che+18]. Également un des piliers du domaine, un incontournable très compétitif dans l'état de l'art, intégrant plusieurs techniques modernes, comme les convolutions dilatées.

SegFormer [Xie+21] et **Segmenter** [Str+21]. Ces deux modèles sont parmi les premiers à adapter les **Transformers** spécifiquement pour le domaine de la **segmentation sémantique**. Il ont été publiés à peu d'intervalle et, curieusement, ne

se citent pas l'un et l'autre. Ils apportent chacun des concepts intéressants, inspirés de ViT [\[Dos+21\]](#).

3. Méthodologie

Dans cette partie, nous présenterons la méthodologie qui a été utilisée pour conduire les expériences durant ce projet de fin d'études. Cette méthodologie a pour objectif d'apporter des éléments de réponse aux questions suivantes : vaut-il mieux utiliser des convolutions ou des Transformers pour la tâche de segmentation sémantique ? Des modèles basés sur ces technologies se mettent-ils bien à l'échelle sur un supercalculateur ? Cette méthodologie reste cependant idéaliste sur certains points (par exemple PyTorch, qui peut contenir des différences de performances en son sein), dont nous reparlerons plus tard.

3.1. Choix préalables

3.1.1. Jeu de données

Pour cette étude nous choisissons le jeu de données ADE20K [Zho+18], introduit en 2018. Il figure systématiquement parmi les plus utilisées dans le domaine de la **segmentation sémantique**, et pour cause : il possède 22 210 images et 150 classes⁷, une diversité des plus intéressantes dans le domaine.

En comparaison, le jeu PASCAL Context (2010) [Mot+14], qui étend le PASCAL VOC 2010 pour le rendre compatible avec la segmentation sémantique, contient 59 classes⁷ (ensemble habituellement utilisé) pour 10 103 images.

3.1.2. Modèles avec *backbones*

Certains modèles (DeepLab, Segmenter) utilisent des *backbones*, c'est-à-dire que leur architecture est « branchée » à un modèle de classification placé en amont, un *feature extractor* (extracteur de caractéristiques). Généralement, des modèles de type ResNet sont utilisés car ceux-ci sont très connus et bien optimisés de par leur grande maturité. Ils doivent leur nom au fait qu'ils sont pré-entraînés sur un ou plusieurs jeu(x) de données de classification au préalable ; et l'on utilise leur couches [de convolution] comme des outils permettant d'extraire des caractéristiques à différentes échelles suite à leur « expérience » sur des jeux de classification, afin de la transposer à d'autres tâches.

Il s'avèrent d'ailleurs particulièrement utiles sur des petits jeux de données comme ADE20K (comparativement à d'autres, comme ImageNet [Den+09] (classification) qui comporte ~1.2 million d'images), en partant du principe que beaucoup de notions définissant un objet d'un point de vue généraliste peuvent se retrouver dans d'autres sources de données.

⁷Les deux jeux de données contiennent en réalité bien plus de classes, mais nous nous intéressons ici au nombre utilisable en pratique ; les nombres cités sont donc les ensembles habituellement utilisés, ce qui signifie que les classes comportent chacune suffisamment d'exemples pour être correctement représentées.

Ces *backbones* posent donc un problème de biais de par leur natures variées et leur pré-entraînements, c’est pourquoi je choisis de les autoriser mais sans pré-entraînements, pour que tous les modèles partent avec les mêmes chances : nous souhaitons comparer les modèles dans leur capacité à restituer des connaissances.

Il est également envisageable d’effectuer une comparaison avec pré-entraînements, mais cela ne rentre malheureusement pas dans le cadre de ce projet de fin d’études.

3.2. Facteurs de comparaison

Dans cette section, nous souhaitons aborder les configurations communes entre les modèles ainsi que ceux qui permettront de les différencier dans le cadre de notre étude.

Configurations communes	Critères de comparaison
Précision	Énergie (Wh)
Utilisation des GPUs	Scalabilité
Framework (PyTorch)	Nombre de paramètres
Environnement (Champollion)	
Nombre de paramètres	

Tableau 1: Liste des configurations communes et de comparaison pour l’entraînement des différents modèles

3.2.1. Configurations communes

Précision. Le but est d’entraîner tous les modèles afin d’atteindre une *accuracy* (précision) commune. Cela permettra d’évaluer leurs temps de convergence. Cependant, un modèle peut théoriquement converger plus lentement en utilisant moins d’énergie⁸.

Nous souhaitons pour cela maximiser l’utilisation des **GPUs** afin de garantir (autant que possible) un code optimisé, laissant le cœur du modèle et ses capacités s’exprimer pleinement, en déplaçant le **bottleneck** au niveau des GPUs.

Bottlenecks. Lors de l’entraînement d’un modèle sur un supercalculateur, plusieurs facteurs, dont certains sont illustrés sur la Figure 10, peuvent être limitants. Le GPU est généralement le composant le plus cher d’un système, donc le but est d’optimiser les calculs pour qu’il soit le point bloquant (*bottleneck*) lors de l’entraînement. Pour cela, le reste de l’infrastructure doit être dimensionné de manière cohérente, par exemple en terme de communication si le multi-nœud est important pour la tâche choisie.

Ces *bottlenecks* peuvent être analysés par des outils de monitoring par exemple : il s’agit de solutions permettant de surveiller l’activité sur une ou plusieurs machines. J’ai eu l’occasion de développer une solution de bout-en-bout venant se connecter directement

⁸Cela ne peut pas signifier non plus qu’il mettra beaucoup plus longtemps à converger car un serveur consomme de l’énergie en étant *idle* (inactif), soit dans la durée ; converger lentement ne peut donc pas représenter un avantage, l’idée est ici seulement d’être souple dans notre approche.

à notre *job scheduler* (ordonnanceur de tâches) dans le cluster, en affichant une interface web liée à la tâche actuelle afin de lire l'activité des **GPUs**, des **CPUs**, du stockage, des communications réseau et même de la consommation électrique du/des serveur(s) !

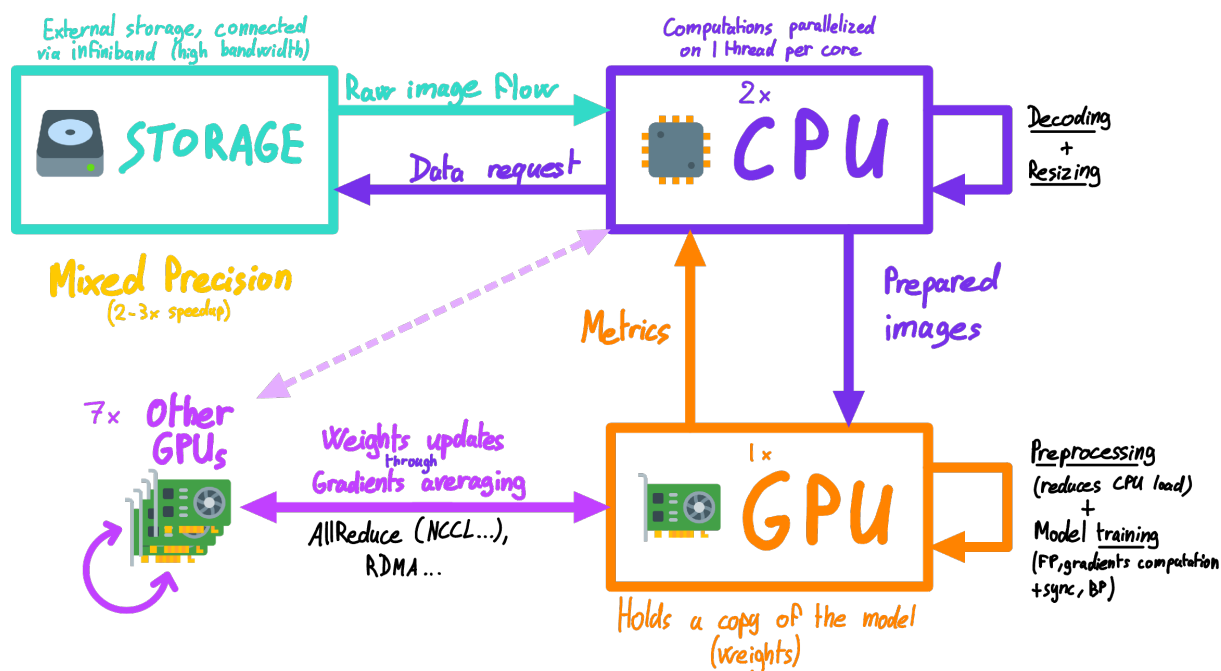


Figure 10: Interactions entre les différents composants d'un système lors d'un entraînement de modèle de deep learning sur un seul nœud d'un cluster de calcul

- **Le stockage** : coût fixe pour chaque fichier donc les gros fichiers sont à privilégier
- **La mémoire** : si la RAM est trop petite, le serveur peut se mettre à *swap* (*offloading* (déchargement) sur le disque - très pénalisant)
- **Les CPUs** : si trop d'opérations se font sur les processeurs, ceux-ci peuvent devenir limitants. Dans les grandes lignes, durant un entraînement de deep learning, les CPUs doivent s'occuper des opérations de chargement, décodage et augmentation des données, en les *streamant* (communiquant en flux asynchrone) au **GPUs**
- **Les communications** : la Figure 10 représente les communications entre les GPUs et les CPUs ainsi qu'entre les GPUs eux-mêmes, mais ne montre pas les communications inter-serveurs dans le cas d'un entraînement distribué multi-nœuds. En effet, tous ces points peuvent poser problème, notamment dans le cadre d'entraînement de modèles de **NLP**, où les communications sont souvent très impactantes (les couches étant réparties sur plusieurs GPUs, en particulier).

Utilisation des GPUs. Afin de déplacer le bottleneck vers les GPUs, nous devons vérifier deux critères qui permettent de s'en assurer de manière plutôt fiable :

1. L'utilisation est à 100% de manière quasi-constante, ce qui montre qu'ils ne sont pas inactifs
2. L'utilisation d'énergie est proche du seuil maximal annoncé par le constructeur : dans le cas des GPUs NVIDIA, dépasser la consommation maximale annoncée est un bon signe qu'un code est optimisé

Le critère d'utilisation des GPUs peut même être étendu : les modèles vont tous devoir être optimisés à la main pour respecter les critères ci-dessus.

Framework. Le framework populaire **PyTorch** a été choisi pour l'implémentation des 4 modèles choisis, ce qui permet d'avoir un critère supplémentaire en commun pour cette comparaison.

Environnement de calcul. Le choix de l'environnement matériel sera discuté un peu plus loin.

Nombre de paramètres. Le nombre de paramètres des modèles est à la fois un critère commun car un nombre de paramètres égal peut vouloir dire capacité de mémoire identique, mais dans les faits les types numériques des paramètres peuvent être différents et leur impacts individuels varier. Essayer d'avoir un nombre de paramètres similaire permet toutefois d'approximer grossièrement le coût calculatoire de l'entraînement.

3.2.2. Critères de comparaison

2 facteurs nous permettrons de différencier les modèles dans leurs performances :

Énergie. Pour des configurations communes données, l'énergie consommée pour converger, mesurée en Wh (Watt · heures), diffèrera et sera notre critère d'évaluation. L'énergie consommée prend en compte la somme des puissances perçues par les blocs d'alimentation des serveurs durant tout l'entraînement, ce qui signifie que des facteurs externes comme les *switchs*, les climatisations et le réseau ne sont pas pris en compte. En effet, leur impact énergétique reste minime une fois réparti sur tout le datacenter.

Scalabilité. Un modèle peut se comporter différemment lorsqu'il est entraîné sur un ou plusieurs GPUs, ainsi que sur un ou plusieurs serveurs : ces différences de comportement peuvent se manifester sous la forme d'une convergence qui diffère lorsque l'entraînement est mis à l'échelle, ou des moins bonnes performances de calcul. Nous inspecterons alors les différents comportements afin de constater des potentiels problèmes.

3.3. Choix de l'environnement

Dans cette section, nous discutons les choix d'environnement. Ces choix peuvent varier en fonction des conditions de (re)production de ces travaux, mais leur explicitation permet de comprendre les conditions dans lesquelles nous avons obtenu nos résultats et de les extrapoler à d'autres environnements si nécessaire.

3.3.1. Champollion

Dans le lab (datacenter expérimental) de Grenoble, nous possédons un supercalculateur construit en partenariat avec nos partenaires, mis à disposition en interne mais également pour des travaux de recherche via l'institution [MIAI](#).

Ce supercalculateur est composé de 20 serveurs HPE Apollo 6500 Gen10+. Chaque serveur est équipé de 2 processeurs AMD 7763, 8 GPUs NVIDIA A100 SXM4 (l'architecture SXM permet d'avoir, en plus du bus PCIe gen4 (32 GB/sec unidirectionnel), un bus NVLink permettant des taux de transfert de 300 GB/sec inter-GPUs), 1 TB de mémoire RAM, 4 cartes InfiniBand HDR offrant un taux de transfert inter-serveurs total de 800 Gb/sec (soit 100 GB/sec) ainsi qu'une connexion Ethernet 100 Gb.

InfiniBand. L'InfiniBand est un standard de communication réseau utilisé dans le calcul hautes performances qui se caractérise par un débit très élevé et un temps de latence très faible. Il est utilisé pour l'interconnexion de données entre serveurs, mais aussi avec des systèmes de stockage.

Le but est de reproduire une portion d'un supercalculateur livré chez un client afin de pouvoir exécuter des démonstrations, des **benchmarks** ou des expériences en interne.

Le lab de Grenoble est géré par les différentes personnes qui composent l'équipe, et tout le monde peut donc participer, ce qui permet d'avoir une vue concrète et un grand contrôle du matériel sur lequel nos codes tournent, contrairement à une solution (non-HPC) cloud (Azure, AWS...).

3.3.2. HPE MLDE

Comme présenté précédemment **MLDE** permet de paralléliser des entraînements de deep learning via une solution conteneurisée. Une fois une solution entraînable avec la plateforme, il est facile de la partager à d'autres personnes dans le but de la reproduire, étant donné sa nature conteneurisée. Suite à son rachat par HPE, c'est une solution que nous souhaitons pousser chez nos clients sur leurs machines, c'est pourquoi il nous semble évident de devoir proposer, au possible, une implémentation compatible **MLDE** dans le cadre de ce projet.

3.4. Challenges

3.4.1. Compatibilité avec MLDE

Parmi les modèles choisis, U-Net est déjà compatible avec MLDE, tandis que SegFormer, Segmenter et DeepLab ne le sont pas.

SegFormer et Segmenter ont publiés des implémentations basées sur MMCV, qui est une surcouche de PyTorch facilitant la création de modèles. Le problème de ce framework est qu'il cache notamment le processus d'entraînement, bloquant un grand nombre d'optimisations. Toutefois, MLDE propose un module MMCV qui pourrait peut-être être réemployé.

Concernant DeepLab, une implémentation officielle existe également mais est très personnalisée (la base de code est complexe) et difficile à porter, c'est pourquoi le modèle devra peut-être être réécrit.

3.4.2. Des environnements différents

Les entraînements s'effectuant sur une base de code Python, le réel challenge se situe certainement au niveau des environnements (logiciels). En effet, Python ne dispose pas de standard robuste dans le domaine du deep learning, et encore moins en ce qui concerne le **calcul haute performance** (HPC). Installer un environnement directement (nativement) sur une machine est compliqué à cause de la manipulation de variables d'environnement, mais obtenir un conteneur avec un environnement aligné avec l'état de l'art est une tâche ardue.

3.5. Pour aller plus loin

Idéalement, une recherche d'hyperparamètres pourrait être faite : elle consiste à chercher les jeux d'hyperparamètres régissant les caractéristiques principales des modèles jusqu'à en trouver la meilleure configuration selon un critère donné. Ici, le critère serait la quantité d'énergie utilisée pour l'entraînement, afin d'atteindre la précision cible en utilisant le minimum d'énergie pour chaque configuration (par exemple en diminuant le nombre de paramètres).

4. Résultats

Dans cette partie, nous développerons les résultats obtenus lors de nos expérimentations suivant la méthodologie précédemment décrite, en la confrontant aux problématiques rencontrées ne permettant pas d’avoir autant de résultats qu’espéré.

4.1. Contraintes technologiques

J’ai rencontré de nombreuses limitations technologiques durant la réalisation de ce projet de fin d’études, et souhaite partager dans ce rapport les étapes importantes et bloquantes de ce processus d’optimisation, ne permettant pas d’aboutir à tous les résultats au terme de ce projet; en effet, un échec peut également être considéré comme un résultat.

4.1.1. Technologies initiales

Nous verrons ici quel était l’état des projets correspondants aux modèles choisis, tels que j’ai pu les récupérer.

U-Net. U-Net, le modèle de référence sur lequel nous souhaitons nous baser, utilise une base de code que nous devons modifier afin de rendre l’entraînement compatible avec notre jeu de données : ADE20K. Les technologies alors utilisées ne sont pas dépréciées mais méritaient une mise à jour : **CUDA** est en version 11.3 alors que la 12.1 est désormais disponible, **PyTorch** utilise la version 1.10 alors que la 2.01 est disponible, et ceci sans parler de leurs propres dépendences.

Mettre à jour les piles logicielles est, pour moi, une étape primordiale car en calcul hautes performances, l’amélioration continue est recherchée. **CUDA**, dont l’objectif principal est d’accélérer des calculs sur **GPU**, en est un exemple. Dans un domaine évoluant aussi vite que le **deep learning**, un retard de mises à jour de plus d’un an comme ici représente un écart potentiel considérable.

SegFormer et Segmenter. Ces deux modèles, tous deux écrits par des doctorants, ont été programmés en utilisant la librairie MMSegmentation, elle-même basée sur MMCV. Ces librairies sont des abstractions basées sur **PyTorch** et ne permettent pas, par exemple, de personnaliser la boucle d’entraînement, réduisant alors le contrôle que nous avons sur les algorithmes, et rendant ainsi difficile l’optimisation des différentes opérations exécutées pendant l’entraînement des modèles. Par ailleurs, je constate que les entraînements respectifs sont très peu optimisés en utilisant des outils de monitoring (système de mesure).

En effet, comme visible sur la Figure 11, nous pouvons constater que les **GPUs** sont sous-utilisés : comme mentionné précédemment dans la méthodologie, le taux moyen d’utilisation est de 80%, et nous remarquons que des périodes où un seul GPU est utilisé de manière prolongé sont présentes (phase de validation). D’autre part, nos soupçons sont confirmés en regardant la puissance moyenne consommée, ici d’environ 150W en

moyenne par GPU (plus de 400W peuvent être attendus sur un GPU A100 SXM4 comme dans ce cas).



Figure 11: Premier aperçu de l'entraînement via un système de monitoring

DeepLab v3+. Une implémentation officielle est disponible, utilisant le framework **TensorFlow**. La base de code est très vaste et complexe à prendre en main, ce qui rend compliqué son optimisation pour fonctionner sur notre supercalculateur avec nos technologies.

4.1.2. Technologies choisies

Suite aux points de départ desquels nous sommes partis, les choix principaux suivants ont été réalisés.

U-Net. Mise à jour de la pile logicielle et prise en charge du nouveau jeu de données.

SegFormer et Segmenter. Mise à jour vers les nouvelles versions d'MMCV et MMSegmentation, puis tentative d'optimisation en l'état afin d'éviter de devoir réimplémenter les modèles.

DeepLab v3+. Une tentative de réimplémentation a été réalisée avec **TensorFlow** mais a échoué du fait de la mauvaise prise en charge de ce framework par **MLDE**. Il a alors été jugé plus rentable d'effectuer une nouvelle réimplémentation avec **PyTorch**.

4.1.3. MLDE

Réaliser ce projet de fin d'études en utilisant **MLDE** aura eu au final un impact très négatif sur l'avancement du projet. Cette plateforme présentant de très gros avantages comme le suivi des expériences, de leurs résultats ou encore la parallélisation « auto-

matique », je suis aujourd’hui en mesure d’affirmer qu’elle n’est pas adaptée⁹ pour des travaux nécessitant des ressources au niveau de l’état de l’art comme dans cette étude.

En effet, MLDE est basée sur un système de conteneurisation (donc utilisant des **conteneurs**) pré-conçus, c’est-à-dire qu’en personnaliser un pour une tâche donnée est un long processus car celui-ci n’a pas été pensé pour. Après de nombreux échanges avec les équipes de **Determined.AI**, il a été déterminé qu’il ne serait pas possible de mettre à jour les piles logicielles comme je le voulais, même si cela ne m’a pas empêché de continuer à essayer de contourner ces limitations par la suite.

4.2. Entraînements

Dans cette partie, nous exposons les résultats obtenus lors des entraînements réussis de U-Net et de SegFormer.

Model	Param #	Training time	Energy used
U-Net	6.7M	9 min	302 Wh
SegFormer	3.7M	38 min	1700 Wh

Tableau 2: Résultats de l’entraînement des modèles U-Net et SegFormer sur un nœud (8 GPUs) du supercalculateur Champollion, pour une précision cible de 16 mIoU.

Comme vu dans la méthodologie (Chapitre 3), ces modèles ont été entraînés sans pré-entraînement sur d’autres jeux de données au préalable, mais sont partis de zéro. Pour ces raisons, nous ne pouvons pas les comparer à l’existant, les papiers dont ils sont issus utilisant le pré-entraînement car celui-ci est quasiment obligatoire si l’on souhaite obtenir des résultats compétitifs avec le reste de l’état de l’art.

Nous avons essayé de choisir des modèles dont les nombres de paramètres sont les plus proches entre eux afin de rendre la comparaison plus fiable. Ici le modèle U-Net dispose de 6.7 millions de paramètres, contre 3.7 millions pour SegFormer.

La précision (*accuracy*) cible est fixée à 16 **mIoU**, suite à des limitations rencontrées pendant l’entraînement, sans quoi nous aurions pu monter jusqu’à environ 32 mIoU, point qu’il est difficile de dépasser sans pré-entraînement d’après mes tests. Lorsque le jeu de données n’est pas assez grand, comme ici si l’on poursuit l’entraînement, un phénomène nommé *overfitting* se produit alors : le modèle commence à mémoriser le jeu de données d’entraînement, et on voit alors apparaître une divergence entre les fonction de coût (*loss*) d’entraînement et de validation, comme visible dans la Figure 12. En effet, à-partir d’environ 1000 batch (paquets de données) entraînés, la fonction de coût de validation (en orange) s’éloigne de la bleue. Sur ce schéma, les *loss* finales correspondent respectivement à 49 mIoU (jeu d’entraînement) et 32 mIoU (jeu de validation).

⁹À moins d’utiliser la [Core API](#), pour les connaisseurs, qui était ici un trop gros investissement étant donné nos points de dépôts

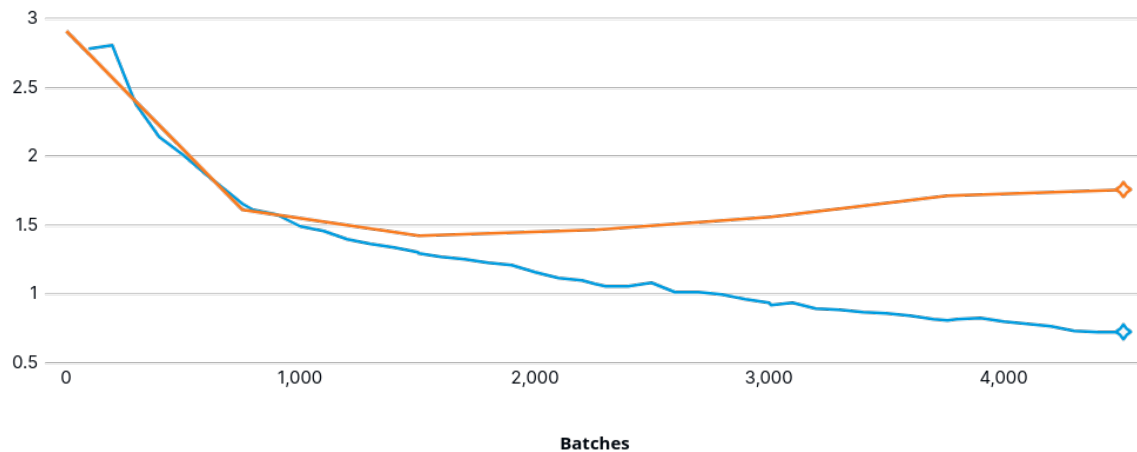


Figure 12: Illustration du phénomène d’overfitting lorsque l’entraînement de U-Net sans pré-entraînement sur ADE20K est poursuivi (*loss* d’entraînement en bleu, *loss* de validation en orange)

Comme visible dans le Tableau 2, U-Net a demandé moins d’énergie ($\sim 6 \times$ moins) que pour l’entraînement que SegFormer, même si ce dernier possédait moins de paramètres (3.7 millions contre 6.7 millions pour U-Net) ! La différence est également visible sur les temps d’entraînement, où U-Net a nécessité 9 minutes contre 38 pour SegFormer.

Les détails de consommation d’énergie sont disponibles dans les Figure 13 et Figure 14 : en moyenne 2.25 kW pour U-Net contre 2.75 kW pour SegFormer.

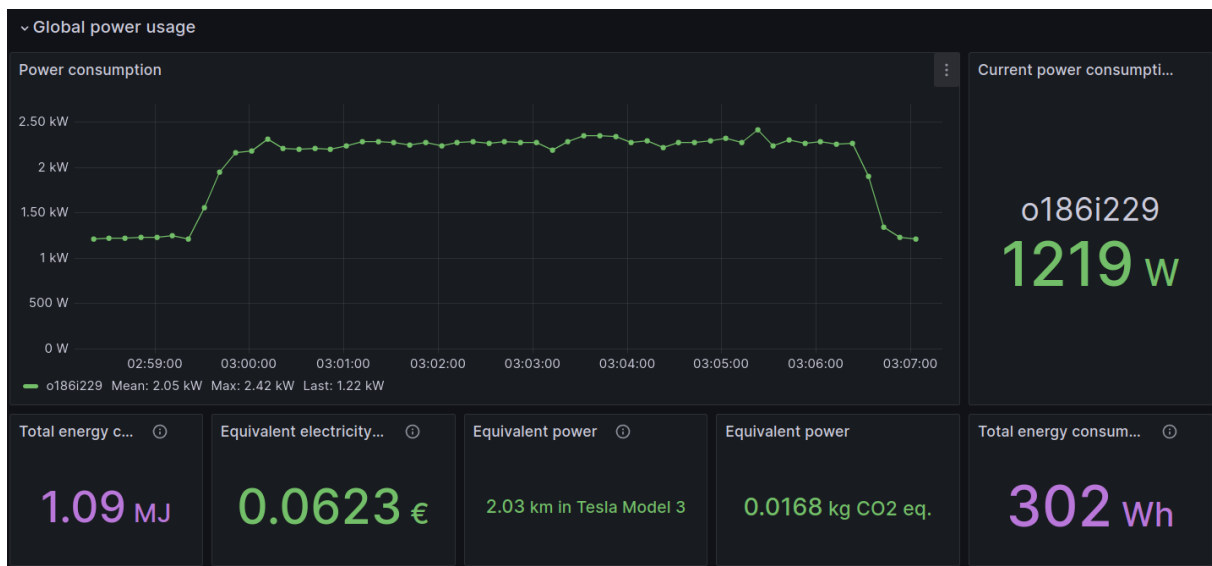


Figure 13: Consommation d’énergie durant l’entraînement de U-Net

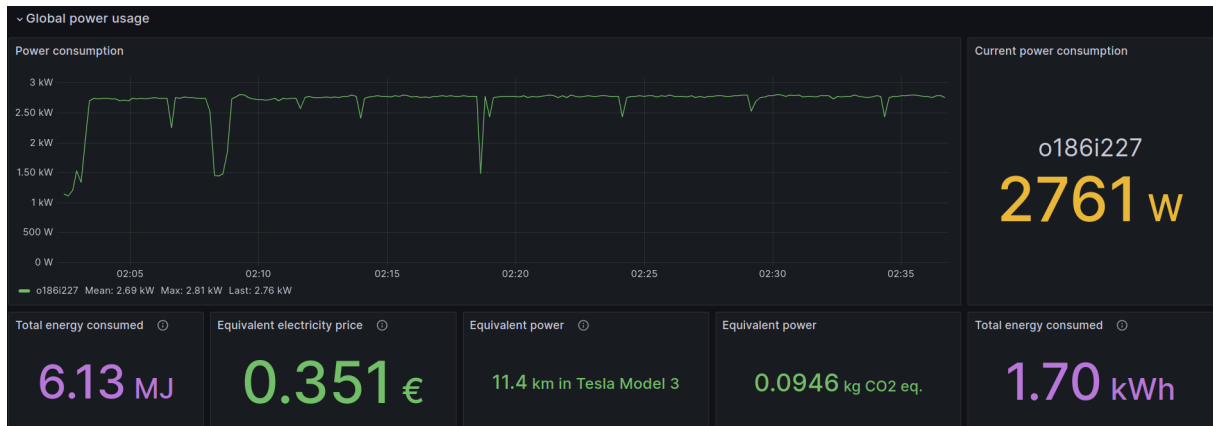


Figure 14: Consommation d'énergie durant l'entraînement de SegFormer

Les différences dans les résultats pourraient s'expliquer de plusieurs manières, mais en voici plusieurs plausibles :

Kernels CUDA. Un *kernel* **CUDA** (un noyau de calcul **CUDA**) est une routine compilée pour des accélérateurs (par exemple des **GPUs**) distincte du programme principal mais utilisée par celui-ci. Il consiste en une implémentation optimisée d'une ou plusieurs opérations. Idéalement, un calcul sur **GPU** est constitué d'une suite d'exécution de *kernels*.

Le **Transformer** étant une technologie récente, peu de *kernel* **CUDA** existent pour définir de telles opérations. De plus, des papiers comme SegFormer viennent redéfinir l'essence même de certains composants du Transformer pour la vision, c'est pourquoi leur implémentation ne peut pas être aussi efficace qu'un composant largement utilisé comme une couche de convolution qui, elle, dispose de multiples kernels depuis des années déjà.

Mémorisation de l'information et généralisation. Certaines équipes [Dai+21]¹⁰ affirment que les **Transformers** possèdent une meilleure capacité à mémoriser mais qu'ils font preuve de moins bonnes capacités de généralisation que les **couches de convolution**, dû à un biais induit par leur nature (traitement de pixels dans leur contexte/entourage).

Les résultats que nous exposons dans cette partie montrent donc qu'un modèle basé sur des **couches de convolution** semble être moins coûteux à entraîner qu'un modèle basé sur des **Transformers**, que ce soit en termes de temps, d'énergie (électricité) ou d'argent (prix de l'électricité + temps d'utilisation des machines).

¹⁰Voir l'abstract du papier cité. Cet article scientifique est d'ailleurs un exemple d'architecture hybride convolutions/self-attention en classification

5. Discussions

Dans cette partie, nous analyserons de manière critique l’approche développée dans les précédentes parties.

5.1. Forces de l’approche

Le point de vue **HPC** d’une solution de **deep learning** n’est que rarement abordé dans l’état de l’art, les résultats montrant un meilleur score sur un jeu de données attisant généralement plus la curiosité des lecteurs. Dans un contexte où l’on cherche délibérément le « toujours plus », il nous semble bon de se poser la question « comment consommer moins ? ».

Paradoxalement, répondre à cette question passe par de la consommation de notre côté, pour pouvoir fournir ces résultats. Cependant, la rentabilité de ce coût sera déterminée par la réutilisation de ce travail par la suite, via le présent document par exemple ou par la méthodologie que nous présentons.

Enfin, une des forces de notre approche est la généricité de la méthodologie, qui peut s’appliquer à tout type d’accélérateur et de système afin d’effectuer une comparaison du même type.

5.2. Faiblesses de l’approche

Dans notre démarche, il est possible d’observer que de nombreux éléments sont « uniques ». Par exemple l’environnement d’exécution (un seul type de serveur donc un seul type de GPU, etc.), le framework (PyTorch seulement), nos résultats (un seul modèle de chaque type) ou encore MLDE (un seul outil de parallélisation).

MLDE. D’une manière générale, des technologies comme **MLDE** rajoutent des couches d’abstraction au-dessus d’une couche déjà existante (dans ce cas, PyTorch). Avec le recul, dans un contexte de recherche très près de l’état de l’art, ce détail est très important car il limite immédiatement notre capacité à innover en dépendant d’un facteur supplémentaire, c’est-à-dire d’avoir moins de contrôle sur notre solution ; ce qui vient en contradiction avec le domaine du **HPC** où il s’agit souvent, justement, de contrôle sur les opérations.

De la redondance dans les outils est donc nécessaire afin de stabiliser les résultats ; cela vient compléter les critères communs lors d’une comparaison, afin que les réelles différences puissent ressortir sans être masquées par des facteurs externes.

Le « problème » que nous essayons de résoudre peut-être représenté comme un espace infini-dimensionnel dans lequel nous devons chercher notre réponse, donc nous ne pourrions jamais avoir un environnement parfait : la difficulté est de déterminer quels facteurs sont les plus importants et peuvent avoir le plus d’influence sur les résultats.

Une question intéressante que nous pouvons nous poser est si, au final, les différences entre les couches de convolution et les Transformers n’étaient pas liées à l’implémenta-

tion ? Celle-ci varie en fonction des frameworks voire même en fonction du matériel utilisé, donc il se pourrait qu'elle soit en fait prédominante dans tous les tests effectués si nous généralisons à d'autres frameworks.

5.3. Notre approche répond-elle à la problématique ?

Je pense que cette approche répond à la problématique « *pourquoi devrions-nous utiliser des mécanismes basés sur l'attention en vision par ordinateur ?* » en investiguant les différences entre les couches de convolution et les Transformers dans le contexte d'entraînement de modèles de deep learning appliqué à la tâche de segmentation sémantique.

5.4. Pistes de poursuites

Cette étude se déroulant sur 6 mois, le sujet pourrait très largement être approfondi, sur de nombreux axes ! En voici quelques exemples.

Dans un premier temps, finaliser les entraînements des deux autres modèles (DeepLab v3+ et Segmenter) serait un vrai plus, et adapter tous les modèles pour les rendre compatible avec **MLDE** (tâche très compliquée sans réaliser de compromis).

Deuxièmement, effectuer les tests à nouveau mais avec les phases de pre-training serait très intéressant car les capacités mémorielles des **Transformers** seraient plus sollicitées.

Ensuite, une comparaison bas niveau entre les convolutions et les Transformers serait une bonne chose. Un exemple d'une telle comparaison est effectué dans [Li+21], mais je pense qu'il serait mieux de l'axer sur une comparaison au niveau des **FLOPs** et de l'énergie liée.

Enfin, le deep learning et l'intelligence artificielle en général sont des domaines où de nombreux articles scientifiques paraissent tous les jours, et il est donc probable que les meilleures méthodes et choix émergent simplement empiriquement dans le futur !

5.5. Impact environnemental et sociétal

Dans cette section, nous verrons les aspects liés à l'impact environnemental et sociétal de ce PFE.

Estimation de la consommation énergétique au travail. Pendant ce PFE de 6 mois, j'estime la consommation des équipements suivants, utilisés au travail :

- Ordinateur portable + 2 écrans: 200 kWh
- Supercalculateur Champollion, en état inactif (hors charge), hors infrastructure externe (clims, *switchs*...), à partager entre les utilisateurs : 89 000 kWh
- Éclairage : 240 kWh

Déplacements. Mes déplacements ont été réalisés en utilisant les transports en communs ou en trottinette électrique, soit environ 1.68kg CO₂eq.

Politique de la structure d'accueil. D'après les informations que j'ai pu trouver, HPE est engagé dans la réduction de l'impact environnemental de ses clients mais ne donne pas d'exemples précis concernant sa *supply chain*.

L'entreprise a toutefois annoncé qu'elle souhaitait atteindre le « Net 0 » émissions dans sa *value chain* d'ici 2040¹¹.

Impact sociétal du produit fini. Le produit fini, dans notre cas cette étude à-travers ce rapport et les codes associés, a le potentiel d'aider à réduire la consommation d'énergie (d'électricité) sur les entraînements de deep learning en mettant en lumière les différences entre les technologies étudiées, par exemple sur des entraînements à large échelle.

¹¹<https://www.hpe.com/us/en/living-progress.html>

6. Conclusions

Dans ce projet de fin d'études, nous avons cherché à montrer en quoi il vaut mieux utiliser des **couches de convolution** ou des **Transformers** afin de répondre à la problématique « *pourquoi devrions-nous utiliser des mécanismes basés sur l'attention en vision par ordinateur ?* », dans un contexte de modèles d'IA générative devenant toujours plus coûteux à entraîner ainsi que de l'effervescence du **NLP**.

Nous avons pu voir que les approches existantes étaient limitantes même si techniquement intéressantes car elles n'étaient pas étudiées pour des applications de **HPC**, et ne proposaient pas de comparaison plus fondamentales entre les convolutions et les Transformers.

Notre proposition a été d'ajouter un œil « performance » en optimisant des bases de codes afin de pouvoir les comparer, en appliquant notre problématique au domaine de la segmentation sémantique.

Nous avons pu constater que les modèles basés sur des **couches de convolution** ont l'avantage d'être plus facilement optimisés à-travers les différentes bibliothèques de code existantes et seront donc certainement préférées sur des jeux de données plus petits, alors que la capacité potentielle de mémorisation des Transformers sera plutôt mise à profit à grande échelle, sur de très larges jeux de données.

Toutefois, notre étude possède des limites : il faudrait multiplier les observations avec divers paramètres, comme par exemple en finalisant les entraînements des modèles restants pour venir ajuster nos propos.

Dans le futur, cette étude pourrait être appliquée à d'autres sous-domaines de la **computer vision**, en testant également d'autres systèmes de parallélisation, comme une pile logicielle basée sur **Alpa + Jax + Ray** qui ne rentrerait finalement pas dans le cadre de ce PFE !

Enfin, ce PFE a été bien plus que ce rapport : durant mon alternance, j'ai pu développer des solutions qui doivent toujours être maintenues, nous avons accueilli une doctorante dans le cadre du programme MIAI sur Champollion afin d'aider la recherche et j'ai également apporté de l'aide à d'autres collègues qu'ils m'ont bien souvent rendu en échange !

Je pense que ce projet de fin d'études aura été une très riche expérience.

7. Bibliographie

Vous pouvez retrouver dans cette section l'ensemble des papiers cités dans ce document, par ordre alphabétique.

- [BCB16] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. 2016. *Neural Machine Translation by Jointly Learning to Align and Translate*. arXiv, <https://doi.org/10.48550/arXiv.1409.0473>.
- [Che+17a] Chen, Liang-Chieh et al. 2017a. *Deeplab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected Crfs*. arXiv, <https://doi.org/10.48550/arXiv.1606.00915>.
- [Che+16] Chen, Liang-Chieh et al. 2016. *Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected Crfs*. arXiv, <https://doi.org/10.48550/arXiv.1412.7062>.
- [Che+17b] Chen, Liang-Chieh et al. 2017b. *Rethinking Atrous Convolution for Semantic Image Segmentation*. arXiv, <https://doi.org/10.48550/arXiv.1706.05587>.
- [Che+18] Chen, Liang-Chieh et al. 2018. *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*. arXiv, <https://doi.org/10.48550/arXiv.1802.02611>.
- [Dai+21] Dai, Zihang et al. 2021. *Coatnet: Marrying Convolution and Attention for All Data Sizes*. arXiv, <https://doi.org/10.48550/arXiv.2106.04803>.
- [Den+09] Deng, Jia et al. 2009. “Imagenet: A Large-Scale Hierarchical Image Database.” In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. Conference Presentation at 2009 IEEE Conference on Computer Vision and Pattern Recognition, 248–255, <https://doi.org/10.1109/CVPR.2009.5206848>.
- [Dos+21] Dosovitskiy, Alexey et al. 2021. *An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv, <https://doi.org/10.48550/arXiv.2010.11929>.
- [HV20] Heffels, Michael R., and Joaquin Vanschoren. 2020. *Aerial Imagery Pixel-Level Segmentation*. arXiv, <https://doi.org/10.48550/arXiv.2012.02024>.
- [Kim+17] Kim, Yoon et al. 2017. *Structured Attention Networks*. arXiv, <https://doi.org/10.48550/arXiv.1702.00887>.
- [Kir+23] Kirillov, Alexander et al. 2023. *Segment Anything*. arXiv, <https://doi.org/10.48550/arXiv.2304.02643>.
- [Li+20] Li, Liam et al. 2020. *A System for Massively Parallel Hyperparameter Tuning*. arXiv, <https://doi.org/10.48550/arXiv.1810.05934>.

- [Li+21] Li, Shanda et al. 2021. *Can Vision Transformers Perform Convolution?*. arXiv, <https://doi.org/10.48550/arXiv.2111.01353>.
- [LVL22] Luccioni, Alexandra Sasha, Sylvain Viguier, and Anne-Laure Ligozat. 2022. *Estimating the Carbon Footprint of BLOOM, a 176b Parameter Language Model*. arXiv, <https://doi.org/10.48550/arXiv.2211.02001>.
- [Mot+14] Mottaghi, Roozbeh et al. 2014. “The Role of Context for Object Detection and Semantic Segmentation in the Wild.” In . Conference Presentation at Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 891–898, https://openaccess.thecvf.com/content_cvpr_2014/html/Mottaghi_The_Role_of_2014_CVPR_paper.html.
- [Mäd22] Mädje, Laurenz. 2022. *A Programmable Markup Language for Typesetting*. Thesis: tu-berlin, <https://www.user.tu-berlin.de/laurmaedje/programmable-markup-language-for-typesetting.pdf>.
- [Par+16] Parikh, Ankur P. et al. 2016. *A Decomposable Attention Model for Natural Language Inference*. arXiv, <https://doi.org/10.48550/arXiv.1606.01933>.
- [RFB15] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. 2015. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv, <https://doi.org/10.48550/arXiv.1505.04597>.
- [Str+21] Strudel, Robin et al. 2021. *Segmenter: Transformer for Semantic Segmentation*. arXiv, <https://doi.org/10.48550/arXiv.2105.05633>.
- [Vas+23] Vaswani, Ashish et al. 2023. *Attention Is All You Need*. arXiv, <https://doi.org/10.48550/arXiv.1706.03762>.
- [Xia+18] Xiao, Tete et al. 2018. *Unified Perceptual Parsing for Scene Understanding*. arXiv, <https://doi.org/10.48550/arXiv.1807.10221>.
- [Xie+21] Xie, Enze et al. 2021. *Segformer: Simple and Efficient Design for Semantic Segmentation with Transformers*. arXiv, <https://doi.org/10.48550/arXiv.2105.15203>.
- [Yan+22] Yang, Jianwei et al. 2022. *Focal Modulation Networks*. arXiv, <https://doi.org/10.48550/arXiv.2203.11926>.
- [Zha+17] Zhao, Hengshuang et al. 2017. *Pyramid Scene Parsing Network*. arXiv, <https://doi.org/10.48550/arXiv.1612.01105>.
- [Zho+18] Zhou, Bolei et al. 2018. *Semantic Understanding of Scenes Through the Ade20k Dataset*. arXiv, <https://doi.org/10.48550/arXiv.1608.05442>.