



## Projet de Recherche (PRe)

Spécialité : STIC

Année scolaire : 2021/2022

---

**Adaptation de domaine non supervisée pour la segmentation  
sémantique**

---

Auteur : Rémi GASTAUD      Promotion : 2023

**Mention de confidentialité**  
**Rapport non confidentiel**

**Stage effectué du 12 mai au 5 août 2022**

**Maître de stage :** Gianni Franchi, Enseignant-Chercheur  
UMA ENSTA Paris, 828 boulevard de Maréchaux, 91120 Palaiseau, France

**Enseignant référent :** Goran Frehse, Enseignant Chercheur  
UMA ENSTA Paris, 828 boulevard de Maréchaux, 91120 Palaiseau, France

---

## Remerciements

Je tiens d'abord à remercier mon maître de stage M. Gianni Franchi, qui m'a proposé ce stage et qui m'a accompagné tout le long, malgré un nombre important de stagiaires à l'U2IS. J'ai appris beaucoup grâce à lui, ses directives et ses conseils.

Je tiens également à remercier M. Goran FREHSE, qui a accepté d'être mon enseignant référent pour ce stage, et donc accepté de lire et d'évaluer ce rapport.

Merci aussi à tout le laboratoire de l'U2IS pour son accueil chaleureux.

---

## Résumé

Les réseaux de neurones sont sensibles aux changements de domaines, c'est-à-dire que s'ils sont appliqués sur des données trop éloignées de celles sur lesquelles ils se sont entraînés, leur résultats deviennent vite mauvais. Cela est problématique, puisque l'on ne dispose pas toujours des données sur lesquelles on veut appliquer un réseau. C'est là qu'intervient l'adaptation de domaine. Celle-ci vise à utiliser des techniques permettant de palier à ce problème. L'une de ces méthodes est l'image-to-image translation, c'est-à-dire la transformation d'image provenant d'un certaine distribution en image provenant d'une autre distribution. Nous allons voir comment est-ce que l'on peut appliquer tout cela à des tâches de segmentation sémantique.

## Abstract

Neural networks are sensitive to domain changes, i.e. if they are applied to data that is too far away from the data on which they were trained, their results quickly become poor. This is problematic, since one does not always have the data on which one wants to apply a network. This is where domain adaptation comes in. Domain adaptation aims to use techniques to overcome this problem. One of these methods is image-to-image translation, i.e. the transformation of an image from a certain distribution into an image from another distribution. We will see how this can be applied to semantic segmentation tasks.

## Mots-Clés

Segmentation sémantique, Deep Learning, Adaptation de Domaine, Prédiction de profondeur, Image-to-image translation

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>6</b>
<b>2</b>	<b>Déroulement du stage</b>	<b>7</b>
<b>3</b>	<b>Présentation du problème</b>	<b>8</b>
3.1	Les réseaux de neurone et le deep learning . . . . .	8
3.2	Le traitement d'image et les réseaux de convolution . . . . .	9
3.3	La segmentation sémantique . . . . .	11
3.4	La prédiction de profondeur . . . . .	12
3.5	L'adaptation de domaine non supervisée . . . . .	14
<b>4</b>	<b>Image-to-Image translation</b>	<b>16</b>
4.1	Présentation . . . . .	16
4.2	GAN . . . . .	16
4.3	CycleGAN . . . . .	16
4.4	Cycada . . . . .	18
4.5	TSIT . . . . .	19
<b>5</b>	<b>Mes contributions</b>	<b>22</b>
5.1	Le choix du dataset . . . . .	22
5.2	Premiers essais avec DeepLabV3Plus . . . . .	24
5.3	Essais avec CycleGAN . . . . .	25
5.4	Essais avec Cycada . . . . .	28
5.5	Essais avec TSIT . . . . .	28
5.6	Modifications . . . . .	30
5.6.1	Consistance sémantique . . . . .	30
5.6.2	Histogram matching . . . . .	31
5.6.3	SPADE . . . . .	32
5.7	Récapitulatif des modifications et de leur résultats . . . . .	33
<b>6</b>	<b>Conclusion</b>	<b>35</b>
6.1	Mon expérience personnelle . . . . .	35
6.2	Travail mené et à venir . . . . .	35

## Table des figures

1	Schéma d'un neurone formel . . . . .	8
2	Schéma d'un réseau de neurones . . . . .	8
3	Exemple de convolutions . . . . .	9
4	Max Pooling . . . . .	9
5	Un réseau de convolution entier . . . . .	10
6	Exemple de segmentation sémantique . . . . .	11
7	Formule de l'IoU . . . . .	11
8	Exemple de prédiction de profondeur . . . . .	13
9	Type d'intervalles . . . . .	14
10	Architecture d'AdaBins . . . . .	14
11	Exemples d'image-to-image translation avec CycleGAN . . . . .	15
12	Fonctionnement d'un GAN . . . . .	16
13	Fonctionnement du CycleGAN . . . . .	17
14	Exemples d'image-to-image translation avec CycleGAN . . . . .	17
15	Schéma du fonctionnement de CyCADA . . . . .	18
16	Fonction objectif de CyCADA . . . . .	19
17	Architecture de TSIT . . . . .	20
18	Schéma de FADE . . . . .	21
19	Exemple d'image de Cityscapes . . . . .	22
20	Exemple d'image de GTA . . . . .	23
21	Exemple d'image de Synthia . . . . .	23
22	Exemple d'image de kitti (à gauche) et virtual kitti (à droite) . . . . .	24
23	Exemple de transformation (Réel - CycleGAN - TSIT) . . . . .	30
24	Histogram Matching . . . . .	31
25	Exemple de transformation avec consistance sémantique et Histogram Matching (réel- TSIT( $\alpha = 1$ )-Histogram Matching . . . . .	32
26	Schéma de SPADE . . . . .	32

## Liste des tableaux

1	Les deux modes d'entraînement dans notre cas particulier . . . . .	15
2	Les deux modes d'entraînement dans notre cas particulier . . . . .	24
3	Résultats de l'adaptation de domaine avec CycleGAN, LR de base . . . . .	26
4	Résultats de l'adaptation de domaine avec CycleGAN, LR de base divisé par 10 . . . . .	27
5	Résultats de l'adaptation de domaine avec CycleGAN, LR de base multiplié par 10 . . . . .	28
6	Résultats de l'adaptation de domaine avec TSIT . . . . .	29
7	Résultats de l'adaptation de domaine avec TSIT, LR de base divisé par 10	29
8	Résultats de l'adaptation de domaine avec TSIT, LR de base multiplié par 10 . . . . .	30
9	Résultats de l'adaptation de domaine avec TSIT et consistance sémantique	31
10	Résultats avec $\alpha = 0$ et SPADE combiné à FadaIN . . . . .	33
11	Résultats avec $\alpha = 0$ et SPADE combiné à FADE . . . . .	33
12	Récapitulatif des résultats avec les différentes modifications . . . . .	34

## 1 Introduction

Le sujet initial de ce projet était "Adaptation de domaine non supervisée pour la prédiction de profondeur à une caméra", mais au fur et à mesure de l'avancée du projet, la trajectoire des recherches s'est orientée vers de la segmentation sémantique. Le but est donc d'appliquer l'adaptation de domaine à la segmentation sémantique. Cette dernière désigne les techniques utilisées pour prédire les caractéristiques sémantiques d'une image (piéton, voiture, panneau ...). L'adaptation de domaine désigne l'ensemble des techniques utilisées pour adapter sur un certain ensemble un prédicteur entraîné sur un autre ensemble, suivant une autre distribution. Le terme "non supervisée" signifie que l'on ne dispose pas de données sémantiques sur l'ensemble "cible". L'enjeu de ce projet réside surtout dans l'adaptation de domaine non supervisée, et nous allons explorer différentes techniques, en s'inspirant de travaux déjà effectués, et en apportant de nouvelles idées.

## **2 Déroulement du stage**

Le stage a débuté le 16 Mai 2022, et s'est achevé le 5 Aout 2022, pour une durée totale de 12 semaines. L'objectif de ce stage est de nous donner une première expérience dans la recherche, en mettant à contribution ce que l'on a appris lors de nos études.

Le stage s'est déroulé à l'ENSTA, Ecole Nationale Supérieure des Techniques avancées, école d'ingénieur généraliste disposant de différents laboratoires : l'Unité de Chimie et Procédé (UCP), l'unité d'Informatique et d'Ingénierie des Systèmes (U2IS), l'Unité de Mathématiques Appliquées (UMA), l'Unité de Mécanique (UME), l'Unité d'Optique Appliquée (LOA) et l'Unité d'Économie Appliquée (UEA). Mon stage a pris place à l'U2IS, dirigée par M. Goran Frehse.

L'U2IS effectue des recherches en cybersécurité, en informatique, en robotique, et plus précisément en computer vision, où les réseaux de neurones, et en particulier les réseaux profonds sont largement utilisés.

### 3 Présentation du problème

#### 3.1 Les réseaux de neurone et le deep learning

Les réseaux de neurones existent depuis quelques dizaines d'années maintenant, mais c'est dans les années 2010 qu'ils se sont démocratisés, avec le développement du deep learning et l'augmentation des capacités de calcul. Le principe de base d'un réseau de neurones s'appuie sur son composant le plus simple : le neurone artificiel.

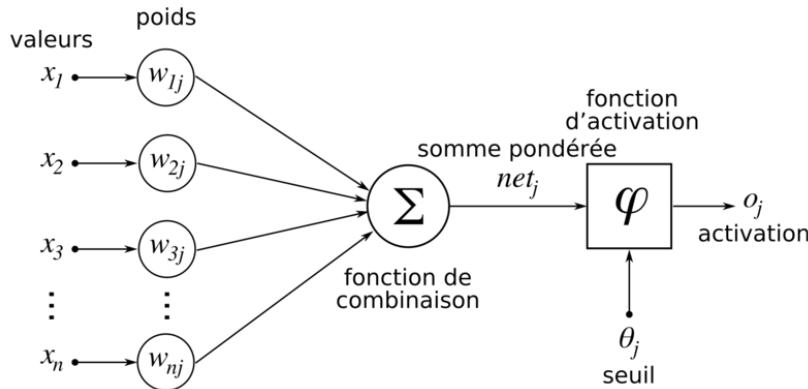


FIGURE 1 – Schéma d'un neurone formel

L'entrée est un vecteur  $x$ , et les paramètres du neurone sont les poids  $w$ . Le tout est combiné via une fonction de combinaison, linéaire, puis le résultat passe par une fonction d'activation non linéaire. La fonction d'activation la plus courante est nommée ReLU [1], pour Rectified Linear Unit, qui correspond à la formule suivante :

$$f(x) = \max(0, x)$$

L'étape d'après est de mettre ces neurones à la suite, et de les "empiler" sous forme de couches.

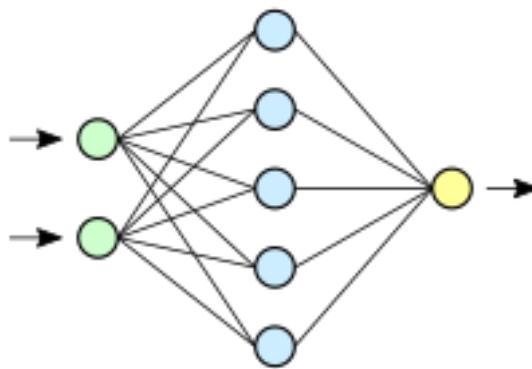


FIGURE 2 – Schéma d'un réseau de neurones

Un tel réseau est capable "d'apprendre", c'est-à-dire d'adapter ses poids pour obtenir les valeurs souhaitées en sortie. On lui donne une entrée, on observe la sortie, on la compare à une "étiquette", ou "label", qui correspond à la sortie attendue. On peut

ensuite ajuster les poids en fonction de cela à l'aide de divers algorithmes, le plus populaire étant la descente de gradient stochastique, que nous ne détaillerons pas dans ce rapport.

Le deep learning désigne l'ensemble des techniques utilisant un grand nombre de couches, le réseau devient donc "profond". C'est très utilisé dans le domaine de le traitement de l'image, ayant en grande partie remplacé toutes les techniques dites classiques, n'utilisant pas de réseau de neurone.

### 3.2 Le traitement d'image et les réseaux de convolution

Concernant le traitement d'image, la technique la plus largement utilisé est le réseau de convolution [2]. Celle-ci consiste à partitionner une image en tuiles de taille définie (3 pixels par 3 pixels par exemple), et à effectuer une opération sur chacune de ces tuiles. Cela donne en sortie un tenseur sur lequel on peut répéter d'autres opérations de convolutions.

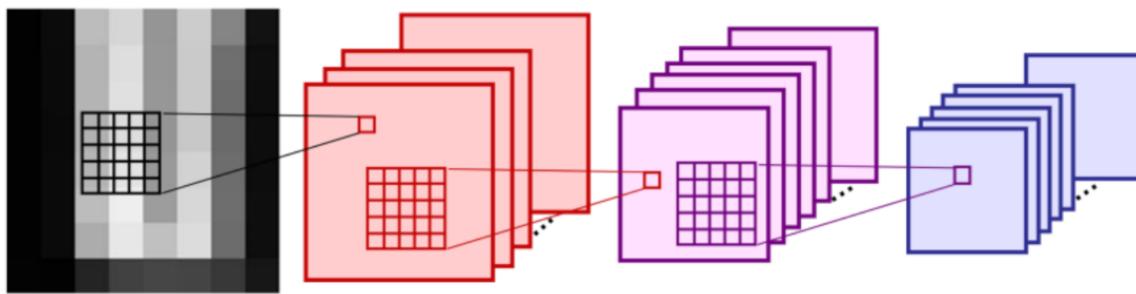


FIGURE 3 – Exemple de convolutions

Un réseau complet de convolution ne comporte en général pas uniquement des couches de convolution. Il peut être composé de couches dites de "pooling" [3], c'est-à-dire de couches qui sous-échantillonne l'image suivant une certaine règle. Le max pooling par exemple, qui prend le maximum sur chaque tuile de l'image (ou du tenseur).

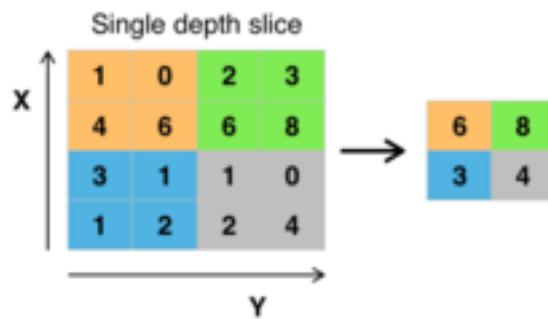


FIGURE 4 – Max Pooling

En sortie de réseau se trouve toujours une couche "entièrement connectée" (FC ou Fully Connected Layer) qui correspond à un réseau classique où tous les neurones sont connectés entre eux, et qui doit fournir la sortie du réseau total.

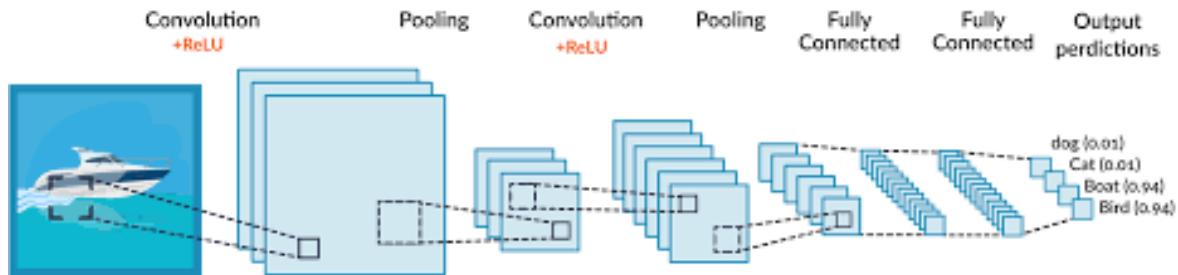


FIGURE 5 – Un réseau de convolution entier

### 3.3 La segmentation sémantique

La segmentation sémantique consiste à indiquer différents éléments qui constituent une image (voiture, trottoir, panneaux ...), le contexte le plus classique étant dans le cas d'image de paysages urbains. En pratique, cela demande de prédire, pour chaque pixel composant l'image, la classe d'objet à laquelle ce pixel appartient. Le nombre de classe est déterminé à l'avance, et la prédiction se compose, pour chaque pixel, d'un vecteur indiquant les probabilités d'appartenance à chaque classe.

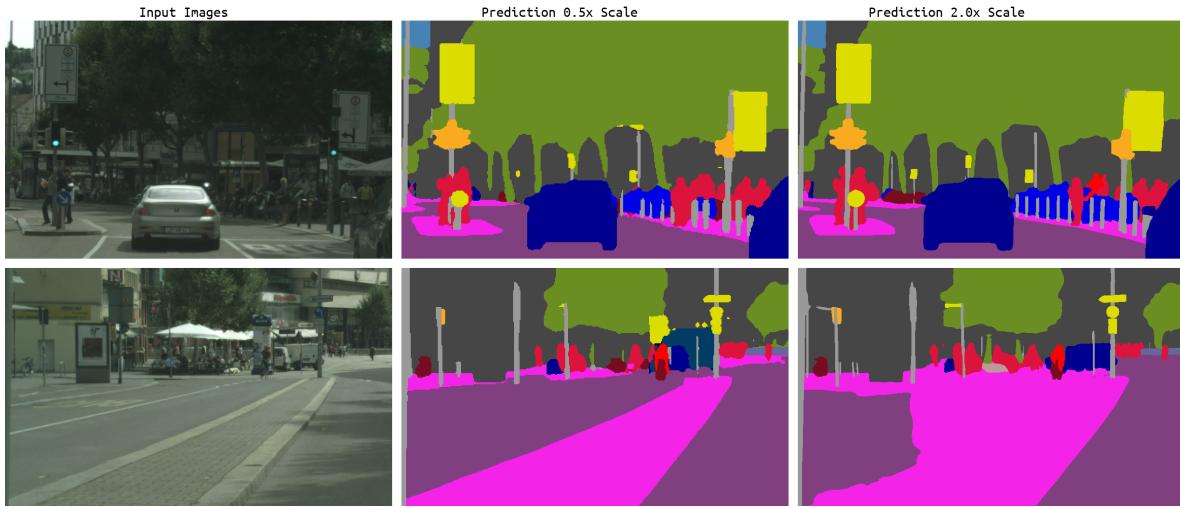


FIGURE 6 – Exemple de segmentation sémantique

En pratique, beaucoup de réseaux de DeepLearning ont été mis au point, celui que nous utiliserons dans ce projet se nomme DeepLabV3Plus[4].

Pour mesurer les performances d'un réseau de segmentation sémantique on utilise plusieurs métriques, les plus importantes étant le “mIoU” et la “pixel accuracy”.

Le mIoU est l'acronyme de mean Intersection over Union. Cela correspond à la moyenne des IoU, qui calculent pour chaque classe le nombre de pixels corrects divisé par le nombre de pixels appartenant à cette classe dans la réalité terrain, mais également dans la prédiction du réseau. Formellement, cela correspond à cette formule

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

FIGURE 7 – Formule de l'IoU

On effectue ensuite la moyenne de l'IoU de chaque classe pour obtenir le mIoU

La pixel accuracy est plus intuitive, puisqu'elle calcule simplement la part de pixel où la prédiction est juste.

$$PA = \frac{\sum_{j=1}^k n_{jj}}{\sum_{j=1}^k t_j}$$

Où  $n_{jj}$  correspond aux pixels bien prédis pour la classe  $j$ , et  $t_j$  aux pixels étiquetés de la classe  $j$ .

La métrique que nous utiliserons le plus est le mIoU, comme il est l'usage dans la littérature. Les autres pourront servir à obtenir d'autres informations.

L'intérêt du mIoU est qu'il prend en compte de la même manière chaque classe, et essayer de l'augmenter permet de ne pas oublier les classes peu représentées. Typiquement dans les paysages urbains, on observe beaucoup de routes et de trottoirs, et peu de panneaux (en quantité de pixels). Si l'on se contente de la précision pixel par pixel, on pourra obtenir de bons résultats, alors même que les panneaux sont très mal prédis. A l'inverse, le mIoU baissera drastiquement si une classe, même sous-représentée, est très mal prédictée.

### 3.4 La prédiction de profondeur

Une des applications du deep learning peut être la prédiction de carte de profondeur d'une image. Cela peut se faire avec des paires d'image, à la manière des yeux humains. En effet le cerveau se sert de cette dualité pour donner l'impression de profondeur. Toutefois, même avec un unique oeil le cerveau est capable de percevoir la profondeur. C'est à partir de ce constat que l'on peut espérer prédire la carte de profondeur d'une unique image. C'était le premier sujet de ce projet de recherche, prédire une carte de profondeur à l'aide d'une seule image, mais pour des raisons pratiques le sujet a dévié. Toutefois des recherches ont quand même été faites sur le sujet. Le principe est simple : on doit assigner à chaque pixel une valeur de profondeur, comprise dans un ensemble défini au préalable.

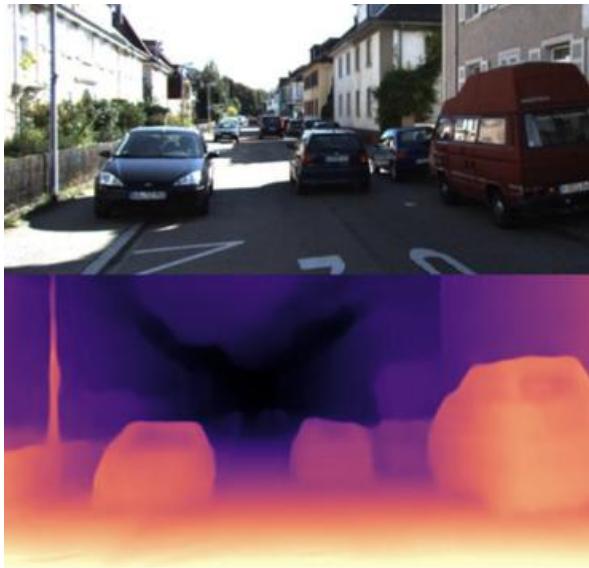


FIGURE 8 – Exemple de prédiction de profondeur

Il existe beaucoup de réseaux pour la prédiction à une image, mais celui utilisé durant les recherches est AdaBins[5], qui a de bons résultats pour la prédiction de profondeur à une seule image.

Pour la prédiction de profondeur, il existe plusieurs techniques, l'une d'elles est d'utiliser des "bins" (bacs en français), qui sont des intervalles de valeurs. Au lieu de prédire directement la valeur, on prédit un intervalle de valeur dans lequel va se trouver le pixel. Le problème se rapproche ainsi d'un problème de segmentation. Pour ensuite choisir la valeur prédite, on peut prendre le milieu de l'intervalle prédit avec la plus grande probabilité, ou bien faire la somme pondérée des milieux de ces intervalles par les probabilités. C'est cette deuxième technique qui sera utilisée.

$$d = \sum_i p_i m_i$$

où  $d$  est la profondeur prédite,  $m_i$  est le milieu de l'intervalle  $i$ , et  $p_i$  la probabilité prédite pour l'intervalle  $i$  par le réseau.

Reste à savoir comment choisir ces intervalles. Encore une fois, plusieurs méthodes existent. On peut choisir des intervalles uniformes, logarithmiques, ou bien utiliser un réseau pour apprendre ces intervalles. Dans ce cas, il faut choisir entre apprendre un intervalle commun à toutes les images d'un ensemble, ou apprendre à donner un intervalle pour chaque entrée. C'est cette dernière solution qui est utilisée dans AdaBins.

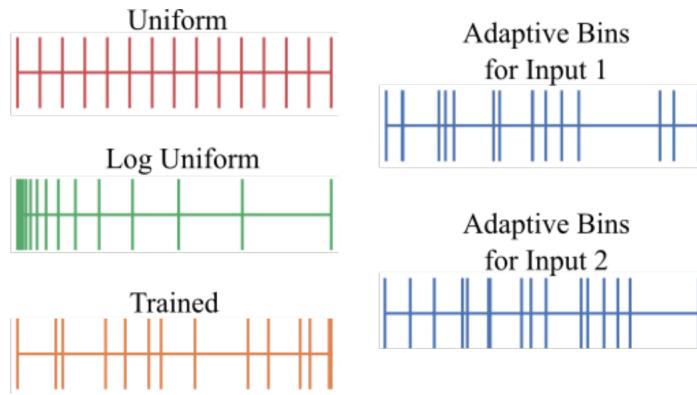


FIGURE 9 – Type d'intervalles

Nous n'allons pas entrer dans les détails de l'architecture, mais on peut voir quelles en sont les parties importantes

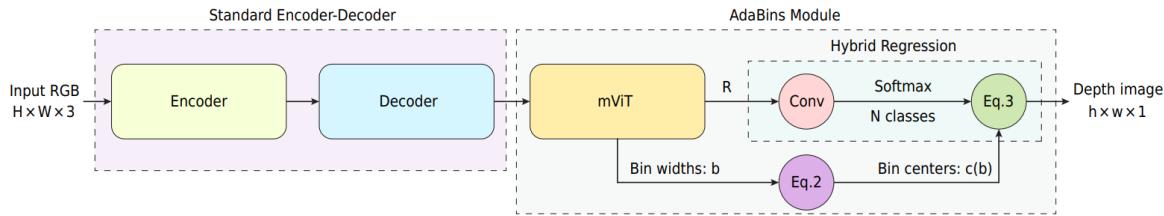


FIGURE 10 – Architecture d'AdaBins

La première section est un Encodage-Décodage, qui permet de travailler par la suite dans l'espace des "features", ou caractéristiques. Cette pratique est courante, elle consiste à utiliser des réseaux de convolutions pour obtenir un tenseur de caractéristiques contenant des informations plus générales sur les rapports entre les différentes parties de l'image passée en entrée. Vient ensuite le module "AdaBins", qui calcule la largeur des "bins" pour un total de  $N$  bins, leur centre, et prédit pour chaque pixel le vecteur de probabilité (taille  $N$ ) d'appartenance à ces bins.

### 3.5 L'adaptation de domaine non supervisée

L'une des principales contraintes lorsque l'on utilise des réseaux de neurones pour de l'apprentissage automatique est de se procurer des données d'apprentissage. Dans le cadre de l'apprentissage dit "supervisé", l'ensemble d'apprentissage est constitué de paires images-label, où le label est la donnée à prévoir (dans le cadre de la segmentation sémantique par exemple, le label est l'image où chaque pixel est coloré de la couleur de sa classe, un piéton d'une telle couleur, une voiture d'une autre). Le principal problème de cette technique, c'est qu'elle nécessite la création de ces labels. Pour la segmentation sémantique, c'est d'autant plus fastidieux que chaque pixel doit être annoté. Cette opération est très coûteuse en temps, c'est pourquoi la recherche s'est orientée vers des méthodes qui permettent d'outrepasser cette contrainte. Il existe diverses techniques, comme l'apprentissage semi-supervisé, où seule une partie des images servant

à l'apprentissage sont labelisées. Dans notre cas, nous allons utiliser l'adaptation de domaine. Celle-ci consiste à s'entraîner sur un certain ensemble dit “source”, pour faire des prédictions sur un autre ensemble dit “cible”. Dans notre cas, l'ensemble “source” est constitué d'images de synthèses représentant des paysages urbains, et l'ensemble “cible” de photographies de paysages urbain. Notre but va donc être de s'entraîner avec l'ensemble source, disposant de labels, et d'obtenir des résultats sur l'ensemble “cible”, pour lequel on ne dispose pas de label (en pratique on les a, mais on ne les utilise pas pour les entraînements, seulement pour les évaluations). Une technique d'adaptation de domaine consiste à utiliser la translation d'image d'un domaine vers un autre. L'image-to-image translation désigne l'ensemble des techniques permettant de transformer une image provenant d'une certaine distribution pour la faire ressembler à une image d'une autre distribution.

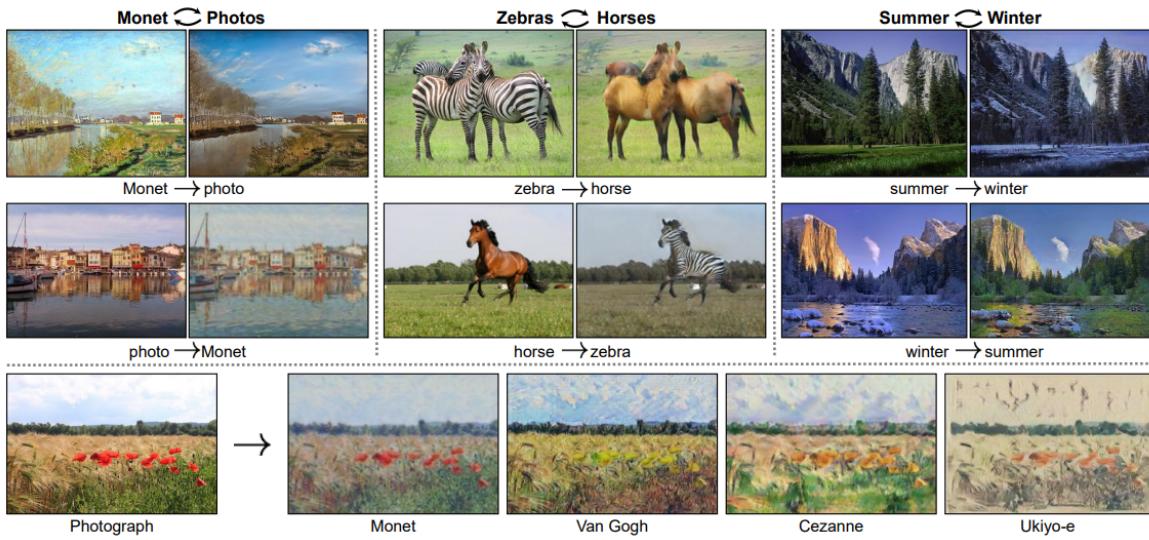


FIGURE 11 – Exemples d'image-to-image translation avec CycleGAN

Les applications sont très variées, la translation peut s'effectuer directement d'image vers label, comme vu dans l'image ci-dessus, pour reproduire les résultats d'un réseau de segmentation sémantique. Dans notre cas, la tâche est un transfert de style, du style “synthétique” (ou “virtuel”) au style “réel”. De nombreux réseaux peuvent effectuer cette tâche, mais l'un des plus connus est nommé CycleGAN[6], que nous détaillerons plus tard. On peut transférer des images du domaine source au domaine cible et entraîner le réseau devant effectuer la tâche dessus, ou bien s'entraîner sur la source directement, et lors des évaluations/inférences, effectuer le transfert de style des images “cible” vers source.

Cas	Ensemble d'entraînement	Ensemble d'évaluation
Premier cas	Images synthétiques stylisées comme une image réelle	Images réelles
Second cas	Images synthétiques	Images réelles stylisées comme une image synthétique

TABLE 1 – Les deux modes d'entraînement dans notre cas particulier

Nous allons tester ces deux méthodes, mais nous nous concentrerons sur la seconde.

## 4 Image-to-Image translation

### 4.1 Présentation

### 4.2 GAN

Avant de parler de CycleGAN, il faut d'abord expliquer ce qu'est un GAN, qui signifie Generative Adversarial Network[7]. L'idée du GAN est publiée en 2014, et marque un bond dans la génération d'image par réseaux de neurones, grâce à son inventivité et sa versatilité. Le principe est le suivant :

Le GAN se compose de deux réseaux, un générateur et un discriminateur. Le générateur doit générer une image en prenant en entrée un bruit aléatoire, tandis que le discriminateur doit distinguer de quelle distribution cette image provient.

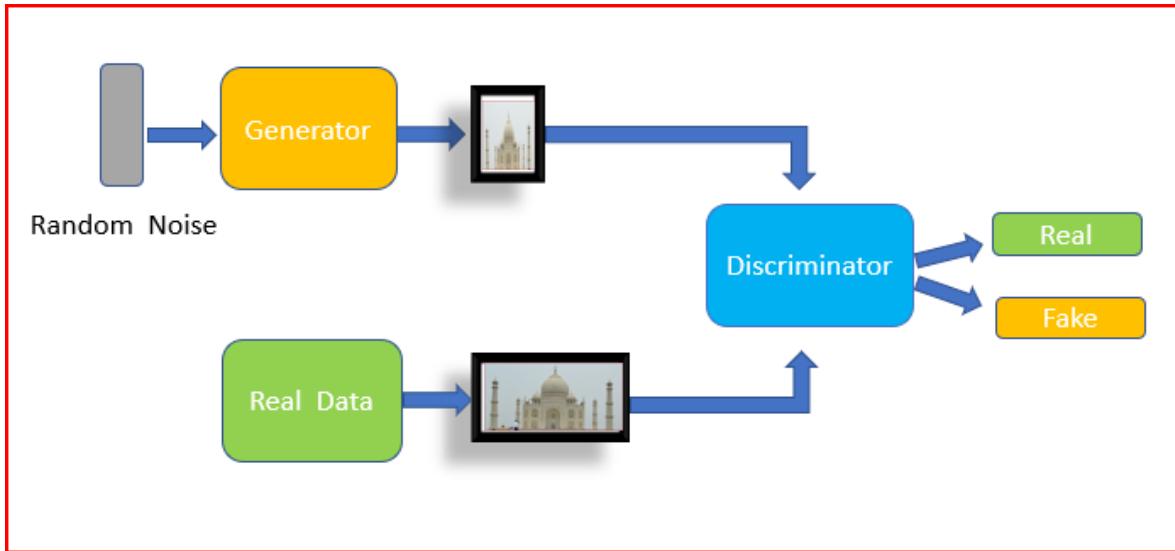


FIGURE 12 – Fonctionnement d'un GAN

Les deux réseaux s'entraînent de en fait de manière opposée, ils sont "adversaires", d'où le nom du modèle. Le but d'un GAN est donc d'optimiser la fonction suivante :

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

où  $x$  correspond à une image réelle,  $z$  à une image générée,  $G$  au générateur, et  $D$  au discriminateur.

### 4.3 CycleGAN

Le CycleGAN est lui composé de deux GAN. Rappelons que sa tâche est d'effectuer une translation d'image d'un domaine "source" vers un domaine "cible". Le rôle du premier GAN est de générer une image du domaine "cible" à partir d'une image du domaine "source". Le second doit faire la tâche inverse, recréer l'image de base à partir de la première image transformée. Cela nous permet d'obtenir une fonction objectif composée de trois parties : la fonction objectif du premier GAN, celle du second, et celle de la similarité entre l'image de base et l'image recréée.

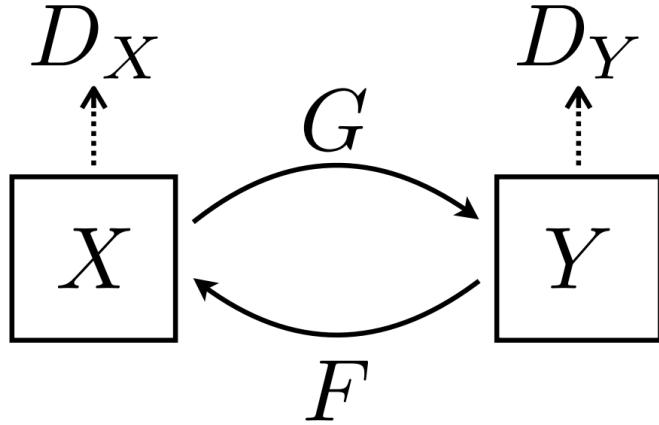


FIGURE 13 – Fonctionnement du CycleGAN

Ici, G correspond au premier Générateur, censé créer une image dans le domaine Y à partir d'une image du domaine X, tandis que F correspond au générateur qui doit effectuer la tâche inverse. Les deux discriminateurs  $D_X$  et  $D_Y$  doivent reconnaître si une image provient de leur domaine respectif.

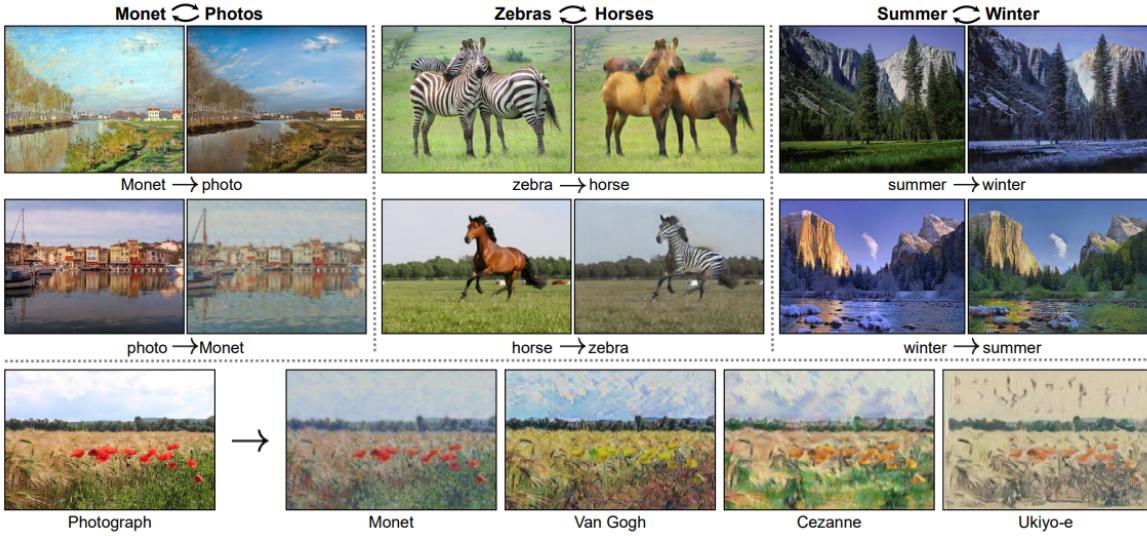


FIGURE 14 – Exemples d'image-to-image translation avec CycleGAN

On peut ainsi obtenir la fonction objectif suivante

$$\mathcal{L}(G, F, D_X, D_Y) = \mathcal{L}_{GAN}(G, D_Y, X, Y) + \mathcal{L}_{GAN}(F, D_X, Y, X) + \lambda \mathcal{L}_{cyc}(G, F)$$

où  $\mathcal{L}_{GAN}$  correspond à la fonction objectif du GAN, vue plus haut, et où  $\mathcal{L}_{cyc}$  correspond à la fonction de perte cyclique, c'est-à-dire aux différences entre les images reconstituées.

## 4.4 Cycada

Cycada[8] permet de pousser encore plus loin le principe du CycleGAN, mais pour effectuer une tâche particulière. L'ensemble est constitué de plusieurs réseaux :

- Un CycleGAN (lui-même composé de deux GAN, comme vu précédemment)
- Deux réseaux de “tâche”, par exemple de la segmentation sémantique
- Deux discriminateurs

On rappelle que l'ensemble source correspond à l'ensemble d'images synthétique, et que l'ensemble cible correspond à l'ensemble d'images réelles. L'entraînement comporte trois étapes :

- On dispose d'un réseau de tâche pré-entraîné sur l'ensemble source des images  $f_S$ . On entraîne le CycleGAN de manière classique, sauf qu'à sa fonction objectif nous rajoutons un nouveau terme qui est une fonction de consistance sémantique. Celle-ci est calculée en mesurant la différence entre la prédiction du réseau tâche  $f_S$  sur l'image source et celle sur l'image source transformée en cible. Cela permet de conserver le contenu sémantique de l'image lors de la transformation.
- La deuxième étape consiste à entraîner le second réseau de tâche sur l'ensemble source stylisé en l'ensemble cible. On obtient ainsi le réseau nommé  $f_T$ .
- La troisième et dernière étape est la plus spécifique. Elle consiste à utiliser un discriminateur  $D_{feat}$  appliqué sur les sorties de  $f_T$  prenant en entrée les images source stylisées en cible et les images cible. En fait, on peut assimiler cette dernière étape à l'entraînement d'un GAN, sauf que le générateur est ici le réseau  $f_T$ , et que le discriminateur  $D_{feat}$  s'applique sur des cartes sémantiques. Son rôle est de déterminer quelles sont les cartes provenant d'image source stylisées en cible (synthétiques transformées en réelles), et quelles sont celles provenant d'image cible (réelles).

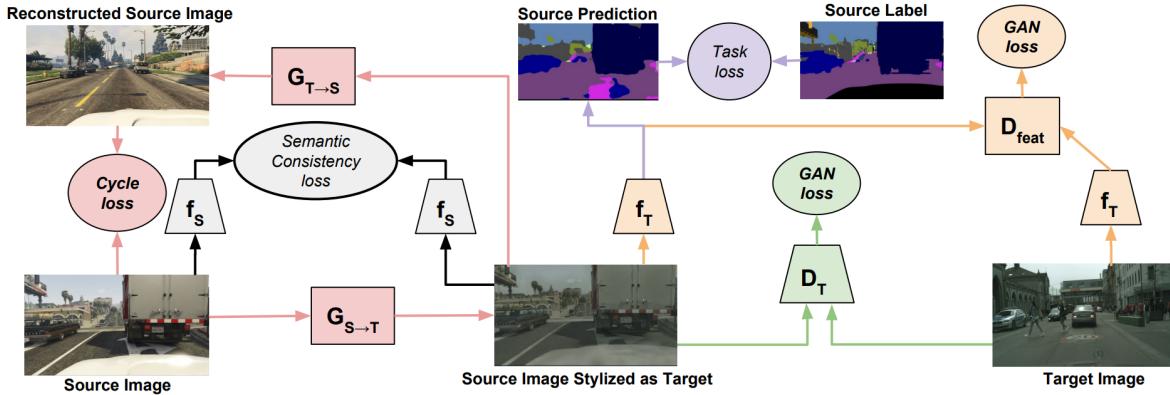


FIGURE 15 – Schéma du fonctionnement de CyCADA

CyCADA consiste donc plus en une méthode d'entraînement qu'un réseau bien spécifique, et le schéma ci-dessus peut aider à comprendre comment elle fonctionne. Les parties en rouge, gris, et vert correspondent à la première étape, la partie en violet correspond à la deuxième, et la partie en orange à la troisième.

On peut ainsi mettre la fonction objectif de CyCADA sous la forme suivante :

$$\begin{aligned}
\mathcal{L}_{\text{CyCADA}}(f_T, X_S, X_T, Y_S, G_{S \rightarrow T}, G_{T \rightarrow S}, D_S, D_T) \\
&= \mathcal{L}_{\text{task}}(f_T, G_{S \rightarrow T}(X_S), Y_S) \\
&\quad + \mathcal{L}_{\text{GAN}}(G_{S \rightarrow T}, D_T, X_T, X_S) + \mathcal{L}_{\text{GAN}}(G_{T \rightarrow S}, D_S, X_S, X_T) \\
&\quad + \mathcal{L}_{\text{GAN}}(f_T, D_{\text{feat}}, f_S(G_{S \rightarrow T}(X_S)), X_T) \\
&\quad + \mathcal{L}_{\text{cyc}}(G_{S \rightarrow T}, G_{T \rightarrow S}, X_S, X_T) + \mathcal{L}_{\text{sem}}(G_{S \rightarrow T}, G_{T \rightarrow S}, X_S, X_T, f_S)
\end{aligned}$$

FIGURE 16 – Fonction objectif de CyCADA

où :

- la seconde et la dernière ligne correspondent à la première étape de l’entraînement, avec les objectifs d’un CycleGAN et celui de la tâche
- la première ligne correspond à la partie violette du schéma, donc la fonction de perte de la deuxième étape
- la troisième ligne correspond à la troisième étape en orange, donc la fonction objectif d’un GAN, mais où le générateur est remplacé par le réseau effectuant la tâche  $f_T$

Toutefois, comme il est précisé dans le papier de CyCADA, un CycleGAN est assez lourd en mémoire, donc la première étape peut être assez demandante en ressources. C’est notamment le cas pour la segmentation sémantique, car les réseaux de segmentation comme DeepLabV3Plus (celui utilisé dans notre cas) sont très profonds. La consistance sémantique n’est donc pas utilisée en pratique lors de la première étape, comme précisé dans le papier, et comme lors de nos expérimentations.

## 4.5 TSIT

Un autre réseau, plus récent, nommé TSIT[9] (Two-Stream Image-to-Image Translation) permet de faire du transfert de style. Ce dernier est plus léger puisqu’il ne comporte pas de mécanisme cyclique comme c’est le cas avec CycleGAN. Le principe de ce réseau repose sur deux “stream”, donc deux parties du réseau qui ont chacune un objectif bien distinct. Le réseau reçoit en entrée une paire d’image, une image définissant le contenu, et une autre définissant le style. La sortie conserve le contenu de la première image, et le style de la seconde. Dans notre cas, cela revient à dire que l’image disposant du contenu est l’image réelle, et qu’on veut l’adapter au style synthétique, en fournissant une image synthétique au réseau. On effectue donc un transfert de l’ensemble cible vers l’ensemble source, afin de pouvoir appliquer un réseau de segmentation entraîné sur ce même ensemble source. Regardons plus en détail de quoi est composé ce réseau

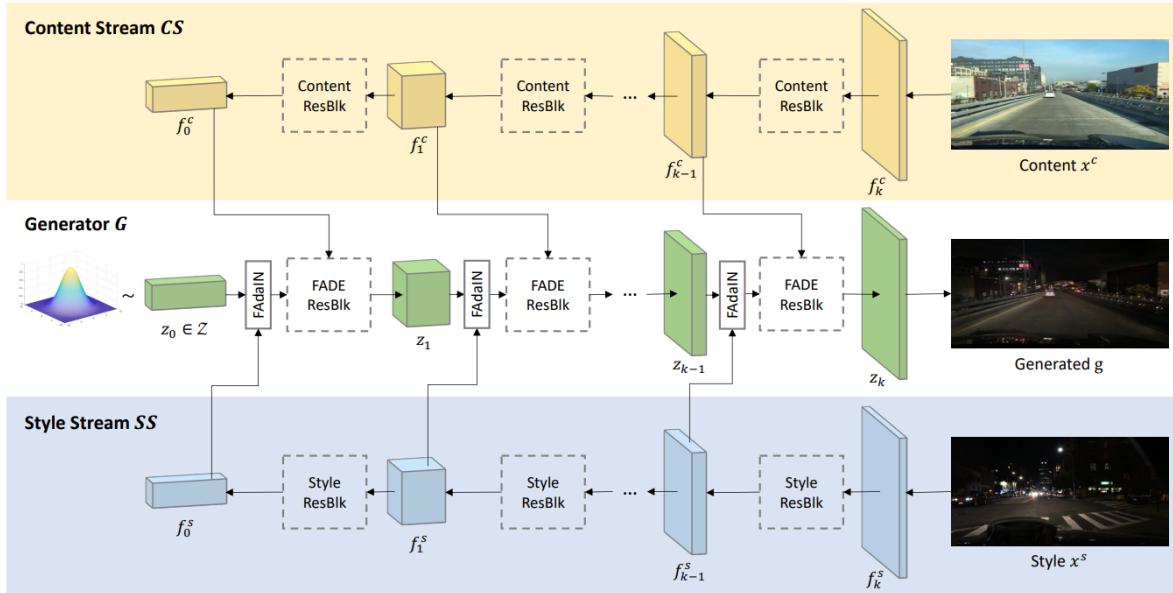


FIGURE 17 – Architecture de TSIT

Les deux stream du haut et du bas sont composés de bloc "ResNet" [10] pour Residual Network, qui sont un type de réseau très populaire en reconnaissance d'image. Nous n'entrerons pas dans les détails mais ces derniers servent à extraire les caractéristiques, ou "features" de l'image. Le générateur prend en entrée un tenseur correspondant à un bruit Gaussien, et en sortie renvoie l'image générée. Le tenseur passe par un certain nombre de couche, composée de deux modules importants :

- FadaIN, qui apporte les informations de style
- FADE, qui apporte les informations sur le contenu

Décrivons d'abord FadaIN (Feature Adaptative Instance Normalization). À chaque couche d'un réseau, il est d'usage de normaliser le tenseur, et il existe pour cela plusieurs techniques. La normalisation peut s'effectuer à plusieurs niveaux, au niveau de l'image (Instance Normalization), ou au niveau du batch (Batch Normalization). Ensuite, le tenseur normalisé peut-être aligné avec une autre moyenne et une autre variance. C'est ce qui se fait avec l'AdaIN[11] (Adaptative Instance Normalization) :

$$\text{AdaIN}(x, y) = \sigma(y) \left( \frac{x - \mu(x)}{\sigma(x)} \right) + \mu(y)$$

Ici l'instance  $x$  est normalisée, puis alignée avec la moyenne et l'écart-type de l'entrée  $y$ . Le terme "Feature" de FadaIN indique que l'on se situe dans l'espace des features (caractéristiques en Français). Concrètement, cela signifie que l'on ne traite pas l'image en tant que telle, mais que l'on traite une image étant déjà passé par un encodeur, les encodeurs étant ici les blocs des streams de style et de contenu. On peut ainsi aligner le tenseur du stream "générateur" avec celui du stream "style". Formellement, on peut l'écrire comme suit :

$$\text{FadaIN}(z_i, f_i^s) = \sigma(f_i^s) \left( \frac{z_i - \mu(z_i)}{\sigma(z_i)} \right) + \mu(f_i^s)$$

Ici  $z_i$  désigne l'entrée se situant dans la partie génératrice du réseau, et  $f_i^s$  l'entrée provenant des caractéristiques de style, donc provenant d'images réelles. Ce bloc FadaIN

est ainsi destiné à fournir les informations du style de l'image, puisqu'il agit au niveau global en moyennant sur toute l'image, et perd ainsi la majorité des informations sur le contenu.

Regardons maintenant le bloc FADE (Feature Adaptive Denormalization). Ce bloc applique une “batch normalization”, donc une normalisation au niveau de tout le batch, puis le batch est aligné avec des paramètres qui sont fournies par des couches de convolutions prenant en entrée les images de contenu, donc les images réelles. A l'inverse de FadaIN qui calcule simplement la moyenne et la variance d'une image, puis qui aligné l'entrée selon ces valeurs, FADE se permet plus de précisions en utilisant des techniques d'apprentissage, et donc en conservant plus d'informations sur le contenu de l'image. La formule correspondante est la suivante :

$$\gamma_i^{l,h,w} \frac{z_i^{n,l,h,w} - \mu_i^l}{\sigma_i^l} + \beta_i^{l,h,w}$$

Où  $\mu_i^l$  et  $\sigma_i^l$  sont les espérances et écart-type du batch entier, et non plus seulement d'une instance :

$$\mu_i^l = \frac{1}{NH_iW_i} \sum_{n,h,w} z_i^{n,l,h,w}$$

$$\sigma_i^l = \sqrt{\frac{1}{NH_iW_i} \sum_{n,h,w} (z_i^{n,l,h,w})^2 - (\mu_i^l)^2}$$

Les paramètres  $\gamma_i^{l,h,w}$  et  $\beta_i^{l,h,w}$  sont, comme dit plus haut, fournis par des couches de convolution.

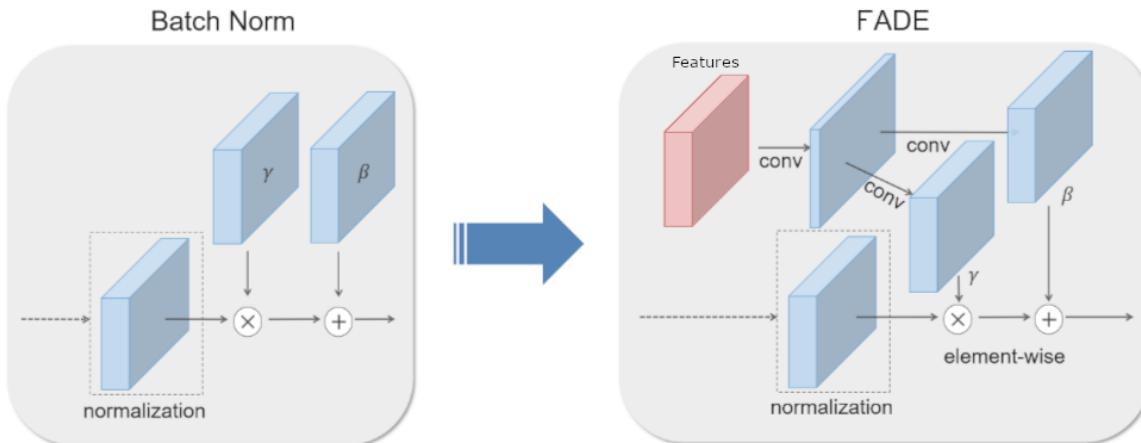


FIGURE 18 – Schéma de FADE

## 5 Mes contributions

### 5.1 Le choix du dataset

Il a tout d'abord fallu choisir un ensemble de données sur lesquelles travailler. Mon stage étant orienté vers la profondeur au départ, j'ai dû choisir un ensemble de données qui fournissent des labels de profondeur, et qui disposaient également d'un ensemble synthétique lui ressemblant.

Dans le domaine des paysages urbains, les datasets les plus connus sont Cityscapes[12] pour les images réelles, et GTA[13] et Synthia[14] pour les images synthétiques.



FIGURE 19 – Exemple d'image de Cityscapes

Les images de Cityscapes sont des photos prises dans différentes villes du monde. Elles sont largement utilisées dans le domaine de la segmentation sémantique, toutefois il n'existe pas d'annotation de profondeurs pour ces images, puisque celles-ci requièrent des capteurs lorsque la photo est effectuée, et il est impossible d'avoir des valeurs réelles après coup. Il n'était donc pas envisageable de travailler avec cette ensemble de données.

Regardons maintenant les ensembles synthétiques, GTA et Synthia



FIGURE 20 – Exemple d'image de GTA



FIGURE 21 – Exemple d'image de Synthia

L'ensemble GTA pour commencer, provient de la modélisation d'une ville utilisée dans le jeu vidéo GTA 5 (Grand Theft Auto 5). En effet le jeu est loué pour son réalisme, donc certains chercheurs ont l'idée de l'utiliser pour obtenir des données synthétiques. En effectuant des modifications au jeu, ils ont pu récupérer des images avec des informations sémantiques directement dans le jeu. Malheureusement, les cartes de profondeur ne sont pas disponibles pour cet ensemble, bien qu'il serait probablement possible de les obtenir en modifiant le jeu, toutefois cela prendrait beaucoup trop de temps, et nécessiterait un sujet à part entière.

L'ensemble Synthia quant à lui, ne comporte pas non plus de cartes de profondeur, en plus d'être, comme on peut le voir sur les photos, moins réaliste, plus lisse que GTA. Il a donc fallu se tourner vers les ensembles utilisées en prédiction de profondeur.

C'est là qu'intervient l'ensemble KITTI[15], composé de plusieurs scènes vidéos prises dans des paysages urbains, avec des capteurs mesurant diverses choses, dont la profondeur. Il existe un ensemble synthétique nommé Virtual Kitti[16] (abrégé vKitti), dont l'objectif est de recréer certaines scènes de Kitti avec le moteur graphique Unity.



FIGURE 22 – Exemple d'image de Kitti (à gauche) et virtual Kitti (à droite)

En fait nous utilisons la version 2 de virtual Kitti, qui utilise une version plus récente d'Unity, mais le principe reste le même.

Cet ensemble est peu utilisé dans le cadre de la segmentation sémantique, les privilégiés étant Cityscapes, Synthia, GTA. Toutefois, même si nous avons bifurqué vers de la segmentation sémantique, nous verrons plus tard pourquoi il nous est tout de même utile d'avoir des informations sur la profondeur d'une image.

## 5.2 Premiers essais avec DeepLabV3Plus

Nous allons tout d'abord entraîner le réseau de segmentation sémantique DeepLabV3Plus sur l'ensemble virtual Kitti, et voir les résultats que l'on obtient avec cela.

Ensemble d'entraînement	Ensemble d'évaluation	Résultat : mIoU
vKitti	vKitti	53.8
vKitti	Kitti	33.9
vKitti + Kitti	Kitti	49.0

TABLE 2 – Les deux modes d'entraînement dans notre cas particulier

Il faut là donner quelques précisions sur les modalités de ces évaluations. Le dataset Kitti ne dispose que de peu d'images ayant des cartes de segmentation sémantique (200 images). 30 sont sélectionnées pour l'évaluation, et les 170 autres peuvent être utilisées pour l'entraînement. Lorsque l'ensemble d'évaluation est "Kitti", cela signifie que l'on a effectué l'évaluation sur ces 30 images. Lorsque l'ensemble d'évaluation est "vKitti + Kitti", cela signifie que le réseau a été entraîné sur vKitti, puis "fine-tuned" (affiné) sur les 170 images disponibles de Kitti pour la segmentation sémantique.

Il faut noter que le score de 53.8 n'est pas exceptionnel pour de la segmentation sémantique, c'est probablement car le dataset vKitti est plus adapté à la prédiction de profondeur. La deuxième ligne du tableau nous donne la borne inférieure des valeurs acceptables pour de l'adaptation de domaine, puisqu'elle correspond au réseau appliqué

directement sur l'ensemble cible (Kitti, images réelles), sans aucune transformation. Si l'application d'une transformation résulte en un résultat moins bon, alors c'est qu'il faut changer quelque chose.

La troisième ligne correspond à une borne supérieure. En effet, il risque d'être difficile de faire mieux que cela, puisque le réseau a cette fois-ci été affiné sur l'ensemble cible (Nous ne sommes donc plus dans de l'adaptation de domaine, mais ce résultat reste utile pour notre information). On peut voir cela comme le but théorique de cette adaptation de domaine.

Il est maintenant temps d'appliquer les techniques d'image-to-image translation dont nous avons parlé précédemment.

### 5.3 Essais avec CycleGAN

Cette fois-ci nous allons effectuer les mesures sur des images transformées avec CycleGAN. Il faut savoir que lors de l'entraînement de CycleGAN, la fonction de perte utilisée n'a rien à voir avec la segmentation sémantique, puisque c'est  $\mathcal{L}_{GAN}$  décrit précédemment. Il est donc tout à fait possible que le réseau devienne moins bon pour la tâche que nous sommes en train de réaliser (Adaptation de domaine pour de la segmentation sémantique). Pour palier à ce problème, on enregistre l'état du réseau à différents paliers de l'entraînement, et on regarde ce que chacun donne. On essaye également avec différents "Learning Rate" (LR) initiaux, ou "taux d'apprentissage", qui correspondent à la vitesse à laquelle le réseau apprend lors de son entraînement. Dans la plupart des cas ce Learning Rate diminue au cours du temps, en commençant avec une valeur donnée en paramètre.

En plus de regarder le mIoU, on regarde également le FID, pour Fréchet Inception Distance. Celle-ci correspond à cette formule :

$$d_F(\mu, \nu) := \left( \inf_{\gamma \in \Gamma(\mu, \nu)} \int_{\mathbb{R}^n \times \mathbb{R}^n} \|x - y\|^2 d\gamma(x, y) \right)^{1/2}$$

où  $\mu$  et  $\nu$  correspondent à des distributions de probabilités. Le FID est une manière de calculer la distance entre ces distributions. En pratique, avec des images cela sert à voir si deux ensemble sont "éloignés" l'un de l'autre. Les résultats sont les suivants :

<b>Etape de cycleGan</b>	<b>mIoU</b>	<b>FID</b>
10	26,7	199,51
20	37,0	191,70
30	36,9	197,55
40	33,8	185,81
50	32,6	195,65
60	33,5	188,16
70	32,9	200,62
80	31,6	190,82
90	31,0	207,55
100	33,8	201,85
110	33,2	204,47
120	33,0	203,28
130	32,5	201,26
140	33,3	195,85
150	31,0	192,75
160	31,5	199,44
170	32,1	197,67
180	29,5	206,24
190	31,0	206,08
200	31,2	206,40

TABLE 3 – Résultats de l'adaptation de domaine avec CycleGAN, LR de base

Et voici maintenant les résultats des entraînements avec des Learning Rate modifiés :

Etape de cycleGan	mIoU (LR de base /10)	FID (LR de base /10)
10	25,6	305,2755356
20	27,7	279,2071737
30	30,3	272,2806777
40	29,3	276,0050411
50	31,4	275,0106515
60	32,2	279,4146519
70	31,1	276,9334005
80	31,4	278,4594763
90	28,9	286,2291014
100	33,0	276,9749375
110	30,3	273,3796292
120	29,1	273,6501333
130	29,4	273,8609189
140	31,4	273,2653737
150	30,9	264,9101918
160	31,8	269,0417083
170	31,1	274,1760951
180	31,2	264,1115304
190	31,8	269,4719194
200	31,4	268,3807711

TABLE 4 – Résultats de l’adaptation de domaine avec CycleGAN, LR de base divisé par 10

Etape de cycleGan	mIoU (LR de base x10)
10	00,0
20	15,7
30	30,1
40	28,7
50	26,3
60	26,8
70	14,6
80	26,1
90	28,6
100	27,5
110	23,4
120	29,4
130	23,4
140	27,8
150	24,6
160	24,2
170	24,3
180	23,8
190	24,8
200	24,6

TABLE 5 – Résultats de l’adaptation de domaine avec CycleGAN, LR de base multiplié par 10

On observe que multiplier ou diviser le Learning Rate par 10 n’améliore pas le mIoU maximum, lorsqu’il est multiplié par 10 les résultats sont tellement mauvais que nous n’avons pas pris la peine de calculer les FID. Le mIoU maximum obtenu est de 37,0 avec la vingtième étape du CycleGAN au Learning Rate de base, c’est-à-dire 0,0002. On améliore ainsi de plus de 3 points le mIoU obtenu.

## 5.4 Essais avec Cycada

Nous allons donc utiliser CyCADA en suivant les différentes étapes de l’entraînement décrites plus haut. Nous n’utiliserons pas la consistance sémantique pour l’entraînement de CycleGAN pour les raisons précisées plus haut, c’est-à-dire les problèmes de mémoire. A savoir que le réseau de segmentation utilisé n’est pas DeepLabV3Plus, mais un réseau plus générique fourni dans le code de CyCADA.

On obtient un mIoU de 37,6, c’est donc un peu meilleur qu’un CycleGAN simple, mais nous allons voir que l’on peut encore améliorer ce résultat avec les méthodes suivantes.

## 5.5 Essais avec TSIT

Pour la suite nous allons utiliser le réseau TSIT présenté plus haut, plus récent et plus léger. Regardons tout d’abord ce que nous obtenons en effectuant les mesures comme précédemment, c’est-à-dire appliquer le réseau de segmentation sémantique sur

les images de l'ensemble cible (réel) transformées en l'ensemble source (synthétique). Voici ce que nous obtenons pour le mIoU et le FID :

<b>Etape de TSIT</b>	<b>mIoU</b>	<b>FID</b>
20	28,9	251,51809463790602
40	25,0	262,1420693002249
60	22,3	255,36185557065937
80	23,8	272,1276034130048
100	26,0	247,87397161083996
120	24,8	241,25455428320817
140	27,0	245,59937463581957
160	29,6	250,8364980667324
180	25,6	259,0358833287116
200	28,7	257,3396797754418

TABLE 6 – Résultats de l'adaptation de domaine avec TSIT

Les résultats sont ici largement moins bons qu'avec CycleGAN et Cycada, nous allons donc essayer avec d'autres valeurs de Learning Rate, comme ce qu'on a fait plus haut, celui de base valant 0,0002 (comme CycleGAN).

<b>Etape de TSIT</b>	<b>mIoU (LR de base /10)</b>	<b>FID (LR de base /10)</b>
20	28,6	269,0958196798722
40	26,9	262,0041166036592
60	29,9	252,43690790871693
80	26,8	265,8247167666007
100	28,0	265,6329012124063
120	27,4	259,25436748607234
140	24,3	256,9115299255891
160	31,3	260,0933664758801
180	31,8	265,2698516611955
200	25,6	253,61251122872437

TABLE 7 – Résultats de l'adaptation de domaine avec TSIT, LR de base divisé par 10

Etape de TSIT	mIoU (LR de base x10)	FID (LR de base x10)
20	27,5	255,86578601891657
40	26,8	259,54178739467113
60	23,5	253,74373352961922
80	25,7	263,18637380798987
100	24,4	268,48274951098045
120	26,4	261,55311332785675
140	23,8	263,60431856751075
160	22,5	255,835130290146
180	23,0	266,409656174795
200	23,1	269,27091749242453

TABLE 8 – Résultats de l’adaptation de domaine avec TSIT, LR de base multiplié par 10

C’est donc à l’étape 180, et avec un Learning Rate divisé par 10 que l’on obtient le meilleur mIoU, de 31,8. Bien que ce résultat reste moins bon que ceux de CycleGAN et CyCADA, nous allons conserver TSIT pour effectuer les modifications, car celui-ci est léger et relativement simple.



FIGURE 23 – Exemple de transformation (Réel - CycleGAN - TSIT)

## 5.6 Modifications

Il est maintenant temps d’essayer d’améliorer ces résultats, en modifiant le réseau à des endroits stratégiques.

### 5.6.1 Consistance sémantique

La première modification a été, à l’instar de CyCADA, d’ajouter une fonction de consistance sémantique à la génération d’image (une cross-entropy loss pour être précis). Comme précisé au dessus, TSIT est plus léger que CycleGAN, il est donc possible de le charger aux côtés un modèle de segmentation sémantique. La fonction objectif du générateur est ainsi pondérée par cette cross-entropy par une valeur  $\alpha$ .

$$\mathcal{L} = \alpha \mathcal{L}_{task} + (1 - \alpha) \mathcal{L}_G$$

Cette fois-ci, comme on peut charger les deux réseaux en même temps, on peut effectuer les évaluations directement pendant l’entraînement, et pas après coup comme on faisait avant. De ce fait on peut garder en mémoire les meilleures valeurs de mIoU. Nous obtenons les résultats suivants :

Valeur de $\alpha$	Résultat : mIoU
0.5	34
0.9	34
1.0	35

TABLE 9 – Résultats de l’adaptation de domaine avec TSIT et consistance sémantique

De manière assez surprenante, les meilleurs résultats sont obtenus avec  $\alpha = 1$ , c’est-à-dire lorsque l’on supprime totalement la perte du générateur, et qu’on ne laisse que la perte sémantique. Cela donne en résultat des images avec un masque gris transparent par dessus, mais la technique suivante palie à ce problème.

### 5.6.2 Histogram matching

L’histogramme matching, ou correspondance d’histogramme en français, consiste à modifier l’histogramme de couleur d’une image pour le faire correspondre à celui d’une autre image. L’histogramme de couleur indique, pour chacune des trois couleurs Rouge, Vert et Bleu (couleurs du format RVB) le nombre de pixel correspondant à une certaine intensité.

L’algorithme utilisé pour cela ne sera pas détaillé ici, toutefois il faut savoir que le résultats n’est qu’une approximation. Qualitativement, utiliser cette technique permet d’ajuster les teintes de couleurs d’une image à celles d’une autre image

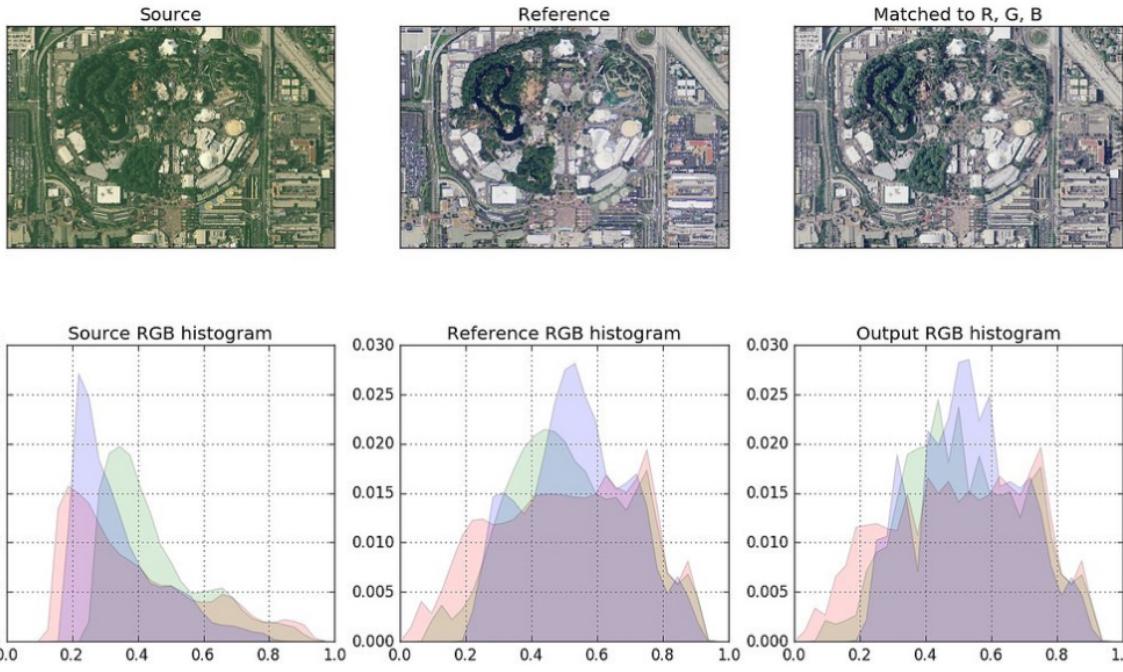


FIGURE 24 – Histogram Matching

Une des différences entre nos deux ensembles source (synthétique, vKITTI) et cible (Réel, Kitti) réside dans la teinte des images, les images synthétiques étant un peu plus claires, et un peu plus saturées que les images réelles. De ce fait on peut utiliser cette technique pour espérer obtenir de meilleurs résultats.



FIGURE 25 – Exemple de transformation avec consistance sémantique et Histogram Matching (réel- TSIT( $\alpha = 1$ )-Histogram Matching

### 5.6.3 SPADE

Le module FADE présent dans TSIT, et décrit plus haut, s'inspire d'un module nommé SPADE, pour Spatially Adaptive Normalization. Ce dernier est assez similaire à FADE sauf qu'au lieu de calculer la moyenne et l'écart-type sur lesquels aligner l'entrée à partir de «features», il les calcule à partir de la carte de segmentation.

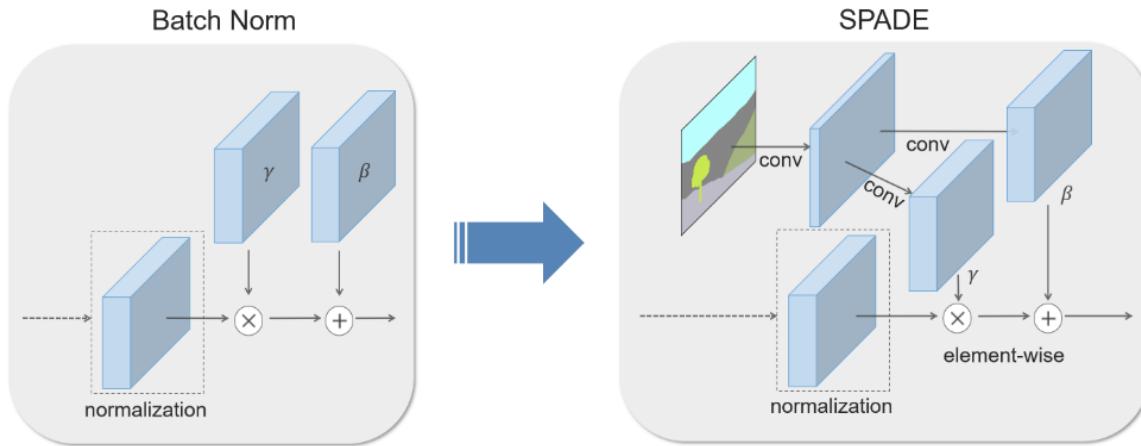


FIGURE 26 – Schéma de SPADE

Rappelons que FADE utilise les features provenant de l'image réelle, donc celle qui fournit le contenu, et surtout celle pour laquelle on ne dispose pas de carte de segmentation. On ne peut donc pas utiliser SPADE avec la carte de segmentation du contenu, puisque cela reviendrait à faire de l'apprentissage supervisé. Nous avons donc tenté deux approches différentes pour intégrer SPADE au réseau TSIT :

La première consiste à utiliser SPADE avec les cartes de segmentation de l'ensemble synthétique. Bien que ce dernier ne serve pas à indiquer le contenu, mais seulement le style, on peut tout de même espérer que les informations supplémentaires aide le tout à obtenir de meilleurs résultats. Une technique courante est d'utiliser un encodage "one-hot" pour les carte de segmentation. Cela correspond à remplacer les entiers désignant la classe par un vecteur de 0 avec un 1 à l'indice de la classe. Par exemple (en considérant qu'il y a 5 classes au total) :

$$\begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Cette technique donne généralement de meilleurs résultats.

Pour des raisons de cohérence, le résultat de ce SPADE sera combiné linéairement avec le résultats du FadaIN, les deux étant chargés de récupérer des informations dans les images cible.

<b>mIoU sans Histogram Matching</b>
33.4

TABLE 10 – Résultats avec  $\alpha = 0$  et SPADE combiné à FadaIN

On observe que cela n'améliore pas les résultats. Il est donc probable que les informations apportées par les pseudo-label soient redondantes par rapport à celles extraites par le stream de style. Nous allons donc passer à la deuxième approche

La deuxième approche consiste à utiliser les cartes de profondeur disponibles pour KITTI c'est-à-dire l'ensemble source, fournissant les informations de contenu. Certes, nous ne sommes pas censés disposer d'informations sémantiques sur cet ensemble, mais on peut supposer qu'en situation réelle (robot terrestre, voiture autonome...), on dispose de capteurs de profondeurs, bien plus faciles à installer qu'un dispositif de segmentation sémantique (quasiment exclusivement des réseaux de neurone).

Il a plusieurs manières de formater les cartes de profondeur pour les donner à SPADE. La première est de donner directement les valeurs telles quelles, toutefois SPADE étant pensé pour la segmentation sémantique, il est préférable de découper la profondeur en intervalles (ici logarithmiques, et fixe pour l'ensemble de données), où chaque pixel appartient à un intervalle. On obtient donc des données qui ressemblent à des cartes de segmentation. On peut même appliquer un encodeur "one-hot", comme dans la première approche. On peut ensuite combiner la sortie de SPADE avec celle de FADE. Regardons les résultats obtenus :

<b>mIoU sans Histogram Matching</b>	<b>mIoU avec Histogram Matching</b>
38.7	39.3

TABLE 11 – Résultats avec  $\alpha = 0$  et SPADE combiné à FADE

Les résultats obtenus sont les meilleurs jusqu'ici. La carte de profondeur apporte bel et bien des informations sémantiques, et permet bien de constituer une image plus précise.

## 5.7 Récapitulatif des modifications et de leur résultats

Finalement, on peut obtenir le tableau suivant

$\alpha$	HM	SPADE + FadaIN	SPADE + FADE	Résultat : mIoU
0.0	Non	Non	Non	33.8
0.0	Oui	Non	Non	35.0
0.5	Non	Non	Non	33.4
1.0	Non	Non	Non	37.7
1.0	Oui	Non	Non	38.8
1.0	Non	Oui	Non	34.1
1.0	Oui	Oui	Non	36.7
1.0	Non	Non	Oui	38.7
1.0	Oui	Non	Oui	39.3

TABLE 12 – Récapitulatif des résultats avec les différentes modifications

La méthode la plus efficace est d'utiliser SPADE et FADE combiné, avec  $\alpha = 1$  ainsi que la technique d'Histogram Matching. Les cartes sémantiques de l'ensemble source (synthétique) ne fournissent pas plus d'informations sémantique, ce qui est cohérent puisque les images synthétique sont seulement censées fournir des informations de style et non de contenu. Par contre, les carte de profondeur des images de l'ensemble cible (réel) fournissent elles bien des informations sémantiques, et cela se ressent dans les résultats. Notons aussi que l'Histogram Matching améliore toujours les résultats. Cela se comprend aisément lorsque l'on regarde les teintes des couleurs de virtual Kitti et Kitti, qui sont relativement éloignées.

## 6 Conclusion

### 6.1 Mon expérience personnelle

Ce stage a été pour moi une première expérience dans le monde de la recherche. Cela m'a donné un bon aperçu de ce à quoi peut ressembler la vie d'un doctorant, d'un post-doctorant ou d'un chercheur. J'ai pu discuter avec beaucoup du personnel de l'U2IS et cela a été très enrichissant, en tant qu'étudiant encore incertain sur l'issue de ses études. J'ai été très content de pouvoir découvrir ce à quoi ça ressemblait, et plus généralement à quoi ressemblait le monde du travail, même s'il est ici différent de celui d'une entreprise privée.

### 6.2 Travail mené et à venir

Nous avons pu, à travers ce projet, utiliser des techniques d'adaptation de domaine sur un dataset moins commun nommé KITTI. Le fait que ce dataset comporte des données de profondeur est vraiment important puisque c'est ce qui le distingue des autres, et surtout c'est ce qui permet d'utiliser une technique améliorant bien les résultats. En effet le stage s'est orienté vers l'image-to-image translation en observant ce qui se faisait pour la profondeur, et nous avons pu dépasser des techniques très connues dans ce domaine (CycleGAN, CyCADA) en améliorant le réseau TSIT, en comprenant bien quels étaient les rôles de chacun de ces composants. Les derniers résultats ont été obtenus à la fin du stage, nous avons donc manqué de temps pour appliquer d'autres idées, mais il est certains que l'on pourrait faire d'autres expérimentations à partir de cela. On pourrait par exemple appliquer le principe de CyCADA mais en utilisant TSIT à la place de CycleGAN, et en utilisant effectivement la consistance sémantique.

## Références

- [1] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2018.
- [2] Keiron O’Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.
- [3] Hossein Gholamalinezhad and Hossein Khosravi. Pooling methods in deep neural networks, a review, 2020.
- [4] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018.
- [5] Shariq Farooq Bhat, Ibraheem Alhashim, and Peter Wonka. AdaBins : Depth estimation using adaptive bins. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, jun 2021.
- [6] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.
- [7] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.
- [8] Cycada : Cycle consistent adversarial domain adaptation. In *International Conference on Machine Learning (ICML)*, 2018.
- [9] Liming Jiang, Changxu Zhang, Mingyang Huang, Chunxiao Liu, Jianping Shi, and Chen Change Loy. TSIT : A simple and versatile framework for image-to-image translation. In *ECCV*, 2020.
- [10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
- [11] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization, 2017.
- [12] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding, 2016.
- [13] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data : Ground truth from computer games, 2016.
- [14] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. The synthia dataset : A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
- [15] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics : The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.
- [16] Yohann Cabon, Naila Murray, and Martin Humenberger. Virtual kitti 2, 2020.