Rapport sur le Projet CycleGan

Paulin Brissonneau Thomas Kebaili Ilyas Moutawwakil
Valentin Laurent Lilian Lecomte

XX mai 2020

Table des matières

Chapitre 1

Le Multiperceptron

1.1 Le perceptron simple

Le principe du perceptron est de bla bla bla

Chapitre 2

Les réseaux à convolutions

2.1 Présentation générale

2.1.1 Introduction

L'architecture d'un multiperceptron se prête bien à des applications simplistes comme de la reconnaissance de chiffre. Cependant, dès que la complexité des données augmente ne serait-ce que légèrement, le MLP devient inneffi cace en un temps raisonnable pour pouvoir reconnaitre des objets . Heureusement, il existe une structure bien plus efficace qui est parfaitement adaptée à la reconnaissance d'image : le CNN (convolutional neural network). Dans cette partie, nous allons nous efforcer d'expliquer le fonctionnement de celui-ci ainsi que ses avantages.

2.1.2 Structure d'un CNN

La structure d'un CNN se rapproche de celle d'un multiperceptron dans la mesure où celui-ci est aussi organisé sous forme de couches. Cependant, certaines couches, nommées couche de convolution, ont un fonctionnement radicalement différent de celui d'une couche dense.

Voici l'architecture d'un CNN :
insérer image architecture CNN

Comme on peut le voir, l'image en entrée est d'abord traitée par une succession de couches de convolutions suivies d'une couche de pooling. Ensuite ce processus se répéte un certain nombre de fois et finalement le résultat passe par une couche dense(plus rarement plusieurs) afin de donner la bonne classification.

2.1.3 Principe général

Le principe directeur se cachant derrière les couches de convolution est assez intuitif : on dispose d'un noyau(pattern) capable de reconnaître un motif en particulier. Il suffit alors de balayer l'image pixel par pixel(notion expliquée plus en détails ultérieurement) pour savoir où se trouvent précisément certains motifs sur l'image. En répétant ce procédé avec un grand nombre de noyaux

différents, nous sommes dans la possibilité de pouvoir identifier la présence ou non de certains motifs caractéristiques de l'objet à reconnaître.
Nous disposons ainsi d'une "carte" des différents motifs qu'il nous est possible de réduire en taille(étape de pooling) pour réduire le nombre de calculs effectués. Le processus se répéte un certains nombre de fois jusqu'à ce que l'on considère que les données soient suffisamment bien réduites pour qu'un MLP puisse s'en charger.

2.2 Formalisation d'un CNN

Une fois le principe directeur mis en évidence, nous pouvons nous atteler à la formalisation du CNN.

2.2.1 Le produit de convolution : un outil indispensable

Le coeur de l'efficacité d'un CNN repose sur le produit de convolution. On assimile l'image à une matrice notée I(nous la supposons pour l'instant en niveau de gris, de telle sorte que la matrice est en 2D mais la généralisation en 3D se fait aisément). De même, le noyau sera noté sous la forme d'une matrice K. L'opération de convolution s'écrit alors :

$$\forall (x,y) \in [|0, W_I|] \times [|0, H_I|],\ (N \otimes I)_{x,y} = \sum_{i=0}^{W_K} \sum_{j=0}^{H_K} K_{i,j} \times I_{x-i,y-j}$$

Où l'on a posé W_I, H_I la taille de l'image et W_K, H_K la taille du noyau.

Voici quelques exemples de l'effet du produit de convolution avec plusieurs noyaux :

insérer image chien de prairie

Ainsi, cette opération nous permet de mettre en évidence des motifs élémentaires en activant la case correspondante si le motif est présent au niveau de celle-ci.

2.2.2 Le balayage de l'image : une histoire de Padding et de Stride

Pour pouvoir évaluer toute l'image, il va falloir la balayer avec le noyau. De manière assez logique, on commence par le placer sur le coin supérieur gauche de l'image, puis on affectue un produit de convolution. Ensuite, il suffit de faire glisser latéralement le noyau d'un pixel et de répéter l'opération. Une fois au bout de la ligne, on descend d'un pixel et on repart à gauche.
Voici un exemple de balayage d'un noyau sur une image :

insérer image balayage image

On remarque alors que le résultat à une dimension plus petite que l'image d'origine : on a ici effectué ce que l'on appelle un **simple padding**. Cependant, on

pourrait souhaiter que le résultat ait la même dimension : c'est du **same padding**. Pour cela, nous pouvons rajouter des gardes autour de l'image d'origine : on rajoute des zéros autour puis on effectue l'opération de convolution comme présenté sur cette image :

insérer image same padding

Il existe encore un paramètre permettant de personnaliser l'opération : le **stride**. Celui correspond au décalage à utiliser pour le noyau :

insérer image stride

L'avantage d'avoir un stride plus grand que 1 est aussi son désavantage : en effet il va permettre d'avoir un résultat en sortie de plus petite dimension mais en contrepartie, il y a une perte d'information.

2.2.3 Généralisation en 3D

Les images en couleurs peuvent être représentées par une matrice 3D de profondeur 3. Pour pouvoir gérer ce cas, nous prenons des noyaux de de la même profondeur que l'image, puis nous appliquons séparément le produit de convolution sur les profondeurs correspondantes, le résulat final n'étant que la somme des résultats sur chaque profondeur. Puisqu'un schéma vaut bien plus qu'un texte :

insérer image 3D

Chapitre 3

Les GAN (Réseaux Adverses Génératifs)

3.1 Principe général des GAN

Le principe général des GAN repose sur l'utilisation de deux réseaux, ayant des objectifs contraires, on dit qu'ils sont **adversaires**. Le premier réseau transforme du bruit en image, c'est le **générateur** (G). Le deuxième réseau prend en entrées des images et les classe en deux catégories, en leur associant leur probabilité d'être issues de la base de donnée : c'est donc un classifieur binaire, il est appelé **discriminateur** (D). Le plus souvent, le discriminateur sera alimenté par des images de deux sortes : celles provenant de la base de donnée (images réelles), et celles générée par le générateur, on rôle sera donc de dire si une image est réelle ou générée. Il s'agit ensuite d'entraîner G afin qu'il maximise la probabilité que D fasse une erreur, et D la justesse de sa classification

L'architecture des GAN a été introduite pour la première fois par Ian Goodfellow [ref] en 2014. Cet article innovant montrait déjà un gain de performance pour la génération d'images suivant une base de donnée. Mais l'atout majeur des GAN sont leur adaptabilité à tous types de données.

3.2 Le DCGAN (Deep Convolutionnal Adversarial Network)

Le DCGAN est la première architecture de GAN qui a été proposée [ref GoodFellow]. Le générateur et le discriminateur sont tous les deux des **réseaux à convolutions** [ref].

Cette architecture est caractérisée par les fonctions de coût de G et D. La fonction de coût de G à minimiser est la suivante :

$$\mathbb{E}_{x \in dataset}(\log(D(x))) + \mathbb{E}_z(\log(1 - D(G(z))) \tag{3.1}$$

Pareillement, on donne comme fonction de coût pour D l'opposé de celle de G. L'architecture du DCGAN correspond alors à un jeu minmax. La théorie des équilibres de Nash donnent un unique état stable. Il correspond à un coût égal à $-\log 4$ pour G et $\log 4$ pour D. Cette situation représente un discriminateur

forcé d'associer une probabilité de 0,5 pour chaque image donnée en entrée, le générateur étant devenu trop fort.

Une variation intéressante sur le DCGAN est de poser $\mathbb{E}_{x \in dataset}(\log(D(x)))+ \mathbb{E}_z(\log(D(G(z)))$ comme fonction de coût en début d'apprentissage. L'intérêt de cette modification résulte d'un problème : le discrimateur a tendance à facilement distinguer les images générées par G de celles de la base de donnée en début d'apprentissage. Dans ce cas, le terme $\log(1 - D(G(z))$ sature vers 0. Le remplacer par $\log(1 - D(G(z))$ résoud ce problème.

3.3 Le W-GAN (GAN de Wasserstein)

blablabla

3.4 Étude de la convergence des GAN

De par leur caractère adversaires, les GAN requièrent un équilibre fin entre la générateur et le discriminateur, ils sont donc par nature **instables**. L'étude de la convergence des GAN est un domaine encore très actif de la recherche. Nous allons discuter de deux phénomènes très communs qui peuvent gêner ou ruiner l'apprentissage des GAN : l'**effondrement des modes** (*mode collapse*), et la **non-convergence** due à la perte d'équilibre du système.

3.4.1 L'effondrement des modes

L'effondrement des modes survient quand le réseau générateur ne génère pas des images conforment à l'ensemble de la distribution des images réelles, mais seulement à une petite partie. L'effondrement des modes est très visible lorsque la distribution des images réelles forme des zones bien séparées, c'est à dire quand celle-ci comporte des classes bien définies. La manifestation de ce phénomène se traduit par des images générées qui se ressemblent toutes. La figure [ref figure] montre un exemple du phénomène sur la base de données MNIST et CelebA.

FIGURE 3.1 – Exemples d'effondrement des modes. À gauche sur la banque de chiffres MNIST, à droite sur la banque de visages CelebA.

Pour mieux comprendre le phénomène, il est intéressant de regarder la distribution des images de MNIST dans son ensemble, cela est possible grâce à des algorithmes de réduction de dimension. Attention, la réduction de dimension se fait dans l'espace des pixels, et non pas dans un espace sémantique, la visualisation ne permet donc pas de séparer efficacement les différentes classes,

elle permet seulement un aperçu de la distribution dans l'espace sémantique. La figure [ref figure] présente une visualisation de MNIST par transformation t-sne [ref tsne].

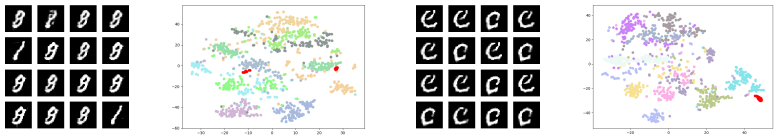

FIGURE 3.2 – legende

L'ensemble de points rouges correspond à un ensemble d'images générées par le réseau générateur lors de l'effondrement des modes. Sur les données MNIST, on observe différents groupes de points (des *clusters*), ce sont les **modes** inhérents à la base de donnée MNIST : les chiffres de 1 à 9. Ce qu'il est intéressant de noter, c'est que les points générés sont rassemblés autour de un ou plusieurs pôles denses très localisés, qui ne sont pas répartis dans tout l'espace. Cela traduit l'effondrement des modes : les images générées ne couvrent qu'une petit partie de la distribution de la base de donnée d'entraînement.

Il n'y a pas de solution simple, directe et universelle pour lutter contre l'effondrement des modes, mais quelques solutions ont été proposées :

- La pénalisation de la similarité des images en sortie de générateur *mini-batch discrimination*. Cela consiste à ajouter un terme à la fonction de coût pour traduire la similarité (il peut s'agir de calculer une similaire pixel à pixel, ou d'estimer la similarité sémantique avec un autre réseau de neurones).
- Le *one-side label smoothing*. Cela consiste à changer l'objectif du discriminateur : son objectif ne sera plus de discriminer les fausses images avec une probabilité de 1, mais une probabilité plus faible, par exemple 0.9. Cela permet d'éviter la sur-confiance, et permet de laisser le générateur explorer tous l'espace des images réelles.
- Certaines architectures sont plus résistantes que d'autres à l'effondrement des modes. Par exemple, les GAN de Wasserstein ne présentent ce problème.

3.4.2 Perte de l'équilibre

Comme expliqué plus haut, l'apprentissage des GAN repose sur un équilibre fin entre le discriminateur et le générateur. Cet équilibre est parfois difficile à atteindre et est souvent instable, c'est pourquoi parfois le système s'effondre complètement. Cet effondrement vient souvent du fait que le discriminateur est devenu "trop fort" (sa fonction de perte tombe à zéro), et le générateur ne peut plus s'améliorer. Lorsque cela arrive, l'entraînement peut être arrêté : les images générées ne s'amélioreront plus. Un exemple de ce phénomène et illustré [ref figure], où l'on voit qu'à partir d'un cycle d'entraînement, la fonction de perte du discriminateur s'écroule et celle du générateur diverge.

Il existe des solutions pour lutter contre ce problème, et cela consiste souvent à rééquilibrer les puissances ou les vitesse de convergence des différents réseaux.

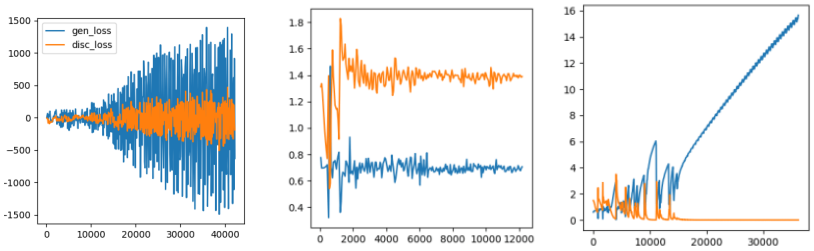

FIGURE 3.3 – legende

On peut par exemple diminuer la complexité du discriminateur, diminuer le taux d'apprentissage du discriminateur, ou mettre à jouer plus souvent le générateur que le discriminateur. Ajouter du bruit sur les images de la base de donnée permet aussi se renforcer la stabilité de l'apprentissage. Par ailleurs, on peut noter que les GAN de Wassertein sont plus stables que les DCGAN, mais ne sont pas totalement immunisés aux problèmes de convergence.

Chapitre 4

Le cycleGAN

4.1 Présentation de la problématique

[AJOUTER DES REFS]

Les cycleGAN sont des architectures de GAN qui permettent de répondre à une problématique bien spécifique : le **transfert de style non appairé**, que nous expliciterons.

Le transfert de style consiste à transformer des données d'un *style à un autre*. Le terme de *style* est à prendre au sens large et les données que l'on manipule peuvent être de natures diverses. Il peut s'agir par exemple de transformer des images de pommes en images d'orange, de transformer un paysage d'été en un paysage d'hiver, de transformer une musique classique en rock, ou encore de modifier l'expression les expressions faciales d'individus présents sur une image. Quelques exemples sont présentés sur la figure ? ?.

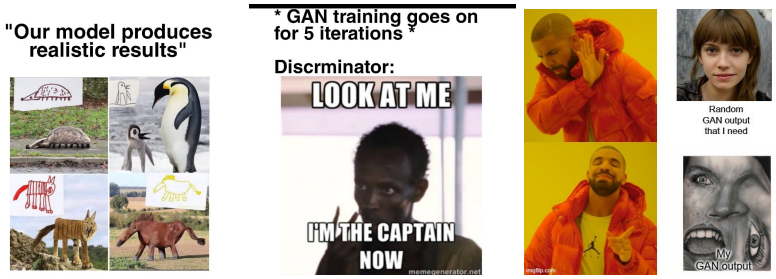

FIGURE 4.1 – legende

Le transfert de style peut s'effectuer entre plusieurs *classes de styles*, mais nous allons ici nous concentrer dans le cas binaire où l'on considère deux styles. La problématique est donc de transformer des images d'un style à l'autre, et ceci dans les deux sens.

Le transfert de style (à deux classes), repose sur deux banques de données, que l'on notera A et B. Suivant les données auxquelles nous avons accès, il existe deux cas différents :

— Dans le cas où nous connaissons un appairage entre les images de A et de B, le problème est un **transfert de style appairé**. Le but est donc

d'apprendre et de généraliser le transfert d'une donnée de A à une donnée de B à partir d'exemples de paires déjà existantes.
Par exemple, si A représente des bâtiments de jour, et B représente des bâtiments de nuit, il est possible de prendre la même photo de jour et de nuit. Ces deux photos constituent une paire dont chaque élément est d'un style différent..

— Dans le cas où chaque élément de A n'a pas de lien direct avec un élément de B en particulier, le problème est un **transfert de style non appairé**. Le but n'est plus d'apprendre et de généraliser le transfert d'une donnée de A à une donnée de B à partir d'exemples de paires déjà existantes, mais d'apprendre le transfert entre le style de A et le style de B, sans avoir d'exemple d'une telle transformation. Il faut donc *comprendre* à un niveau sémantique les style de A et B.
Par exemple, si vous voulez transformer une image de votre chien en image de chat, vous ne pouvez pas obtenir une banque d'image de chiens déguisés en chats. Vous devez donc travailler avec d'une part des images de chiens (A), d'autre part des images de chats (B), sans pouvoir former de paires entre A et B.
La différence entre ces deux cas est illustrée par la figure ? ?.

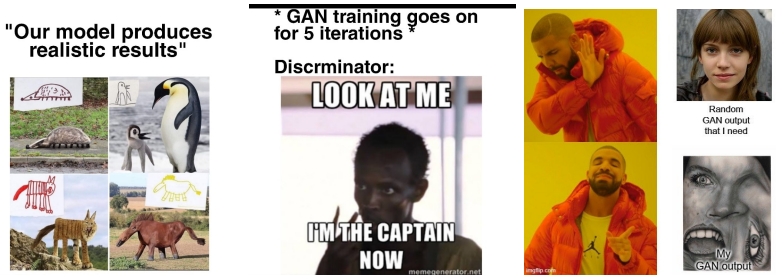

FIGURE 4.2 – legende

Ces deux types de transfert de style se traitent différemment. Pour le transfert de style appairé, une structure de GAN classique suffit puisque le discriminateur peut aisément comparer l'image générée avec l'image *idéale*. Ce problème, que nous ne développerons pas ici, est traité et manière efficace par différents algorithmes, dont **Pix2Pix**. Le transfert de style non appairé ne permet pas la comparaison à l'image-cible puisqu'il n'existe pas de paires. **Il faut donc utiliser d'autres architectures, comme par exemple le cycleGAN.**

4.2 Principe général du cycleGAN

En vertu des explications présentées au paragraphes précédent, le problème se présente ainsi : nous avons une banque de données structurées A, et une banque de données structurées B, de même nature, dont les styles sont différents. Dans la suite, nous nous placerons dans le cas où s'est données sont des images. Le but est de transformer les images de A pour leur donner le style des images de B, et inversement.

Le cycleGAN repose sur deux GAN, tête-bêche, l'un permettant de passer du style A au style B, l'autre du style B au style A. Plus précisément, il y a

deux générateurs, un générateur qui prend des images de la banque A et doit générer des images du style de B (noté G), l'autre qui prend des images de la banque B et doit générer des images du style de A (noté F). Il y a aussi deux discriminateurs, notés D_A et D_B, qui respectivement discriminent des images du style A et celles du style B. L'architecture est présenté par la figure ? ?.

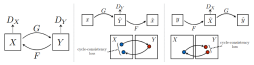

FIGURE 4.3 – legende (figure à changer)

Comme on l'a entrevu dans le paragraphe précédent, une difficulté est que les données ne sont pas appairées, la fonction de coût ne peut donc pas venir de la comparaison directe de l'image générée à l'image souhaitée. Pour pallier à ce manque, deux fonctions de coûts principales et indépendantes sont utilisées.
La première est celle d'un GAN classique : pour une transformation A (resp. B), le discriminateur D_B (resp. D_A) prédit si l'image est une image qui appartient réellement à la banque B (resp. A). Le coût associé au GAN ainsi défini est appelé *Adversarial Loss* ou *GAN Loss*.
Comme on peut s'y attendre, cela ne suffit pas. En effet, si l'on considère seulement ce coût, comment peut-on s'assurer que l'image obtenue a encore un lien avec l'image de départ ? Pour garantir cela, il faut s'assurer de pouvoir reconstruire l'image de départ après lui avoir fait subir la transformation A suivie de B. En d'autres termes, cela revient à ajouter des conditions sur les générateurs G et F telles que

$$\forall a \in A, F(G(a)) \approx aet \forall b \in B, G(F(b)) \approx b.$$

Le coût qui en découle (et qui sera détaillé dans la suite), est appelé *Cycle Consistency Loss*.

Pour résumer le fonctionnement global du cycleGAN. Le générateur G qui assure la transformation A est optimisé pour tromper le discriminateur D_B comme dans un GAN classique, mais aussi aussi pour que à F fixé, $F = \nVdash$. Symétriquement, il en est de même pour le générateur F qui assure la transformation B. Les discriminateurs, quant à eux, sont mis à jour selon la même fonction de coût qu'un discriminateur de GAN classique. Les fonctions de coûts utilisées sont détaillées dans la partie suivante.

4.3 Les fonctions de coûts

4.4 Les métriques d'évaluations

4.5 Implémentation et résultats