Generative Adversarial Networks and Cycle-GAN
Deep Learning

Supervisé par Marco Cuturi

Eléonore Blanchard
ENSAE Paris
eleonore_blanchard@ensae.fr

Axel Durand
ENSAE Paris
axel_durand@ensae.fr

Abstract

L'objectif de notre projet est, après avoir présenté succinctement les Deep Convolutional Generative Adversarial Network (DCGAN), d'implémenter un algorithme CycleGAN pour convertir des chiffres écrits à la main (base de données MNIST[1]) pour qu'ils répondent au standard du United States Postal Service (base de données USPS[2]).

1 Introduction

Les modèles génératifs ont pour objectif de générer des imitations les plus réalistes possibles à partir de données réelles. L'algorithme doit donc tenter de "résumer" les données dans un premier temps, afin d'en générer des similaires avec les mêmes caractéristiques. Il a donc acquis une bonne compréhension des données fournies, qu'on pourra réutiliser pour d'autres tâches.
Les Generative Adversarial Network (GANs) sont des algorithmes génératifs dont la spécificité est qu'ils permettent d'ajuster les paramètres du modèle pour qu'il améliore son travail d'imitation. Dans un premier temps on entraîne le **discriminateur** à identifier les vraies et fausses données, puis on entraîne le **générateur** à produire des faux et le discriminateur détermine leur probabilité d'authenticité. L'objectif étant de maximiser cette probabilité, l'algorithme va ensuite "remonter" les gradients par backpropagation pour ajuster les paramètres du générateur. En réitérant ces deux phases d'apprentissage, on obtient un générateur créant des imitations très réalistes, et un discriminant capable de détecter les meilleures imitations.
Dans le cadre de notre projet, nous allons nous épancher sur deux algorithmes utilisant les GANs : les DCGAN, puis les CycleGAN dont l'objectif est de transférer les caractéristiques d'une image à l'autre (par exemple une image d'un cheval portant les rayures d'un zèbre).

[1] http://yann.lecun.com/exdb/mnist/
[2] https://www.kaggle.com/bistaumanga/usps-dataset

2 Revue bibliographique

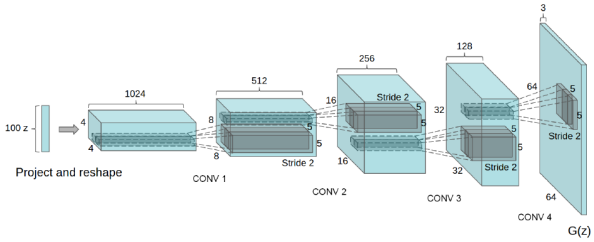
Dans le cadre de l'étude des DCGAN (DCGAN), nous allons présenter deux articles. Le premier article (Alec Radford, 2016) traite d'une nouvelle architecture de GANs (les DCGAN) ainsi que de ses caractéristiques. Le deuxième (Martin Arjovsky, 2017) traite des Wasserstein GANs et apporte des résultats théoriques sur la convergence des DCGANs.

2.1 DCGan

Le papier sur les DCGAN (Alec Radford, 2016) présente essentiellement des améliorations pour l'architecture des réseaux GANs. Les résultats théoriques sont plutôt abordés dans le deuxième papier. Les propositions sont validées par des résultats empiriques sur plusieurs datasets divers, comme LSUN (image de chambres d'hôtel), Imagenet-1k, SVHN (images de numéros de rue), ou CIFAR-10 (images de 10 classes différentes).
Le papier présente plusieurs résultats importants :

- Une version de GAN appelée Deep Convolutional Generative Adversarial Networks (DCGAN), qui sont stables à entraîner dans la plupart des cas.
- Les discriminateurs peuvent être utilisés pour de la classification
- Les filtres peuvent détecter des formes et objets spécifiques et peuvent être visualisés
- Les générateurs ont des propriétés arithmétiques vectorielles intéressantes qui permettent de manipuler facilement des caractéristiques des échantillons générés.

2.1.1 Lignes directrices de l'architecture



- Remplacer toutes les pooling layers par des strides convolutions (discriminateur) et des fractional-strided convolutions (générateur).
- Utiliser le batchnorm dans le générateur et le discriminateur.
- Supprimer les couches cachées fully-connected pour les architectures plus profondes.
- Utiliser l'activation ReLU dans le générateur pour toutes les couches sauf pour la sortie, qui utilise Tanh.
- Utiliser l'activation LeakyReLU dans le discriminateur pour toutes les couches.

Avec l'amélioration de la qualité des échantillons générés, on vérifie que qu'on n'a pas uniquement sur-ajusté ou mémorisé des échantillons d'entraînement. Afin de démontrer comment le modèle s'adapte à une plus grande quantité de données et à une résolution plus élevée, on entraîne un modèle sur l'ensemble de données LSUN contenant un peu plus de 3 millions d'exemples d'entraînement.
Théoriquement, le modèle pourrait apprendre à mémoriser des exemples d'entraînement, mais cela est expérimentalement peu probable car nous nous entraînons avec un petit learning step et des minibatchs SGD.

2.1.2 Validation empirique des capacités du DCGAN

Pour évaluer la qualité des algorithmes de génération d'image non supervisée, on les utilise comme extracteur de caractéristiques sur des datasets supervisées, puis on évalue la performance des modèles linéaires adaptés à ces caractéristiques. On évalue ainsi la qualité des caractéristiques produites par ce générateur sur le dataset CIFAR-10, avec un modèle SVM :

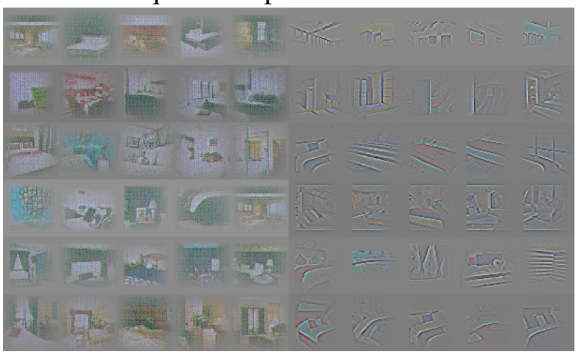
Model	Accuracy	Accuracy (400 per class)	max # of features units
1 Layer K-means	80.6%	63.7% (±0.7%)	4800
3 Layer K-means Learned RF	82.0%	70.7% (±0.7%)	3200
View Invariant K-means	81.9%	72.6% (±0.7%)	6400
Exemplar CNN	84.3%	77.4% (±0.2%)	1024
DCGAN (ours) + L2-SVM	82.8%	73.8% (±0.4%)	512

Comme le DCGAN n'a jamais été entraîné sur le dataset CIFAR-10, cette expérience montre également la robustesse des caractéristiques apprises.

2.1.3 Exploration et visualisation de la structure des réseaux

De petites modifications sur une série de 9 points aléatoires sur Z montre que l'espace appris présente des transitions fluides, chaque image de l'espace ressemblant plausiblement à une chambre d'hôtel. Nous pouvons voir ce qui semble être une télévision se transformer lentement en fenêtre. Il n'y a pas de transitions nettes, donc il n'a pas mémorisé de données d'entraînement spécifiques. Les petits changements finissent par modifier la construction de l'image (des objets sont ajoutés ou retirés). Le modèle a donc appris des représentations pertinentes et intéressantes.

Filtres du discriminateur et extraction de caractéristiques : Les caractéristiques apprises par le discriminateur s'activent sur des parties typiques d'une chambre d'hôtel, comme les lits et les fenêtres. Les caractéristiques initialisées au hasard ne sont pas activées sur tout ce qui est sémantiquement pertinent ou intéressant :

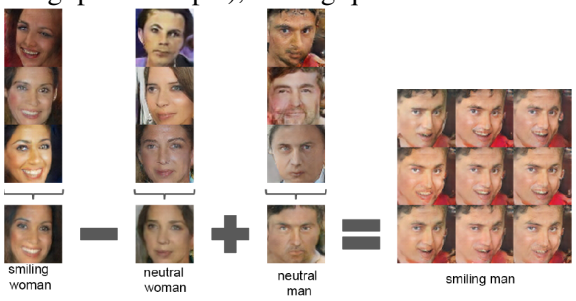


Random filters **Trained filters**

Modification des objets générés : Il est possible de forcer le générateur à ne pas générer certains objets, comme des fenêtres. On désigne manuellement les zones des fenêtres de certaines images, et on entraîne une régression logistique pour détecter les features des fenêtres, sur la deuxième couche de convolution. Alors on réduit à zéro les poids correspondant à ces features. Puis on relance le générateur : les fenêtres ne sont jamais produites.

2.1.4 Arithmétique des vecteurs générateurs

Les caractéristiques des vecteurs de génération d'un objet répondent à des lois d'arithmétiques très intéressantes, assez similaires à celles qu'on peut trouver pour le langage naturel une fois vectorisé (King - Man + Woman = Queen). Ici les essais montrent que s'il y a évidemment beaucoup plus de bruit (une infinité d'images sont possibles pour un seul mot, King par exemple), la logique reste la même.



De la même manière on peut fabriquer des angles de vue différent d'un visage, à partir d'échantillons vus de face, de droite ou de gauche.

2.2 Wasserstein Gan

Le papier sur les Wasserstein GANs (Martin Arjovsky, 2017) présente un amélioration du DCGAN en utilisant une nouvelle perte pour l'entraînement : l'Earth Mover (ou divergence de Wasserstein) au lieu de la Jensen-Shannon. Cela permet d'entraîner le discriminateur jusqu'à convergence, et ainsi d'éviter le compromis avec l'entraînement du générateur.

2.2.1 Apprentissage d'une approximation d'une distribution inconnue

Lorsque nous entraînons des modèles générateurs, nous supposons que les données dont nous disposons proviennent d'une distribution inconnue P_r. Nous voulons apprendre d'une distribution P_θ qui se rapproche de P_r, où θ sont les paramètres de la distribution. On peut imaginer deux approches pour cela :

Première approche : On peut apprendre directement la fonction de densité de probabilité P_θ. Autrement dit, P_θ est une fonction différentiable telle que $P_\theta(x) \geq 0$ et $\int_x P_\theta(x)dx = 1$. Nous optimisons P_θ par une estimation du maximum de vraisemblance (EMV). On apprend une fonction qui transforme une distribution existante Z en P_θ. Ici, g_θ est une fonction différentiable, Z est une distribution commune (uniforme ou gaussienne par exemple), et $P_\theta = g_\theta(Z)$
Étant donné la fonction P_θ, l'objectif de l'EMV est $\max_{\theta \in R^d} \frac{1}{m} \sum_{i=1}^{m} \log P_\theta(x^{(i)})$. En asymptote, cela équivaut à minimiser la divergence de Kullback-Leibler : $KL(P_r \| P_\theta)$.
Pour y remédier, on peut ajouter un bruit aléatoire à P_θ lors de la l'entraînement de l'EMV. Cela permet de s'assurer que la distribution est définie partout. Mais du coup, nous introduisons une certaine erreur, et empiriquement, il faut ajouter beaucoup de bruit aléatoire pour que les modèles s'entraînent bien, ce qui est dommage. De plus, même si nous apprenons une bonne densité P_θ, il peut être coûteux de calculer un échantillon à partir de P_θ, une fois entraîné.

Deuxième approche : On essaye donc la deuxième approche, qui consiste à apprendre un g_θ (un générateur) pour transformer une distribution connue Z. L'autre motivation est qu'il est très facile de générer des échantillons par la suite : Avec un g_θ entraîné, il suffit d'échantillonner un bruit aléatoire $z \sim Z$, et d'évaluer $g_\theta(z)$.
Pour entraîner g_θ (et par extension P_θ), nous avons besoin d'une mesure de la distance entre les distributions. On choisit La distance de Earth Mover (EM) (ou Wasserstein) : Soit $\Pi(P_r, P_g)$ l'ensemble de toutes les distributions conjointes γ dont les distributions marginales sont P_r et P_g. Alors,

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} E_{(x,y)\sim\gamma}[\, \|x - y\| \,]$$

est choisi parce qu'il existe des séquences de distributions qui ne convergent pas sous la divergence JS, KL, KL inverse ou TV, mais qui convergent sous la distance EM.
Parmi les distances JS, KL et Wasserstein, seule la distance de Wasserstein offre des garanties de continuité et de différentiabilité, deux éléments que l'on souhaite réellement dans une fonction de perte.
De plus, toute distribution qui converge sous les divergences KL, KL inverse, TV et JS converge également sous la divergence de Wasserstein.
La distance de Wasserstein est donc une fonction de perte convaincante pour les modèles générateurs.

2.2.2 Approximation de la distance de Wasserstein

Malheureusement, le papier montre que le calcul exact de la distance de Wasserstein est difficile, mais le papier montre également comment nous pouvons en faire une approximation.
Un résultat de la dualité Kantorovich-Rubinstein montre que W est équivalent à

$$W(P_r, P_\theta) = \sup_{\|f\|_L \leq 1} E_{x \sim P_r}[f(x)] - E_{x \sim P_\theta}[f(x)]$$

(où le sup est pris sur toutes les fonctions de 1-Lipschitz.) Nous pouvons prendre le sup sur les fonctions K-Lipschitz $\{f : \|f\|_L \leq K\}$, ce qui est encore difficile, mais plus facile à approximer, nous n'avons pas besoin de connaître K, il suffit de savoir qu'il existe, et qu'il est fixé tout au long du processus d'entraînement. Les gradients de W seront mis à l'échelle par un K inconnu, mais également par le taux d'apprentissage α, où K est absorbé dans l'accord de l'hyperparamètre.
Intuitivement, avec un g_θ fixe, nous pouvons calculer le f_w optimal pour la distance de Wasserstein. Nous pouvons alors faire de la backpropagation à travers $W(P_r, g_\theta(Z))$ pour obtenir le gradient pour θ.

$$\nabla_\theta W(P_r, P_\theta) = \nabla_\theta (E_{x \sim P_r}[f_w(x)] - E_{z \sim Z}[f_w(g_\theta(z)) - E_{z \sim Z}[\nabla_\theta f_w(g_\theta(z))]$$

L'entraînement est divisé en trois étapes :

- Pour un θ fixé, calcul d'une approximation de $W(P_r, P_\theta)$, en entraînant f_w jusqu'à convergence.
- Une fois le f_w optimal trouvé, calcul du gradient θ $-E_{z \sim Z}[\nabla_\theta f_w(g_\theta(z))]$ en échantillonnant plusieurs $z \sim Z$.
- Mise à jour de θ, et répétition du processus.

Toute cette dérivation ne fonctionne que lorsque la famille de fonctions $\{f_w\}_{w \in \mathcal{W}}$ est K-Lipschitz. Pour garantir cela, nous utilisons le weight clamping : les poids w sont contraints à se situer dans la plage de $[-c, c]$, en coupant w après chaque mise à jour à w.

2.2.3 Algorithme du Wasserstein GAN

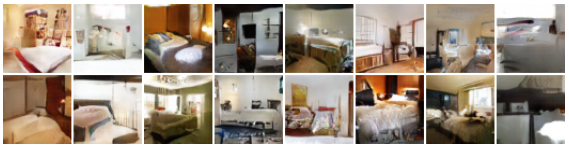
Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

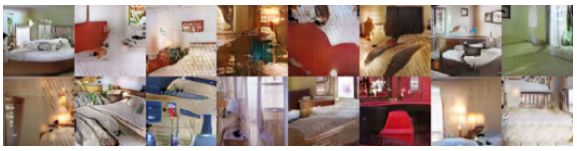
while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Les résultats sont un légèrement meilleurs que ce ceux du DC-GAN, lorsqu'on utilise la même architecture, comme le montrent les essais suivants.

$WGan$:



$DCGan$:



3 Implémentation

Dans cette partie, nous allons tout d'abord présenter l'implémentation d'un type de GAN qui se prête particulièrement bien au traitement d'images DCGAN (que nous avons présenté dans notre première partie). Nous l'entraînerons à générer des chiffres manuscrits.
Puis nous implémenterons une architecture plus complexe de GAN : le CycleGAN utilisé pour la traduction image-à-image (image-to-image translation). Nous l'entraînerons à convertir des chiffres écrits manuellement présents dans la base MNIST, en chiffres du standard United States Postal Service (USPS).

3.1 Deep Convolutional GAN (DCGAN)

Comme nous l'avons présenté dans la première partie, les DCGAN sont des GANs spécifiques dont le discriminateur est un réseau de neurones convolutif et le générateur un réseau de neurones

composé de convolutions transposées.
Pour implémenter le DCGAN, nous nous baserons sur la base de données Tensorflow MNIST qui représente des chiffres écrits à la main :

3.1.1 Générateur

Le générateur a pour objectif de générer des chiffres manuscrits en prenant pour modèle la base de données MNIST.
Le générateur produit une image à partir d'un bruit aléatoire en utilisant les couches de tf.keras.layers.Conv2DTranspose. On commence avec une couche Dense (tf.keras.layers.Dense), où tous les neurones sont liés entre eux, qui prend ce bruit aléatoire en entrée, puis on sur-échantillonne plusieurs fois jusqu'à atteindre la taille désirée de l'image (28x28x1). On utilise une activation LeakyReLu (tf.keras.layers.LeakyReLU) pour toutes les couches, sauf la couche de sortie pour laquelle on utilise une activation Tahn.

3.1.2 Discriminateur

Le discriminateur a pour objectif d'identifier les vraies images de chiffres manuscrits et les images générées par le générateur.
Le discriminateur est un classifieur : il détermine la classe de l'image (vraie ou fausse). Nous utilisons un classifieur CNN (Convolutional Neural Network) classique, utilisé dans la classification d'image, en utilisant des couches de tf.keras.layers.Conv2D et une activation ReLu.

3.1.3 Pertes

Les pertes du discriminateur et du générateur utilisent une cross-entropy binaire qui permet d'évaluer une perte dans le cadre d'une classification binaire.

Perte du discriminateur : La perte du discriminateur mesure à quel point le discriminateur est capable de distinguer les vraies images des fausses. Il compare les prédictions du discriminateur sur des vraies images à un array de 1, et sur des fausses images à un array de 0.
La perte du discriminateur est donc égale à la somme de deux pertes :

- perte sur la classification de vraies images : à quel point le discriminateur réussit à déterminer qu'une vraie image est vraie
- perte sur la classification de fausses images : à quel point le discriminateur réussit à déterminer qu'une fausse image est fausse

Perte du générateur : La perte du générateur mesure à quel point le générateur est capable de tromper le discriminateur. Intuitivement, si le générateur performe bien, le discriminateur va classifier toutes les fausses images comme des vraies. On compare ici les décisions du discriminateur sur les images générées à un array de 1.
La perte du générateur est donc égale à - la perte de classification de faux du discriminateur : à quel point le discriminateur se trompe et détermine qu'une fausse image est vraie.

3.1.4 Entraînement

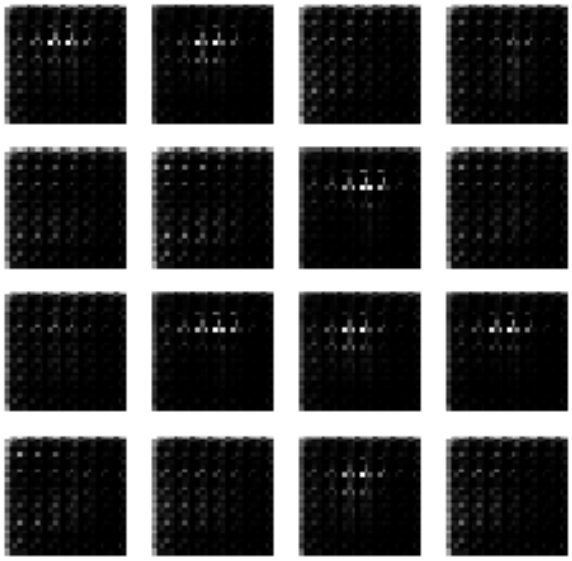
Une boucle d'entraînement se déroule en 3 étapes :

1. Le générateur reçoit le bruit aléatoire comme input et l'utilise pour produire une image.
2. Le discriminateur classifie les vraies images (issues du set d'entraînement) et les fausses images (produites par le générateur).
3. La perte est calculée pour les deux modèles, et les gradients sont utilisés pour mettre à jour le générateur et le discriminateur.

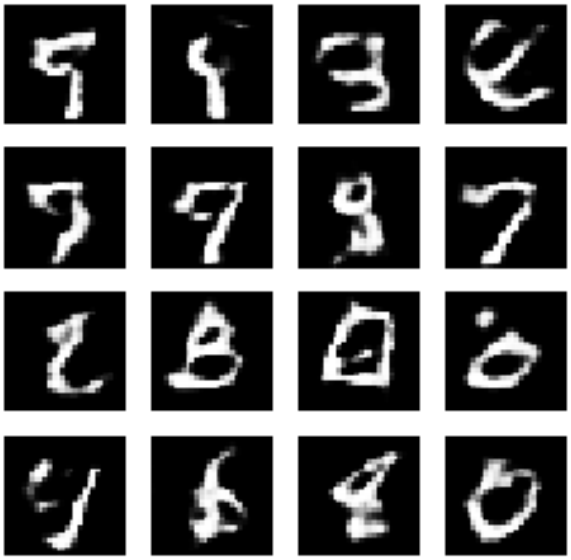
3.1.5 Résultats

Au début de l'entraînement, les images générées ressemblent à des bruits aléatoires. Puis, au fur et à mesure de l'entraînement, des chiffres manuscrits très réalistes apparaissent. Au bout de 40 epochs, ils ressemblent aux chiffres MNIST et ce sont nettement améliorés par rapport à l'epoch 1.

$epoch1$:



$epoch40$:



3.2 CycleGAN

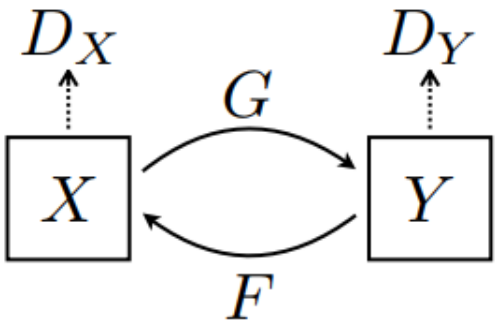
Les CycleGANs servent à faire des traductions image-à-image. Cela consiste à convertir automatiquement une image en une nouvelle image avec une apparence désirée. Par exemple dans le cadre de notre projet, nous travaillons sur des chiffres écrits à la main. L'image en entrée est une image de la base MNIST et l'image en sortie suit le standard du service postal américain (USPS).
L'idée derrière les CycleGANs est d'utiliser des GANs pour générer des nouvelles images, mais sous des nouvelles contraintes : ressembler à une image en input, mais avec les caractéristiques des images de la deuxième base de données. L'avantage des CycleGANs est qu'on peut les utiliser sur des données qui ne sont pas pré-couplés : ils s'appliquent aux problèmes non supervisés.
Pour implémenter le CycleGan nous avons eu recours à un papier traitant des Unpaired Image-to-Image Translation (Jun-Yan Zhu, 2018) et à la documentation de Tensorflow sur les CycleGANs[3].
Pour l'implémentation du CycleGAN, nous allons effectuer une traduction image-à-image en passant de la base de données MNIST (que nous avions précédemment utilisé pour l'implémentation du DCGAN) à la base de données USPS.

3.2.1 Modèle Pix2Pix

Pour l'implémentation du CycleGAN, nous réutilisons les générateur et discriminateur du modèle Pix2Pix, qui fait de la traduction image-à-image avec supervision. Nous importons le modèle à partir d'une librairie d'exemples de Tensorflow.
Dans les deux modèles on a deux générateurs (G et F) et deux discriminateurs (D_X et D_Y) :

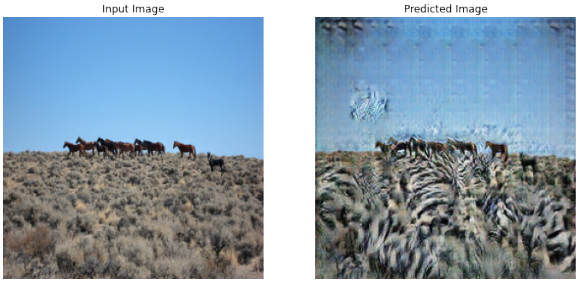
- Le générateur G apprend à transformer les images X en image Y ($G(X) -> Y$)
- Le générateur F apprend à transformer les images Y en image X ($F(Y) -> X$)
- Le discriminateur D_X apprend à différencier les images X des images générées de X ($F(Y)$)
- Le discriminateur D_Y apprend à différencier les images Y des images générées de Y ($G(X)$)

[3] https://www.tensorflow.org/tutorials/generative/cyclegan



3.2.2 Pertes

Contrairement au modèle Pix2Pix, les données ne sont pas couplées, il n'y a donc aucune garantie que l'entrée et que le couple cible ait un sens. Par exemple, on peut mettre en entrée une photo de cheval et souhaiter en sortie obtenir la même photo avec un zèbre à la place. Mais le modèle pourrait également zébrer un autre élément, comme le cavalier ou le sol par exemple :



C'est pourquoi, en plus des pertes des discriminateurs et des générateurs (identiques à celles du DCGAN), l'auteur ajoute une autre perte : la cycle consistency loss (perte de consistance du cycle). Un cycle de traduction est constant si la traduction d'une traduction est égale à son entrée. En d'autres termes, traduire une phrase du français vers l'anglais puis retraduire le résultat de l'anglais vers le français devrait résulter en la phrase originelle. Il en va de même pour les images.

Pour calculer la perte, on commence par transformer une image X via le générateur G : $G(X) = \hat{Y}$. Puis on transforme l'image $\hat{Y}$ via le générateur F : $F(\hat{Y}) = \hat{X}$. On peut ensuite calculer la perte forward avec la MAE (mean absolute error) entre X et la perte backward avec la MAE entre Y et $\hat{Y}$.

L'auteur rajoute également une identity loss (perte d'identité) qui vérifie que le générateur chargé de transformer une image X (resp. Y) en une image Y (resp. X) génère la vraie image la vraie image Y (resp. X) si Y est l'input. En d'autres termes, on devrait avoir : $G(Y) = Y$ et $F(X) = X$. La perte calcule la somme des écarts.

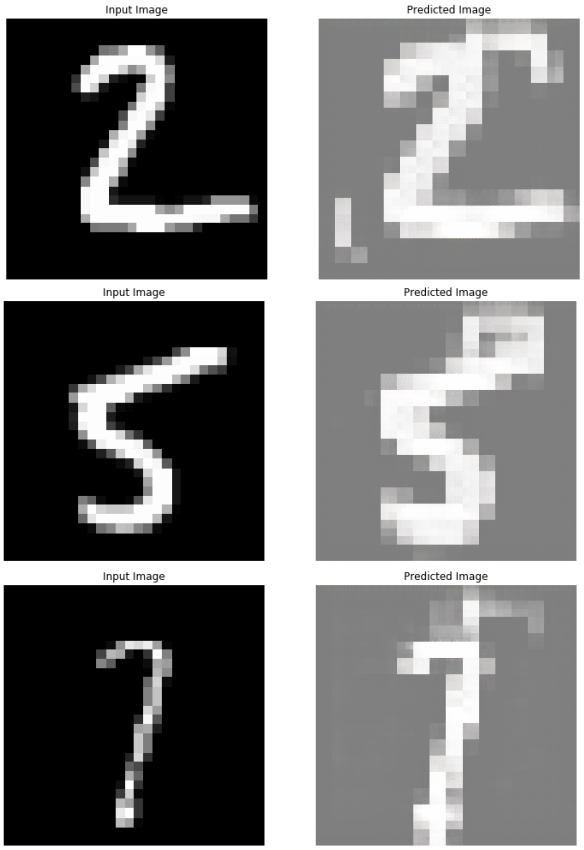
3.2.3 Entraînement

La boucle d'entraînement est composée des 4 étapes suivantes :

1. Génération des images ($G(X)$, $F(G(X))$, $F(Y)$, $G(F(Y))$, $F(X)$ et $G(Y)$) et évaluation par le discriminateur ($D_X(X)$, $D_Y(Y)$, $D_X(F(Y))$ et $D_Y(G(X))$)
2. Calcul des pertes des générateurs, des discriminateurs, de la consistance de cycle et d'identité.
3. Calcul des gradients par backpropagation
4. Mise à jour des gradients dans la fonction d'optimisation

3.2.4 Résultats

Le temps de calcul et l'espace mémoire important exigés par ces algorithmes ne nous ont pas permis d'aller au delà de 20 époch d'entraînement pour ce modèle et sur une partie des données. Les résultats ne sont donc pas définitifs et peuvent être améliorés avec les ressources nécessaires. Toutefois, voici nos résultats obtenus après une nuit d'exécution sur nos machines :



La marge d'amélioration reste importante, mais ces résultats sont tout de même largement probants : ils ressemblent encore beaucoup aux images MNIST d'origine, et on bien été transposés dans le style légèrement différent de USPS (ils touchent les bords de l'image par exemple). Le double objectif principal est donc rempli.

4 Conclusion/Discussion

Ces deux papiers de recherche sont donc très importants dans le champ de la génération synthétique d'image.
Le premier parce qu'il présente une nouvelle architecture avec des résultats empiriques impressionnants, comme les propriétés algébriques des espaces générateurs, la capacité d'apprendre depuis du non-supervisé, ou de modifier la sortie d'un générateur en modifiant quelques point de l'espace latent.
Le second parce qu'il conforte ces avancées avec des preuves théoriques et une amélioration de la convergence des discriminateurs pour l'entraînement.
Ils sont donc parfaitement complémentaires et permettent de générer, moyennant tout de même un certain temps de calcul, des échantillons synthétiques de grande qualité, comme on a pu le voir avec notre génération de chiffre MNIST du premier paragraphe. Ces architectures peuvent par ailleurs être combinées pour aller encore plus loin, et transformer des images bien réelles, grâce aux architectures CycleGAN, dont nous avons présenté une implémentation.

References

Soumith Chintala Alec Radford, Luke Metz. 2016. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

Léon Bottou Martin Arjovsky, Soumith Chintala. 2017. Wasserstein gan. *arXiv preprint arXiv:1701.07875v3*.

Phillip Isola Alexei A. Efros Jun-Yan Zhu, Taesung Park. 2018. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*.