École Normale Supérieure Paris-Saclay

Parcours Intelligence Artificielle

Image IA

Mémoire sur le

Image-to-Image Translation with Conditional Adversarial Networks

LEMBREZ Gabin
REY Thomas
Promotion 2021/2022

-25 novembre 2021-

Table des matières

1 Introduction

Le problème de translation image-to-image. Ce mémoire étudie l'article "*Image-to-image translation with Conditionnal Adversial Networks*". L'objectif d'un système effectuant une translation *Image-to-image* est d'associer une image (*output*) à l'image qu'il reçoit en *input*. Il s'agit de l'analogue de la traduction de texte en traitement d'image. C'est un problème fréquemment rencontré (génération de cartes de chaleur à partir d'images, détection de bords, ...), toutefois les méthodes automatiques usuelles ont une limite commune : un défaut de généralisation. En effet, même si la description de la tâche (prédire des pixels à partir de pixels) est identique pour tous les problèmes, la solution usuelle consiste en la construction d'un système très spécialisé pour une tâche particulière.

Objectifs de l'article. Dans le cadre décrit précédemment, le travail à effectuer est de construire un cadre de travail permettant d'aborder les problèmes de translation image-to-image. Le principal atout de la méthode proposée est sa capacité de généralisation.

La solution des CAN. Comme la plupart des problèmes de traitement d'images, ce problème présente le cadre d'application idéal des CNN (convolutional neural networks). Toutefois, la définition manuelle d'une *loss function* à minimiser demande toujours de l'expertise et impose un degré de spécialisation du réseau. Par ailleurs, la plupart des loss usuelles réagissent mal sur ce type de problème. Par exemple, minimiser la loss $\mathcal{L}_2$ amène le réseau à créer des images "floues" qui correspond à un minimum local de l'erreur en norme euclidienne mais qui est loin d'un rendu réaliste.
Pour pallier ce problème, l'article étudie la solution des GANs (Generative adversarial networks) : deux réseaux sont entraînés en parallèle, un prédicteur et un générateur. Ainsi, au lieu de devoir minimiser une *loss*, le générateur cherche à tromper le prédicteur. Par ailleurs, les GANs utilisés dans l'article ont la particularités d'être conditionnels aux inputs : en effet dans le cas d'une translation image-to-image, le générateur ne crée pas l'output de toute pièce, il se base sur l'input.

2 Méthode et architecture

2.1 Méthode

Ce paragraphe détaille la méthode utilisée pour résoudre le problème de translation image-to-image en utilisant les CANs. Soient G le *generator* et D le *discriminator*, les deux fonctions adverses. Les deux réseaux intéragissent avec des images de deux types (input et output). D apprend à classifier les images selon si elles sont réelles (issues des output d'entraînement) ou créées par G. En parallèle, G apprend à générer des images minimisant la loss définie par D.

$$\text{GAN} : G : \underbrace{z}_{\substack{\text{vecteur aléatoire} \\ \text{(bruit)}}} \longrightarrow \underbrace{y}_{\text{image en output}} \tag{1}$$

$$\text{GAN conditionnel} : G : \{ \underbrace{x}_{\text{image en input}} , \underbrace{z}_{\substack{\text{vecteur aléatoire} \\ \text{(bruit)}}} \} \longrightarrow \underbrace{y}_{\text{image en output}} \tag{2}$$

Les générateurs fonctionnent en propageant un vecteur aléatoire à travers un réseau. C'est une tâche analogue à de la "déconvolution" : le réseau reçoit un vecteur condensé de features et augmente graduellement sa dimension en le propageant à travers ses couches. La seule différence est que le vecteur initial n'est pas issu d'une étape préliminaire de convolution à partir d'une image réelle mais est aléatoire.

Dans le cas d'un GAN (1) l'opération peut se limiter à des successions de *up-convolution* et *up-sampling*. Toutefois, pour le problème décrit en introduction, une telle approche est insuffisante, puisque la translation image-to-image doit se baser sur une image en input.

C'est pourquoi un GAN conditionnel (2) est utilisé : en plus du vecteur de bruit, G reçoit l'image à observer sur laquelle baser la génération. Dans les faits, il est possible de créer une image en propageant l'image d'input à travers un réseau de convolution. Toutefois, ce résultat est déterministe, c'est-à-dire que l'image output sera toujours la même pour une image input donnée. Par ailleurs, un bruit gaussien appliqué sur l'image ne suffit pas : le générateur finit par apprendre à ignorer le bruit. La stratégie suivie dans l'article étudié est d'appliquer le bruit au moyen de *dropouts* aléatoires dans le réseau, c'est-à-dire que certains neurones pris au hasard sont manuellement mis à zéro avant une génération.

Formulation du problème. Un GAN peut être vu comme un problème d'optimisation de la fonction G au sens du critère défini par la fonction D. Les deux réseaux (G, D) évoluent en parallèle. Il convient de définir une fonction $\mathcal{L}(G, D)$ qui mesure la vraisemblance des images produites par G au sens défini par D. De cette façon, G cherche à minimiser la fonction $\mathcal{L}$ tandis que D cherche à la maximiser. Le réseau G^* le plus adapté pour le problème de translation image-to-image est alors :

$$G^* = \arg\min_G \max_D \mathcal{L}(G, D)$$

La fonction $\mathcal{L}$ est définie de sorte à valoir 0 lorsque G trompe parfaitement D (en moyenne, D ne fait pas la différence entre y et $G(x, z)$) et augmente dès que le nombre moyen de discriminations réussies augmente. Deux approches sont possibles : l'équation (3) correspond à l'utilisation d'un GAN entièrement conditionné aux données d'entrées alors que l'équation (4) correspond au cas où seul G a accès aux données d'entrées et D ne les observe jamais.

$$\mathcal{L}^1(G, D) = \underbrace{\mathbb{E}_{xy}(\log(D(x, y))}_{\substack{\text{valeur moyenne du log de la prédiction} \\ = 0 \text{ si } D \text{ se trompe systématiquement}}} + \underbrace{\mathbb{E}_{xz}(\log(1 - \underbrace{D(x, G(x, z))}_{\substack{\in [0,1] \text{ probabilité d'observer} \\ \text{une vraie image (=0 si } G \text{ échoue)}}}))}_{\substack{\text{valeur moyenne de 1-P(prédiction)} \\ = 0 \text{ si } D \text{ se trompe systématiquement}}} \tag{3}$$

$$\mathcal{L}^2(G, C) = \mathbb{E}_y(\log(D(y)) + \mathbb{E}_{xz}(\log(1 - D(G(x, z)))) \tag{4}$$

$$\mathcal{L}_{L_1}(G) = \mathbb{E}_{xyz}(||y - G(x, z)||_1) \tag{5}$$

Dans la situation (4), le résultat produit par G n'est comparé qu'à y qui correspond à l'image réelle issue du set d'entraînement. Ainsi, G est obligé de fournir un résultat proche de la réalité. Avec la même idée, il est possible de comparer directement les productions de G aux données réelles en utilisant une métrique telle que la norme euclidienne ou L_1 (5). Une telle approche fonctionne mal seule. En effet, la meilleure option pour G est de proposer un résultat flou qui sera toujours globalement proche de la réalité mais jamais performant. La solution implémentée dans l'article étudié correspond à la fonction :

$$\mathcal{L}(G, D) = \underbrace{\mathcal{L}^1(G, D)}_{\substack{\text{Performances de } G \\ \text{quantifiées par } D}} + \lambda \underbrace{\mathcal{L}_{L_1}}_{\substack{\text{Performances de } G \\ \text{comparé à la réalité}}} \tag{6}$$

Cette solution offre un bon compromis pour limiter les artefacts non réalistes qui peuvent survenir avec l'utilisation de GAN conditionnel seul tout en évitant le rendu flou obtenu avec une métrique trop rudimentaire. Visuellement, l'hyperparamètre λ correspond à l'intensité du "lissage" (le concept a des définitions variables selon le type de problème, mais l'idée reste la même).

2.2 Architecture

Les deux réseaux D et G sont construits sur une structure classique de CNN. En effet, dans les deux cas, il s'agit de successions de modules comprenant *Convolution-BatchNorm-ReLu*.

Architecture de G. Plusieurs constatations sont nécessaires pour le choix de l'architecture de G. Ce réseau observe une image en haute résolution (input) et doit fournir une image de haute résolution (output). Pour la translation de l'un vers l'autre, le réseau doit pouvoir représenter les informations importantes sous forme condensée, telles que les contours principaux. Il est donc important que l'architecture choisie permette d'avoir des input et output de taille importante en passant par un stade où l'information est condensée. Cette contrainte motive le choix d'une architecture basée sur le modèle *U-Net* (1), c'est-à-dire un réseau en entonnoirs au format "encodeur-décodeur".

Par ailleurs, à chaque goulot dans la structure en entonnoir, il y a un risque de perte d'information. Pour limiter ce défaut, le réseau permet des *skips*, c'est-à-dire une connexion entre la couche i et la couche $n-i$. Ainsi, le rôle de la couche $n-i$ de décodage devient de construire un vecteur à partir des informations condensées dans la couche $n-i-1$ et des informations contenues dans la couche i de codage.

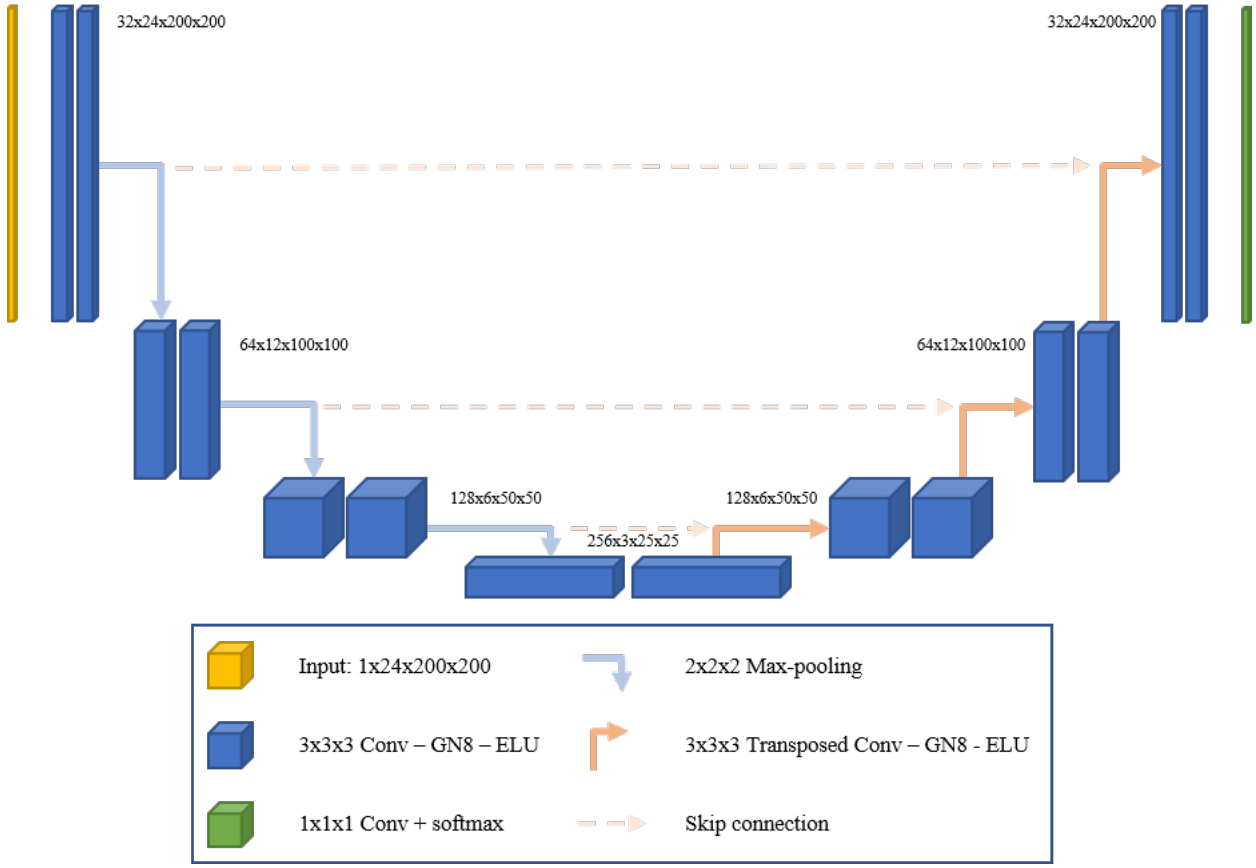

FIGURE 1 – L'architecture U-net avec skips telle qu'utilisée sur un problème de segmentation

Architecture de D. La décomposition décrite par l'équation (6) utilise la norme L_1 pour décrire les comportements de basses fréquences (i.e. qui varient peu). De cette façon, seule la modélisation des informations de hautes fréquences est importante pour D. De fait, l'architecture retenue est un format *PatchGan* (2) : D observe l'image par *patches*, c'est-à-dire en ne considérant que des échantillons de taille restreinte de l'image. Cette opération permet de réaliser un grossissement au sein de l'image, d'où son intérêt dans l'étude des phénomènes de hautes fréquences. Cette modélisation correspond au formalisme des champs aléatoires Markovien, sous l'hypothèse que deux pixels sont indépendants s'ils sont séparés d'une distance supérieure à la taille d'un patch. Ce modèle garantit des résultats de convergence et donne un ordre de grandeur pour les tailles de patches.

3 Données

Le modèle est entraîné à partir de couples d'images de la forme $\{A, B\}$. Les images A et B doivent avoir la même taille de $256 \times 256\,px$ ou $512 \times 512\,px$, il faut donc normaliser les images si nécessaire. Il est important que les deux images du couple décrivent la même structure : par exemple A correspond à des images de bâtiments et B correspond à une carte d'étiquettes associée décrivant le même bâtiment (*labels2facade*). A et B sont également séparés en deux échantillons pour l'entraînement (*train*) et l'évaluation (*test*).

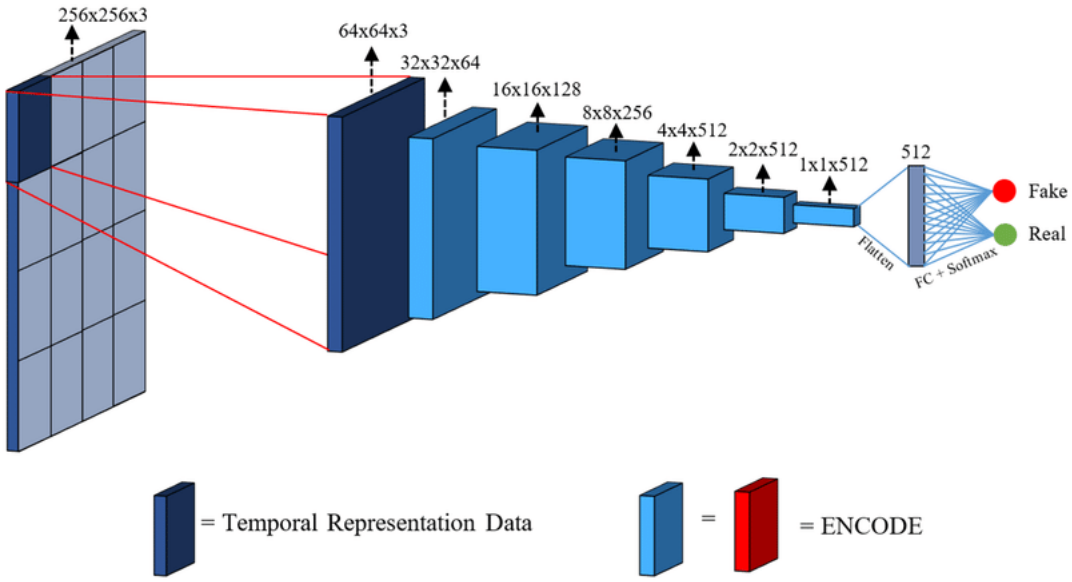

FIGURE 2 – L'architecture Patch-GAN

4 Training

Pour entraîner le modèle, il est nécessaire d'entraîner D et G en parallèle (3). La méthode retenue pour effectuer cette tâche est d'effectuer successivement un pas de descente de gradient sur G puis un pas de descente de gradient sur D. La méthode de descente de gradient utilisée est la SGD : *stochastic gradient descent*. La SGD est une alternative entre la descente de gradient standard (le gradient de la loss est calculé pour chaque input) et la descente de gradient par Batch (la connaissance du gradient relatif à chaque donnée est nécessaire pour faire un pas). La méthode consiste à répéter :

1. Prendre un échantillon de données d'entrée
2. Les propager à travers le réseau
3. Calculer le gradient de la Loss relatif à cet échantillon
4. Mettre à jour les poids du réseau à partir de ce gradient

Par ailleurs, c'est l'*Adam optimizer* qui est utilisé plutôt que la simple SGD : la *learning rate* n'est plus fixe. Il y a une learning rate par poids du réseau et le paramètre s'ajuste au fur et à mesure de l'entraînement.

5 Inference Time

Le réseau fonctionne de la même façon pendant l'entraînement et en utilisation. En effet, pour éviter un comportement déterministe du système, les dropouts sont appliqués aussi bien pendant l'entraînement que pendant l'évaluation. Par ailleurs, le procédé de *batch-normalization* est utilisé : les input sont normalisés (translatés et changés d'échelle) suivant une méthode fixée. L'efficacité de ce procédé est observable dans la pratique mais peu de résultats théoriques confirment son efficacité. La méthode employée dans l'article a permis une amélioration des performances. Usuellement, la *batch-normalization* est effectuée à partir de statistiques collectées sur les données d'entraînement ; toutefois, la méthode choisie ici consiste en une évolution continue de la batch-normalization en prenant également en considération les données statistiques récoltées à partir de la phase d'évaluation.

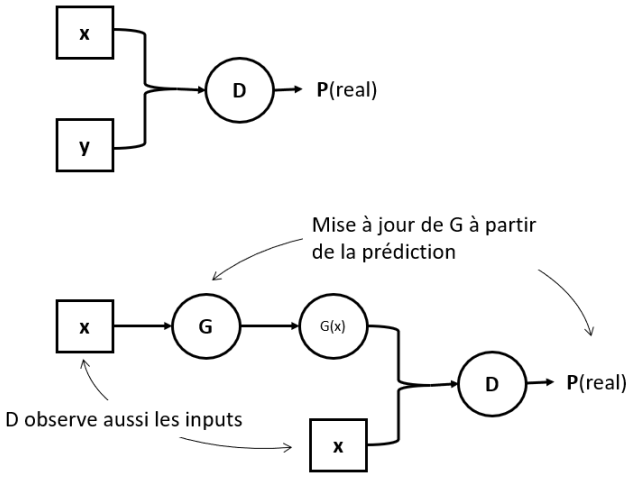

FIGURE 3 – Protocole d'entraînement de D et G en parallèle sur des paires $\{x, y\}$ d'images

6 Expériences

Afin de voir la portée de la méthode décrite dans l'article, les auteurs ont entraîné le CNN pour différentes tâches.

- Semantic labels to photo [1]
- Architectural labels to photo [2]
- Map to aerial photo [3]
- Aerial photo to map [4]
- Black and white to color photo [5]
- Edges to photo [6]
- Sketch to photo [7]
- Day to night photo [8]

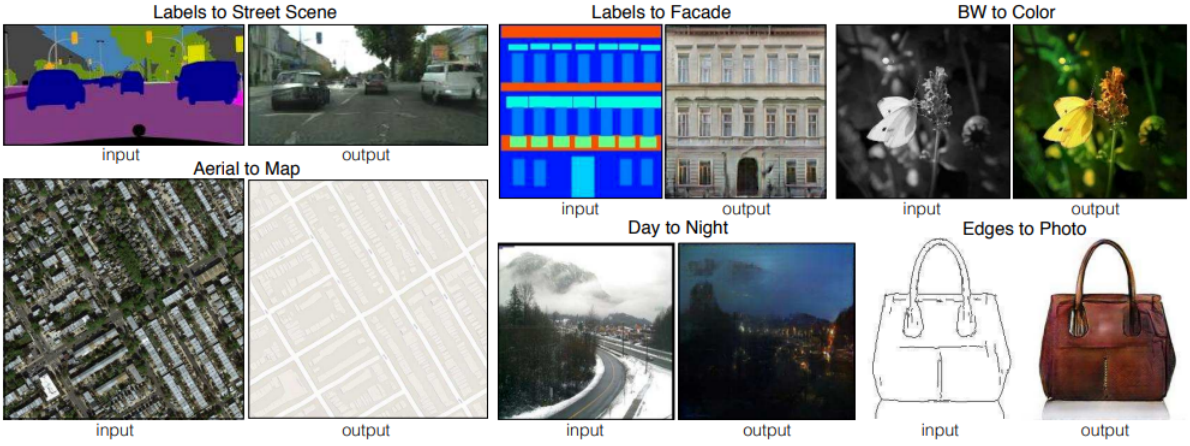

FIGURE 4 – Différentes tâches effectuées par le réseau

Dans tous les cas les images d'input et d'output sont des images RGB.

Mesure de la performance Le principal problème réside dans l'évaluation de la performance des différents réseaux. En effet les techniques usuelles mesurant l'erreur moyenne par pixel ne permettent pas de considérer les structures hautes fréquences de l'image. Ainsi deux techniques différentes ont été utilisées.

Amazon Mechanical Turk percepetual studies (AMT) Cette technique est utilisée pour les tâches de colorisation d'image et de génération d'image, soit les tâches [3], [4], [5] et [8]. Simplement, des hommes ont été payés pour évaluer les différentes images. On leur a présenté des séries de paires d'images mettant en parallèle une image réelle et une générée, présentée l'une après l'autre pendant une seconde. Ensuite ils devaient déduire laquelle était la générée. Un feedback était donné sur les dix premières images. Puis les 40 d'après n'en n'avaient pas. Il y a eu 50 essais sur chaque algorithme, qui ont été tous testés un par un.

FCN-score. Cette technique est utilisée afin de donner une valeur numérique à la performance du CNN. Elle consiste à utiliser un classifieur pré-entraîné avec les images réelles. Ainsi si les images générées sont réalistes, alors il pourra bien les classer. Le classifieur utilisé par les auteurs est un FCN-8s, *fully convolutional network with an output stride of 8* utilisé pour des tâches de segmentation sémantique.

Architectural labels to photo Les auteurs ont utilisé cette tâche afin de mesurer les effets des différentes loss. Sur la figure 5, on peut voir la performance du réseau pour différentes loss sur cette tâche. Afin d'évaluer visuellement la performance du réseau pour différentes loss, les auteurs ont utilisé le réseau avec soit $\mathcal{L}_{L_1}(G)$ ou $\mathcal{L}^1(G, D)$ ou $\mathcal{L}$. On peut voir le résultat sur la figure 6.

Loss	Per-pixel acc.	Per-class acc.
L1	0.42	0.15
GAN	0.22	0.05
cGAN	0.57	0.22
L1+GAN	0.64	0.20
L1+cGAN	**0.66**	**0.23**
Ground truth	0.80	0.26

FIGURE 5 – FCN score du réseau pour différentes loss pour la tâche [2]

FIGURE 6 – Qualité du résultat du réseau pour différentes loss, sur la tâche [2]

Comme il a été mentionné en introduction, l'utilisation de la loss $\mathcal{L}_{L_1}(G)$ n'est pas optimal, le réseau minimise l'erreur globale ce qui donne une image floue. L'utilisation de la loss $\mathcal{L}^1(G, D)$ produit un résultat bien plus net, cependant elle peut introduire des artefacts sur certaines applications. Ainsi faire une composition des deux à l'aide d'un hyperparamètre λ permet de réduire l'apparition de ces artefacts. On peut voir un exemple de ces artefacts sur la figure 7.

FIGURE 7 – Artefact sur la tâche [2]

Black and white to color photo De la même façon que la norme $\mathcal{L}_{L_1}(G)$ crée une image floue, dans les tâches de colorisation elle va créer une image grise. Cependant, de la même façon que précédemment l'utilisation de cGAN va permettre de savoir qu'associer globalement du gris aux différents pixels ne donne pas une image réaliste. Cela va alors contraindre le réseau à associer la bonne couleur au pixel.

Performance du U-net par rapport à un encodeur-décodeur classique Afin d'évaluer l'effet des *skips* entre les couches i et $n - i$. Les auteurs ont pris soit un *generator* sous la forme d'un U-net, soit un encodeur-décodeur classique (un U-net mais en enlevant les *skips*). Ils ont obtenu le résultat présenté dans la figure 8.

FIGURE 8 – Effet des *skips* sur le résultat du réseau

D'après cette image il est évident que les *skips* sont indispensables pour que le réseau ait un résultat convenable. De plus cela montre encore l'efficacité de l'utilisation de la loss $\mathcal{L}$ par rapport à la loss $\mathcal{L}_{L_1}$

Performance du *PatchGAN* selon la taille du *patch* Afin de quantifier la performance du *discriminator*, différentes tailles de *patch* ont été utilisées. Selon la taille des *patch* des artefacts peuvent apparaître. Ainsi augmenter la taille des *patch* paraît être une méthode pour supprimer les artefacts, comme on peut le voir sur la figure 9. Ainsi au

FIGURE 9 – Influence de la taille des *patch* sur la qualité de l'image

vu de la figure 9, deux choses apparaissent. Premièrement, l'utilisation de *PatchGAN* permet d'avoir une meilleure précision en couleur que la norme L1. En effet entre l'image issue de la loss $\mathcal{L}_{L_1}$, floue et grisâtre, et l'image issue du 1x1 *PatchGAN* le contraste est important. Deuxièmement, augmenter la taille du *patch* permet d'obtenir une meilleure résolution. Cependant cela est contredit par le FCN-score ; en effet le 70x70 *PatchGAN*, donne une meilleure image que le 286x286 *PatchGAN*, comme on peut le voir sur la figure 10. Cette différence peut être due au nombre important de paramètres à entraîner dans le 286x286 *PatchGAN* par rapport au 70x70 *PatchGAN*.

Discriminator receptive field	Per-pixel acc.	Per-class acc.
1×1	0.39	0.15
16×16	0.65	0.21
70×70	**0.66**	**0.23**
286×286	0.42	0.16

FIGURE 10 – Influence de la taille des *patch* sur le FCN-score

De plus les auteurs ont pu prouver qu'un *PatchGAN* peut être appliqué à des images de plus grande dimension que celle sur lesquelles il a été entraîné.

Amazon Mechanical Turk perceptual result Enfin, afin d'avoir une estimation de la performance de la tâche [3] ou de l'inverse de celle-ci [4], les auteurs ont confronté leurs résultats avec les AMT. Ainsi environ 19% des personnes ont été trompées en considérant qu'une image générée était réelle pour la tâche [3]. En prenant la même tâche mais avec comme loss $\mathcal{L}_{L_1}$ moins d'1% d'elles ont été trompées, cela montre encore une fois l'importance du $cGAN$ pour avoir une image réaliste. Cependant pour la tâche inverse (de photo à map), le nombre de personnes trompés diminue fortement, passant de 19% à 6%.

6.1 Notre expérience

Cette section présente les résultats obtenus dans une tentative d'entraînement d'un modèle pix2pix sur un problème de translation image-to-image. La méthode employée est décrite en détaille par Christopher Hesse dans https://github.com/affinelayer/pix2pix-tensorflow#creating-your-own-dataset. L'avantage de la méthode proposée est que le preprocessing est très simple. En effet, une tâche plus compliqué aurait nécessité des mois de travail. Pour autant, l'expérience proposée ici est exposée pour montrer la faisabilité de la démarche, et non pour obtenir des résultats de la qualité de ceux obtenus dans l'article.

Le contexte. Christopher Hesse propose une méthode permettant de réaliser de l'inpainting : prendre une photo, en supprimer un rectangle central et laisser au générateur le soin de reconstituer la partie manquante. Le protocole suivi est le suivant :

1. Télécharger des photos pour servir de données d'entraînement et de test. Le plugin Image_Assistant_Batch_Image_downlo de firefox permet d'extraire facilement les images issues d'une recherche google.
2. Convertir toutes les images au format jpg (en s'assurant qu'elles ont toutes le même nombre de canaux, notamment pour les images en niveaux de gris) et les renommer pour avoir le même cas d'utilisation que les datasets disponibles sur le notebook.
3. Dupliquer chaque image et construire une image d'entraînement en accolant l'image et sa copie. Sur la copie, mettre tous les pixels sur un rectangle central (1/9 de l'image) en blanc
4. Suivre la même procédure que pour tous les datasets présentés dans le notebook.

La tâche. Christopher Hesse utilisait des photos de chats. Nous avons testé des données moins figuratives : nous avons pris environ 200 tableaux de Jackson Pollock pour leur appliquer de l'inpainting. Le code python utilisé pour faire l'étape de preprocessing est disponible avec le document. Sous réserve de changer les noms des paths, la méthode utilisée peut s'appliquer à tout problème d'inpainting.

Commentaires. La prise en main du modèle pix2pix pour un problème envisagé dans sa globalité a permis de mettre en évidence la simplicité d'utilisation du modèle. Par ailleurs, aucune adaptation de l'architecture n'a été nécessaire : en effet, seules les données d'input ont nécessité un travail préliminaire. En ce sens, l'objectif de proposer un framework général pour les problèmes de translation image-to-image est atteint. La figure 11 présente quelques résultats obtenus.

Limitations. De nombreux points peuvent être améliorés dans cette expérience.

- Le dataset est constitué d'environ 200 images. En comparaison, le plus petit dataset proposé par les développeurs contient 400 échantillons. Augmenter la taille du dataset est donc une première voie d'amélioration.
- Plusieurs images au sein du dataset nécessiterait un travail préalable plus approfondi (suppression d'arrière plan, élimination d'éléments parasites, correction des effets de perspective) mais qui demanderait un temps trop important
- Les peintures de Jackson Pollock sont parfois très chargées. Or, la compression en $256 \times 256px$ est violente pour ce genre d'usage. Utiliser des images $512 \times 512px$ permettrait de palier le défaut de qualité. Toutefois, il faudrait revoir l'architecture du réseau (rajouter une couche de down-sample et une de up-sample) et cela augmenterait encore le temps d'entraînement.

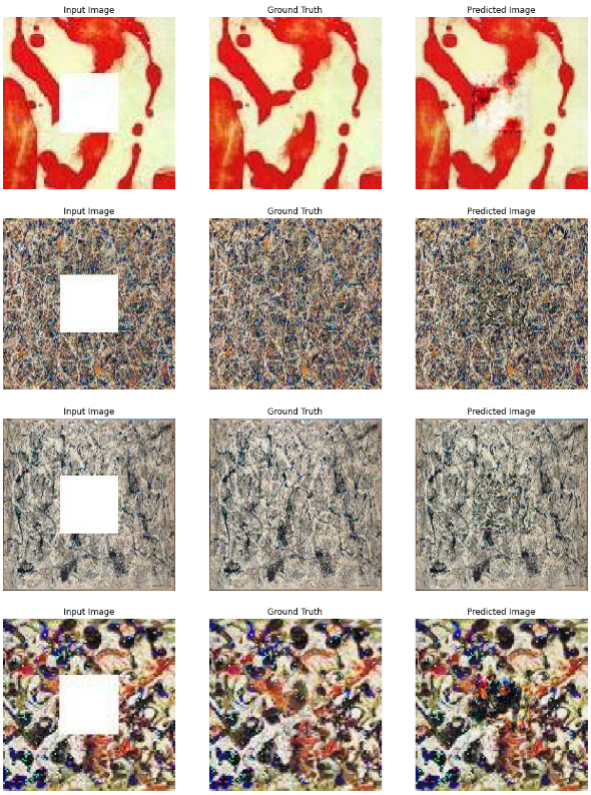

FIGURE 11 – Résultats obtenus après 5h d'entraînement

7 Limitations

Cette diminution importante vient du fait que les *cGAN* sont très bons lorsque les images en sortie sont complexes. Or des cartes ou de la segmentation sémantique sont des images très simples à représenter ; ainsi les *cGAN* n'ont pas de bonnes performances sur ce genre de tâche. De plus sur la segmentation sémantique, la loss $\mathcal{L}_{L_1}$ est meilleure que la loss $\mathcal{L}^1$. Enfin sur une tâche de colorisation, la technique présentée dans l'article a une bonne performance mais face à une technique plus spécialisée sur cette tâche, l'utilisation de *cGAN* est moins adaptée. De plus comme nous en avons déjà parlé les *cGAN* peuvent créer des artefacts dans l'image générée.

8 Conclusion

La force de cette technique de translation d'image à image est qu'elle s'applique à une variété d'exemples, seulement quelques-uns sont présentés dans l'article, mais la communauté a utilisé la technique de *pix2pix* dans un cadre bien plus varié (*edges2cats*,*Background removal*,...). Enfin cette technique offre de bonnes performances globales, elle excelle dans les tâches où l'image de sortie est complexe. Cependant différentes techniques plus adaptées à un certain nombre de tâches parviendront à proposer de meilleures performances.

9 Sources

https://phillipi.github.io/pix2pix/