

Génération de synthèse de texture avec les GANs

BOUJNOUNI Fatine, JACOB Etienne

March 2019

1 Introduction

Le terme texton a été introduit par Bela Julesz en 1981 pour décrire "les atomes de la perception humaine pré-attentive de la texture". Or, le mot "texton" reste un concept vague dans la littérature faute d'un bon modèle mathématique, ou encore un meilleur représentant particulier de la perception humaine à une certaine texture. Une bonne synthèse des textons aiderait à avoir une bonne classification de textures, à l'analyse des textures, mais aussi à réussir des opérations d'Inpainting en faisant des synthèses de texture.

L'objectif de ce projet est de synthétiser des images qui procurent la même sensation visuelle, sans être de simples copies de l'image initiale. Ces images sont des textures qui peuvent être de toutes tailles.

Le problème de la synthèse de texture a été examiné en utilisant plusieurs approches telles que celle de Gatys et al, où nous essayons une descente de gradient sur des images aléatoires pour encourager les corrélations des caractéristiques d'un CNN (Convolutional Neural Network) à ressembler à celle de la texture souhaitée. Une autre approche raisonnable à ce problème consiste à utiliser un GAN (Generative Adversarial Network) pour produire des exemples aléatoires de textures.

Pour commencer notre projet, nous avons été guidés par les travaux de Jetchev et al. ([1]) qui fournissent des détails algorithmiques pour mettre en œuvre une telle méthode et architecture de réseau de neurones. La technique en question s'appelle Spatial GAN (SGAN).

2 GAN : Réseaux antagonistes génératifs

2.1 Introduction aux GANs

Les réseaux d'adversaires génératifs (ou GAN) sont l'un des algorithmes d'apprentissage automatique les plus populaires développés récemment. Ils appartiennent à l'ensemble des algorithmes appelés modèles génératifs. Ces modèles apprennent la fonction de distribution intrinsèque des données d'entrée $p(x)$.

L'application la plus populaire des GANs semble concerter la génération d'image, et on ne parlera que de ce cadre dans ce rapport, même si les GANs sont une technique plus générale. L'utilisation de réseaux de neurones faisant usage de couches de convolutions est assez systématique aujourd'hui lorsque traitement d'image et deep learning se rencontrent. Les techniques de deep learning à base de réseaux de neurones à convolution ont d'abord été particulièrement efficaces pour associer des étiquettes à des images. Le problème de la génération de nouvelles données similaires à une base de données d'apprentissage se révèle plus compliqué. Les GANs ont été introduits par Goodfellow en 2014 [2]. Les GANs ont été une grande avancée en terme de réalisme des données générées, et leur espace latent (dont nous parlerons ensuite) a des propriétés très intéressantes comme l'interpolation entre deux images en restant dans l'espace d'images que l'on veut générer : par exemple si on apprend à générer des images de visages, on pourra facilement produire une suite continue de visages entre toute paire de visages générée.

Les GANs se sont montrés efficaces dans la modélisation et la génération de données de grande dimension. Néanmoins, ils ne sont pas les seuls types de modèles génératifs, les autres incluent les autoencodeurs variationnels (VAE) et pixelCNN / pixelRNN et les NVP réelles. Chaque modèle a ses propres compromis.

2.2 Fonctionnement d'un GAN

2.2.1 Principe du GAN

L'idée des GANs est la suivante : on définit deux modèles (qui sont en pratique définis par des réseaux de neurones). Il y a un modèle générateur G et un autre dit discriminateur D . G génère des données semblables à un ensemble de données, lorsqu'on lui donne du bruit z en entrée qui appartient à l'espace dit "latent". D essaie de trouver la probabilité $D(x)$ qu'une donnée x (une image dans notre cas) appartienne à l'ensemble des données d'entraînement (et ne soit pas générée par G). G et D sont entraînés pour optimiser une fonction objectif "adversaire" : G est optimisé pour que D échoue dans sa tâche de reconnaissance (ce qui l'encourage à produire des données qui ressemblent le plus possible aux données d'entraînement).

Pour être plus précis, les GANs sont entraînés suivant l'objectif suivant (venant de [2]) :

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

On reconnaît la fonction de perte "binary cross entropy" pour l'optimisation de D . Le premier terme de la somme signifie que D essaie de répondre 1 sur les données d'entraînement, et le second terme signifie qu'il essaie de répondre 0 sur les données générées par G . D'autre part G fait l'optimisation dans l'autre sens sur le second terme, ce qui veut dire que G est optimisé pour que D de réponde pas 0 sur les données qu'il génère.

2.2.2 Entraînement de GANs

L'article fondateur des GANs [2] donne un algorithme basé sur la descente ou montée de gradient stochastique (SGD) pour optimiser les réseaux selon l'objectif précédent.

Son principe est le suivant : pour chaque itération de l'algorithme, on optimise d'abord D par montée de gradient sur k étapes (on peut prendre $k = 1$). Le gradient stochastique est calculé en utilisant un mini batch de taille m : on prend m exemples des données d'entraînement et m valeurs de bruit z qui conduisent à m fausses données générées avec le G actuel. Après ces k étapes, on fait une descente de gradient sur G à partir de m échantillons de bruit. Pour résumer, on alterne entre entraîner D à reconnaître les vraies données d'apprentissage, et entraîner G à tromper D .

En pratique la descente ou montée de gradient stochastique de base peut être remplacée par un méthode à base de gradient plus puissante telle que Adam [3].

L'entraînement des GANs peut être difficile. Nous avons observé qu'il faut éviter que le discriminateur soit trop performant car G a plus de mal à découvrir comment progresser dans ce cas. Si G est très performant c'est également plus dur pour D mais nous voulons que G soit très performant. Il semble être bon signe que les pertes changent beaucoup au cours des itérations, des pertes quasi constantes signifiant peu d'évolution. Il faut également que le discriminateur D ait une précision (accuracy) strictement au dessus de 50% la plupart du temps pour que G ait un critère solide pour progresser. Un discriminateur qui répond toujours la même chose aura une precision de 50% donc on cherche mieux que cela. Par contre si le générateur est devenu très bon, on peut s'attendre à ce que le discriminateur ne sache plus distinguer les données générées des vraies données et tende vers une accuracy de 50%.

2.3 Les avantages et les inconvénients les plus pertinents des GANs

2.3.1 Avantages des GANs

Les GANs génèrent actuellement les images les plus nettes. Des méthodes à base de GAN comme "styleGAN" [4] permettent de générer des images sem-

blables à des photographies tout en restant pourtant très génératives dans le sens où les données générées sont bien distinctes des données d'entraînement. La figure 1 montre des résultats de visage humain générés après entraînement de GAN, avec styleGAN. Les GANs que nous utilisons dans ce projet sont différents, mais ces résultats illustrent bien la capacité de réalisme que peuvent avoir les méthodes à base de GANs.



Figure 1: Résultats issus de l'article de styleGAN [4]

Les GANs sont faciles à former (aucune inférence statistique n'est nécessaire), et seule la rétropropagation est nécessaire pour obtenir des gradients.

En se déplaçant de manière continue dans l'espace latent on génère des données de sortie qui ressemblent aux données d'apprentissage (et qui peuvent donc être des images toujours réalistes) de manière continue. D'autres modèles génératifs par exemple à base d'autoencoders peuvent également avoir cette propriété utile avec un espace latent. Cet aspect est représenté en figure 2, on remarque que bien qu'utilisant des valeurs dans l'espace de latent autres que celles censées pouvoir générer un chiffre, les images générées ressemblent presque toujours à des chiffres. Nous avons pensé que cette image représente bien un intérêt essentiel des modèles génératifs comme celui du GAN. Le modèle génératif dans ce cas n'a pas été entraîné comme un GAN mais comme un Variational AutoEncoder, mais cette image illustre très bien le principe d'interpolation subtile, et on peut penser qu'un GAN aurait pu obtenir une interpolation encore plus satisfaisante.

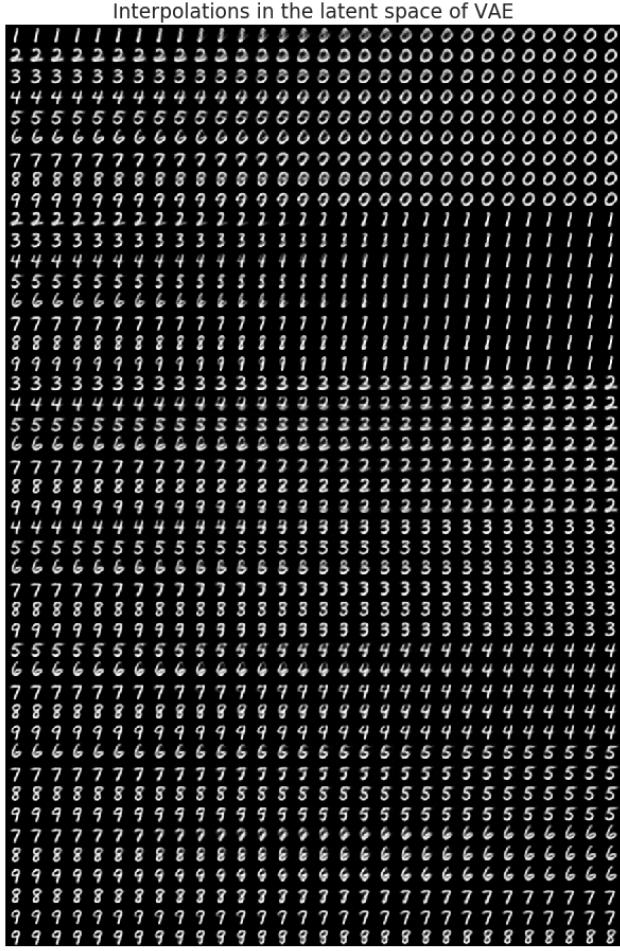


Figure 2: Ici un modèle génératif a été entraîné sur la base de données de chiffres manuscrits MNIST. Sur chaque ligne, les résultats en faisant une interpolation linéaire dans l'espace latent sont représentés.

Les GANs font une sorte de réduction de la dimensionnalité par l'espace latent. Les images qui nous intéressent d'un ensemble de données sont loin d'exploiter toutes les possibilités de couleurs pour chaque pixel indépendamment. La structure des images fait qu'on peut penser qu'il soit possible de les encoder avec une dimension réduite, et c'est ce que semblent pouvoir faire les GANs.

Les GANs sont efficaces pour modéliser une distribution continue qui étend des données d'entraînement.

2.3.2 Inconvénients des GANs

Les GANs sont difficiles à optimiser en raison d'une dynamique d'entraînement instable. C'est ce qui nous a posé le plus de problème lors de notre expérimentation dans ce projet.

Aucune inférence statistique ne peut être faite avec eux. Les GANs appartiennent à la classe des modèles de densité implicites directs, ils modélisent $p(x)$ sans définir explicitement la fonction de densité de probabilité.

Pour nuancer l'intérêt de l'espace latent évoqué précédemment, l'interpolation entre échantillons de bruit semblables à ceux utilisés à l'entraînement n'a pas forcément les statistiques attendues, ce qui peut conduire à des données générées de moins bonne qualité.

3 Différentes approches de synthèse de texture

La qualité d'une méthode de synthèse de texture peut être évaluée visuellement par la proximité de la texture synthétisée à l'originale, mais également par d'autres critères pouvant être spécifiques à une application, par exemple: la rapidité d'analyse et de synthèse, la capacité à générer diverses textures de taille arbitraire, la possibilité de créer des textures à morphing régulier.

Il existe précédemment à Spatial GAN deux grandes catégories de techniques de synthèse de textures :

- **Les techniques non-paramétriques :** Elles reprennent des pixels ou des patchs entiers à partir d'exemples de textures, en randomisant efficacement la texture en entrée de manière à préserver ses propriétés de perception visuelle. Ces techniques produisent des images de haute qualité mais ont des inconvénients. Par exemple elles réorganisent la texture en entrée en se basant juste sur les similarités locales et sans connaître des modèles de textures qui présentent un intérêt. Aussi, elles peuvent prendre beaucoup de temps de calcul lorsqu'on doit synthétiser des textures larges à cause de toutes les routines de recherche impliquées.
- **Les techniques reposant sur l'appariement de propriétés statistiques ou de descripteurs d'images :** Ces méthodes font la synthèse de texture en cherchant des images avec des descripteurs similaires. Cela se fait généralement en résolvant un problème d'optimisation dans l'espace des pixels de l'image. Le travail de Portilla and Simoncelli [5] est un très bon exemple pour cette approche.

Gatys et al [6] présentent cependant une approche paramétrique basée sur les données qui permet de générer des textures de haute qualité sur une variété d'images naturelles. Ils utilisent les corrélations de filtres dans différentes couches de réseaux de convolution (entraînés sur de grandes collections d'images naturelles :ImageNet) pour créer une technique puissante qui capture mieux les statistiques des images.

Cependant, leurs méthodes nécessitent un processus d'optimisation lent et gourmand en mémoire pour générer une texture unique. C'est dans ce cadre qu'un papier plus récent [7] propose une approche alternative qui déplace le fardeau informatique vers une phase d'apprentissage. À partir d'un seul exemple de texture, cette approche consiste à former des réseaux convolutifs à feed-forward compacts pour générer plusieurs échantillons de la même texture de taille arbitraire et pour transférer le style artistique d'une image donnée à une autre image. Un réseau distinct est donc formé pour chaque texture d'intérêt et peut ensuite créer rapidement une image avec les statistiques souhaitées en un seul passage. Les réseaux résultants sont remarquablement légers et peuvent générer des textures de qualité comparable à celle de *Gatys et al*, mais des centaines de fois plus rapide.

Une autre approche générative a été proposée dans le cadre de la synthèse de texture [8] et qui utilise des réseaux de neurones récurrents (LSTMs) pour capturer les dépendances statistiques à longue portée. Le modèle proposé par ce papier introduit un modèle d'image récurrente basé sur des unités de mémoire multidimensionnelles à long terme et à court terme, particulièrement adaptées à la modélisation d'images en raison de leur structure spatiale. Ce modèle donne de bon résultats lorsqu'il est utilisé pour la synthèse et l'inpainting de textures.

Dans la partie qui va suivre, on va présenter une nouvelle classe de modèles paramétriques génératifs pour la synthèse de texture, en utilisant une architecture entièrement convulsive formée selon un critère de compétition entre deux réseaux antagonistes.

4 Synthèse de texture avec le GAN Spatial

Dans cette partie, on va parler d'un modèle de GAN (Spatial GAN) proposé dans l'article "Texture Synthesis with Spatial Generative Adversarial Networks" de *N. Jetchev, U. Bergmann, R. Vollgraf*, [1] et qui présente un nouveau modèle de synthèse de texture basé sur l'apprentissage GAN. En étendant l'espace de distribution du bruit d'entrée d'un vecteur unique à un tenseur spatial entier, nous créons une architecture avec des propriétés bien adaptées à la tâche de synthèse de la texture, que nous appelons GAN spatial (SGAN). Il s'agit de la première méthode de synthèse de texture basée sur les GAN, entièrement pilotée par les données.

4.1 Approche paramétrique générative de génération de texture

Comme on a vu dans la partie où on détaille le fonctionnement des GAN, ces modèles sont suffisamment puissants pour générer des images naturelles de haute qualité qui peuvent tromper les observateurs humains.

Sauf que dans les GANs, la taille de l'image de sortie est codée en dur dans l'architecture du réseau. Ceci est une limitation pour la synthèse de texture, où on peut avoir besoin de textures de tailles variées et beaucoup plus grandes.

Une méthode reposant sur les pyramides de laplacien [9] a été proposée pour générer des images de taille croissante , ajoutant progressivement plus de détails aux images avec des GAN conditionnels empilés. Cette approche utilise une cascade de réseaux de convolution dans un cadre pyramidal laplacien pour générer des images de manière grossière à fine. À chaque niveau de la pyramide, un modèle de convnet génératif distinct est formé à l'aide de la méthode de réseaux d'adversaire génératif (GAN). Cependant, cette technique reste limitée dans les tailles d'image de sortie qu'elle peut gérer, car elle doit former des modèles GAN de plus en plus complexes pour chaque échelle de la pyramide des images. Les niveaux d'échelle doivent également être spécifiés à l'avance afin que la méthode ne puisse pas générer de sortie de taille arbitraire.

Dans le modèle proposé par l'article (SGAN), on met en entrée des images de textures en tant que distribution de données que le GAN doit apprendre. L'architecture du modèle ressemble à celle de DCGAN [10]. On a modifié cette dernière pour permettre l'évolutivité et la possibilité de créer la taille de texture de sortie souhaitée en utilisant une architecture purement convolutionnelle sans aucune couche entièrement connectée. Cette architecture est bien adaptée à la synthèse de texture de grande taille arbitraire, elle représente une nouvelle application des méthodes antagonistes génératives.

4.2 Le modèle SGAN

Le SGAN généralise le générateur $G(Z)$ pour mapper un tenseur $Z \in \mathbb{R}^{l \times m \times d}$ sur une image $X \in \mathbb{R}^{h \times w \times 3}$. l et m représentent les dimensions spatiales et d le nombre de canaux. Comme dans les GANs simples, la variable Z est échantillonnée à parir d'une distribution antérieure : $Z \sim p_Z(Z)$.

L'architecture du GAN impose une contrainte à l'entrainement sur les dimensions l , m , h et w . Si nous avons un réseau avec k couches de convolution de foulée(stride) 1 et un remplissage identique 0 (same zero padding), la relation entre les dimensions s'écrit alors : $h = w = 2^k$.

De la même façon qu'on a étendu le générateur, le discriminateur D mappe un champs de deux dimensions de taille $l \times m$ contenant des probabilités indiquant si une entrée X (ou X') est réelle ou générée. (Voir la figure 3).

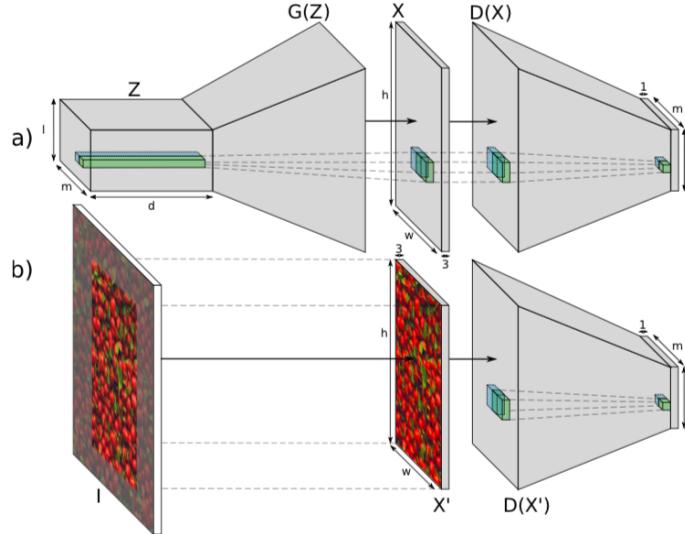


Figure 3: Architecture du SGAN [1]

On optimise simultanément le discriminateur et le générateur sur toutes les dimensions de l'espace:

$$\begin{aligned} \min_G \max_D V(D, G) = & \frac{1}{lm} \sum_{\lambda=1}^l \sum_{\mu=1}^m \mathbb{E}_{z \sim p_z(z)} [\log(1 - D_{\lambda\mu}(G(z)))] \\ & + \frac{1}{lm} \sum_{\lambda=1}^l \sum_{\mu=1}^m \mathbb{E}_{x \sim p_{data}(X)} [\log(D_{\lambda\mu}(X'))] \end{aligned}$$

En pratique, il est préférable d'appliquer une astuce utilisée dans plusieurs papiers : minimiser $-\log(D(G(Z)))$ au lieu de $\log(1 - D(G(Z)))$

5 Implémentation du SGAN

Dans cette partie, on va parler de l'implémentation de l'article qu'on a faite. L'architecture du SGAN a été inspirée de celle de DCGAN, et donc, dans un premier temps on a repris l'architecture de DCGAN et on l'a implémenté avec *Keras*. On l'a testé d'abord sur les images MNIST pour s'assurer qu'elle marche bien et qu'on n'a pas d'erreur d'implémentation (Voir l'architecture du DCGAN sur la figure 4).

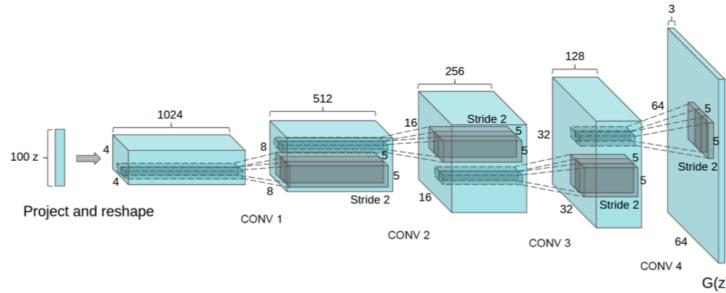


Figure 4: Architecture du générateur du DCGAN [10]

Ensuite, on a apporté des modifications à notre architecture pour qu'elle soit fully-convolutional.

Notre architecture se compose de couches convolutionnelles avec des foulées(strides) de $\frac{1}{2}$ dans le générateur, et de couches convolutionnelles avec des foulées(strides) de 2 en discriminateur. La taille du noyau est de 5×5 et on a utilisé le zero padding. La distribution de $p_z(z)$ est uniforme avec un support sur $[-1, 1]$. On a appliqué *BatchNormalization* à toutes les couches, sauf la dernière couche du générateur G, et la première et dernière couche du discriminateur D. Les poids du réseau ont été initialisés avec des gaussiennes de moyenne nulle et d'écart type $\sigma = 0.02$. On a implémenté un SGAN4, ce qui veut dire qu'on a une architecture avec 4 couches. Les paramètres qu'on a choisi sont ceux qui étaient indiqués sur l'article :

- Nombre de canaux de Z : $d = 20$
- $r = \frac{r}{l} = \frac{w}{m} = 16 = 2^4$
- Nombre de filtres du générateur G: $256 - 128 - 64$
- Nombre de filtres du discriminateur D: $64 - 128 - 256$

On a entraîné notre SGAN sur des patchs de taille 128×128 d'une texture.

6 Méthode de génération de texture de Gatys

Cette section présente une autre méthode de génération de texture, déjà évoquée dans ce rapport : celle de Gatys ([6]). Nous voulons faire la comparaison avec la méthode SGAN. Pour la résumer, c'est également une méthode à base de réseau de neurones à convolutions, mais l'approche reste très différente de SGAN. Elle essaie de retrouver par optimisation sur une image les statistiques d'une texture en utilisant la capacité de traitement d'image d'un réseau déjà entraîné.

6.1 Introduction

La méthode de Gatys n'appartient pas à la classe de méthode non-paramétriques qui réutilisent aléatoirement une texture initiale. Elle reprend une idée déjà utilisée auparavant qui est de se servir de mesures sur une image et de chercher à produire des images de mesures semblables à la texture initiale. Ces mesures doivent correspondre à la perception que nous avons des images de sorte à ce qu'elles soient similaires pour des images différentes mais que nous percevons comme des même textures. Dans cet esprit, avant Gatys, [5] utilisait déjà des statistiques faites à la main pour générer des textures. L'idée de la méthode de Gatys est d'utiliser la capacité de représentation d'un réseau de neurones à convolution, dans ses couches intermédiaires, entraîné pour de la classification d'images afin d'obtenir des statistiques pertinentes, avec des représentations invariantes par translation.

D'une certaine façon, même si cette méthode est considérée comme lente, elle a constitué l'état de l'art en génération de texture par méthode paramétrique au moment de sa publication.

6.2 Réseau utilisé

Le réseau utilisé dans la méthode pour obtenir les caractéristiques recherchées est VGG-19. Son architecture consiste en une première section de 5 blocs chacun composé de couches de convolutions avec activation ReLU puis d'un max pooling. Ensuite il y a des couches fully connected puis un softmax pour la classification, mais ces couches ne sont pas utilisées. Ce réseau a été entraîné sur la base de données ImageNet. En fait la méthode de Gatys le change un peu en remplaçant le max pooling par un average pooling car cela a empiriquement amélioré les résultats. Les poids ont également subi un changement : un changement d'échelle.

6.3 Modèle de texture

L'idée de [5] est reprise : calculer des statistiques sur la texture initiale à partir de filtres, puis faire une descente de gradient à partir d'une image initiale pour essayer d'obtenir les mêmes statistiques. Ce qui change avec la méthode

de Gatys est principalement la qualité des statistiques utilisées grâce à la performance de VGG-19.

Les statistiques utilisées dans la méthode de Gatys sont les corrélations entre les différentes "features maps" d'une même couche du réseau. Pour simplifier les notations, on vectorise les images. Pour une couche l on note N_l le nombre de filtres et M_l la taille des images filtrées (features maps). On note $F^l \in R^{N_l \times M_l}$ la matrice stockant l'ensemble des features maps d'une couche : F_{jk}^l est la valeur à la position k du j -ième filtre de la couche l . Les corrélations entre les différentes features maps d'une couche l sont calculées par une matrice de Gram $G^l \in R^{N_l \times N_l}$:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

L'ensemble des matrices de Gram pour toutes les couches du réseau constituent la caractérisation des textures dans la méthode de Gatys. Mais on pourrait ne garder que les couches les plus utiles. Cette description a la qualité d'être stationnaire.

6.4 Génération de texture

Comme dit précédemment, on essaie de retrouver les bonnes statistiques de texture par descente de gradient. Plus précisément, on part d'une image de bruit blanc et on minimise les moyennes des erreurs au carré entre les matrices de Gram de l'image originale et de l'image qu'on génère. Comme notation on met un chapeau pour ce qui est généré (\hat{x}) et pas de chapeau pour ce qui concerne l'image initiale (x).

On note E_l la perte totale pour une couche l :

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - \hat{G}_{ij}^l)^2$$

La perte complète qu'on minimise utilise une pondération avec des poids w_l :

$$L(x, \hat{x}) = \sum_l w_l E_l$$

Les gradients des E_l par rapport à chaque activation d'une couche l peuvent être calculés de manière analytique :

$$\frac{\partial E_l}{\partial \hat{F}_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2} ((\hat{F}^l)^T (G^l - \hat{G}^l))_{ji} & \text{si } \hat{F}_{ij}^l > 0 \\ 0 & \text{sinon} \end{cases}$$

Ainsi on peut faire une rétropropagation de gradient sur la fonction de perte L . Dans l'article l'optimisation est faite par l'algorithme L-BFGS.

6.5 A propos d'un commentaire des auteurs

Dans l'article, les auteurs insistent sur le fait que le temps de calcul de l'algorithme dépend de la vitesse de rétropropagation de gradient, de l'algorithme d'optimisation, et que la qualité des résultats dépend de la qualité du réseau de neurones pré-entraîné utilisé. Ces domaines étant actuellement l'objet de beaucoup de recherche avec l'explosion du deep learning, les auteurs espèrent que les défauts de l'algorithme (temps de calcul et défauts visuels) pourront être beaucoup amélioré dans un futur proche. Cela semble raisonnable sauf que ça n'empêche pas, comme on le voit avec SGAN, que des méthodes beaucoup plus rapides devraient être recherchées.

7 Résultats illustrés de notre implémentation de SGAN

7.1 Texture de sable

Notre premier résultat assez satisfaisant a été obtenu à partir d'une image de sable représentée en figure 5. Cette image n'a pas de grandes structures complexes, elle est assez semblable à du bruit, ce qui facilite l'apprentissage.



Figure 5: Grande image (résolution 1600*1200*3) de sable utilisée pour prendre des patchs représentant la texture recherché

A partir de cette image on a un script Python qui génère des patchs pris à

des positions aléatoires. On a ainsi générés 1000 patchs de taille 128*128. Voir figure 6.



Figure 6: Exemples de patchs de notre base de donnée de texture de sable

Lors de l'optimisation du SGAN, nous sauvegardons toutes les 10 epochs des images obtenus avec des échantillons de bruit comme ceux utilisés à l'apprentissage (la numérotation d'epochs commence à 0). Nous générerons 4 images à partir d'un même modèle pour visualiser la variabilité des résultats. La première image est sauvegardée séparément en bonne qualité pour bien visualiser la qualité du résultat (ce que nous avons trouvé difficile dans une figure de matplotlib montrant plusieurs images).

Nous avons trouvé utile d'agrandir l'espace latent en hauteur et largeur par un facteur 3 ou 4 par rapport à l'apprentissage, pour rendre les images générées plus satisfaisantes et de meilleure résolution.

Les figures 7, 8 et 9 montrent l'évolution de la qualité des résultats au cours des epochs d'optimisation. Rappel : chacune de ses petites images utilise un espace latent agrandi pour améliorer la résolution.

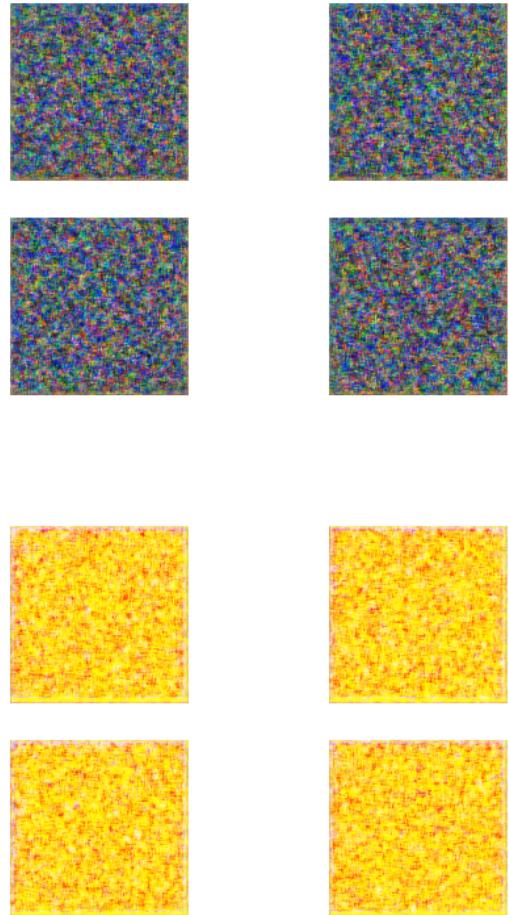


Figure 7: Exemples d'images générées par le modèle générateur à 0 (c'est à dire après 1 pas d'entraînement en fait) et 10 epochs d'entraînement

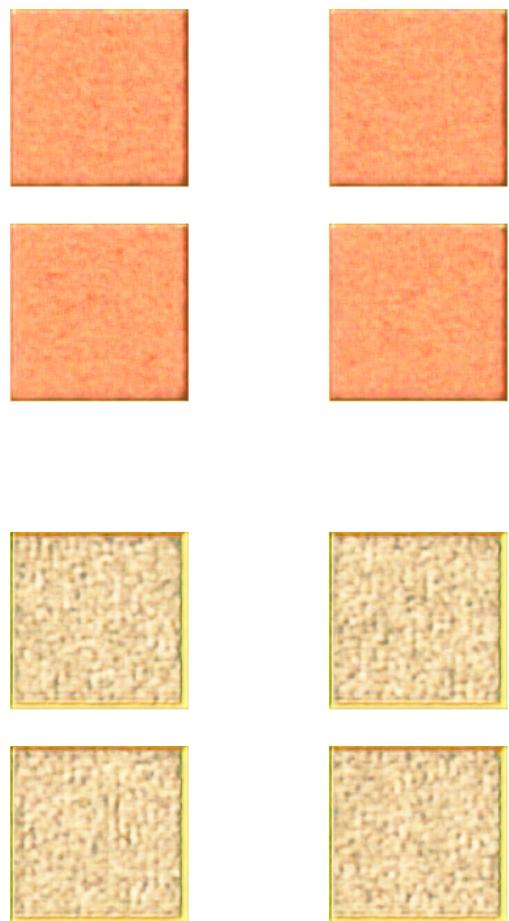


Figure 8: Exemples d'images générées par le modèle générateur à 50 et 100 epochs d'entraînement

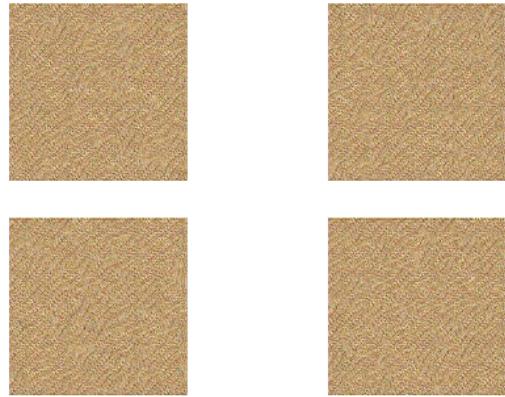


Figure 9: Exemples d'images générées par le modèle générateur à 600 epochs d'entraînement

La variabilité des résultats nous semble satisfaisante.

Les figures 10, 11 et 12 permettent de mieux visualiser la qualité des résultats obtenus. Peu après 1000 epochs l'optimisation n'a plus fonctionné : perte infinie et génération d'image uniforme.



Figure 10: Exemples d'images générées par le modèle générateur à 600 et 700 epochs d'entraînement



Figure 11: Exemples d'images générées par le modèle générateur à 800 et 900 epochs d'entraînement



Figure 12: Exemples d’images générées par le modèle générateur à 990 et 1000 epochs d’entraînement

On remarque que la technique d’agrandissement de la résolution par la structure fully convolutional et l’agrandissement d’espace latent a parfaitement fonctionné. Les couleurs produites (ainsi que les structures) évoluent rapidement au cours des epochs.

7.2 Texture de bulles et comparaison avec Gatys et TextureNet

Nous avons voulu comparer la méthode SGAN avec la méthode de Gatys décrite précédemment, sur une image du TP. L’image choisie possède des bulles qui forment des structures assez grandes et complexes. Elle est en figure 13.

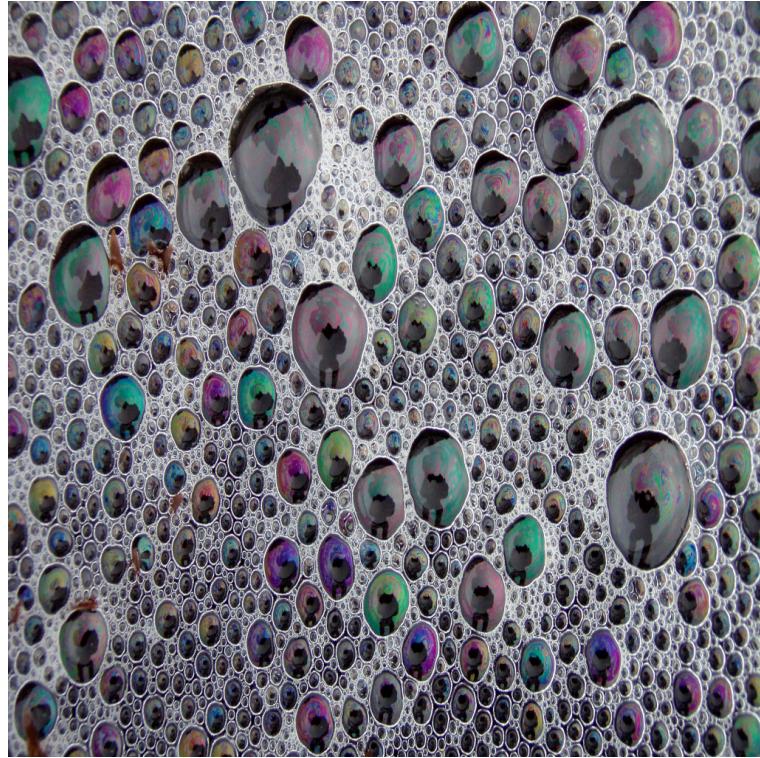


Figure 13: Image de texture de référence qu’on cherche à reproduire

Nous avons essayé plusieurs choses pour améliorer nos résultats, notamment prendre des patchs plus grands de taille 256*256, pour mieux apprendre les répétitions, et aussi réduire la taille des patchs (avec PIL) à 64*64 pour un apprentissage plus facile. Nous sommes loins d’avoir des résultats satisfaisants. Les résultats sont en figure 14

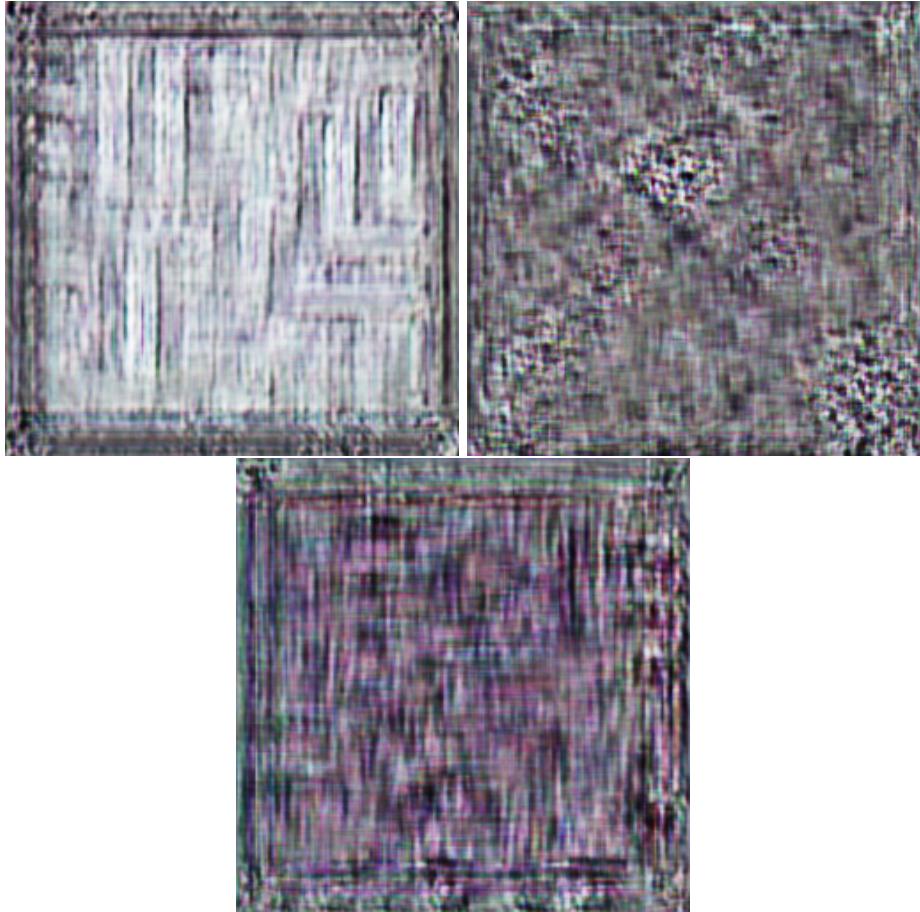


Figure 14: Différents résultats générés par notre implémentation de SGAN à partir de l'image de bulles : le premier avec des patchs de taille 256*256, les deux suivants avec redimensionnement des patchs à 64*64 (qui montrent toujours des structures de taille 256*256 dans l'image initiale) pour un apprentissage plus efficace

Les résultats obtenus par Gatys sont bien meilleurs, voir figure 15.

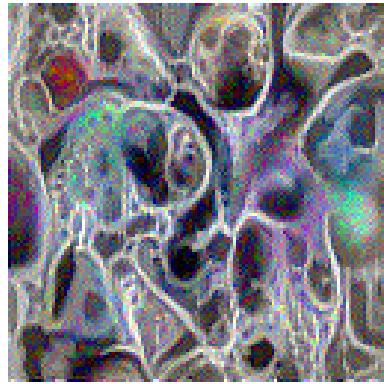


Figure 15: Un résultat par Gatys pour la texture de bulles

7.3 Comparaison avec les résultats de l'article SGAN

En fait nous pensons qu'avec un meilleur entraînement de SGAN que ce que nous avons fait, et peut-être avec l'amélioration des paramètres, SGAN devrait être bien plus performant que la méthode de Gatys en terme de qualité perçue. L'exemple de texture sur des images de la ville de Barcelone utilisé dans l'article de SGAN montre que cet algorithme s'adapte bien à des structures très complexes. Voir figure 16. SGAN bien réalisé et entraîné est censé être très réaliste.

Le code des auteurs est disponible. Cependant, le temps d'entraînement sur CPU était de 3h par epochs, et nous avons trouvé difficile de faire fonctionner les librairies nécessaires pour utiliser le GPU dans le calcul. Il semble que les auteurs de l'article présentent des résultats après 50 epochs. Nous n'avons donc pas entraîné le SGAN précis de l'article sur de nouvelles textures, nous avons seulement réussi à générer une texture à partir du modèle sur la ville de Barcelone déjà entraîné et sauvegardé.



Figure 16: D’abord une image de Barcelone utilisée pour l’entraînement, puis une texture générée par SGAN, ici périodique grâce à une répétition sur l’espace latent

8 Discussion

Concluons ici sur ce qu’apporte SGAN par rapport à la méthode de Gatys, et sur les incovénients de SGAN.

L’utilisation de GANs peut s’expliquer par la recherche de réalisme qui peut être obtenu grâce à la confrontation entre les réseaux G et D , mais ce qui a surtout été utilisé ici est l’espace latent. Ses propriétés pour l’interpolation ont déjà été évoquées mais ce n’est pas ce qui a été utilisé ici : grâce à une structure fully convolutional on peut l’étendre en largeur et en hauteur comme on veut après entraînement et produire des images avec une complexité linéaire en la résolution. L’espace latent permet de produire plus facilement des textures de toutes tailles voulues.

Une autre application peut être d’apprendre facilement une texture sur une base de données et non une seule image.

Une fois que SGAN est entrainé il est plus rapide (de plusieurs ordres de grandeurs) que Gatys pour générer une texture. Cependant l’entraînement de SGAN peut être très long (les auteurs utilisent 30 minutes avec un bon GPU), et difficile à implémenter : l’entraînement de GANs est généralement réputé assez difficile.

9 Partage des tâches et difficultés d'implémentation

Nous sommes principalement passés par deux étapes de progression :

Etape 1 :

- Etienne : A partir du TP des GANs, implémentation d'une architecture fully convolutionnal avec une couche fully-connected qui marche sur un espace latent 1D.
- Fatine : Implémentation de l'architecture de DCGAN sur Keras et test sur MNIST.

Etape 2 :

- Fatine : Implémentation de l'architecture de SGAN sur Keras en se basant sur DCGAN. Implémentation de l'espace latent 3D mais avec des valeurs fixes.
- Etienne : Création de la base de données des patchs à partir d'une texture. Généralisation des tailles de l'espace latent 3D sur l'architecture de SGAN (pour que ça donne des textures des tailles voulues).

Ensuite, on se débloquait mutuellement pour améliorer les résultats de notre algorithme (choix des paramètres, de la loss, entraînement du discriminateur ...).

Nous avons rencontré des difficultés d'entraînement du GAN tout au long de nos expérimentations. Nous avons essayé des techniques comme des labels bruités, d'autre paramètres de l'algorithme Adam. Les difficultés d'entraînement ont déjà été décrites dans ce rapport dans la section sur l'entraînement des GANs. Nos résultats finaux sont loin d'être parfaits, mais ils sont plus ou moins satisfaisants. Nous avons réussi à générer des textures de toutes tailles à partir des patchs extraits de micro-textures. Cependant, des fois, l'algorithme dysfonctionne quand on dépasse 2000 epochs.

Une erreur que nous avons repéré que très tard était que nous avons normalisées les images de la base de données entre 0 et 1 alors que l'activation Tanh à la fin du générateur produit des images entre -1 et 1.

L'utilisation de Google Colab nous permettait d'utiliser des GPUs gratuits et d'utiliser de grandes bases de données stockées sur google drive. Cependant à certains moments ou pendant certaines périodes le site plante ce qui nous a pris beaucoup de temps.

References

- [1] Nikolay Jetchev, Urs Bergmann, and Roland Vollgraf. Texture synthesis with spatial generative adversarial networks. *arXiv preprint arXiv:1611.08207*, 2016.
- [2] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- [3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [4] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *arXiv preprint arXiv:1812.04948*, 2018.
- [5] Javier Portilla and Eero P Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International journal of computer vision*, 40(1):49–70, 2000.
- [6] Alexander Ecker Leon Gatys and Matthias Bethge. Texture synthesis using convolutional neural networks. *Advances in Neural Information Processing Systems 28*, 2015.
- [7] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, volume 1, page 4, 2016.
- [8] Lucas Theis and Matthias Bethge. Generative image modeling using spatial lstms. In *Advances in Neural Information Processing Systems*, pages 1927–1935, 2015.
- [9] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.
- [10] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.