Analyse de l'article "Texture Synthesis Using Convolutional Neural Networks"

Par Rida Lefdali, Sandy Frank Kwamou Ngaha et Cristiano Ulondu Mendes

20 janvier 2022

1 Introduction

La synthèse de texture est l'acte de déduire un processus de génération à partir d'un exemple de texture qui permet de produire de nouveaux échantillons arbitraires de cette texture. Le seul outil permettant de mesurer la qualité de la texture synthétisée est la perception humaine. Nous disons qu'une texture synthétisée est de bonne qualité si nous ne pouvons pas faire la différence entre elle et l'image originale.

Il existe des méthodes pour effectuer une telle tâche. La première approche est une approche non-paramétrique qui est basée sur la génération de la nouvelle texture en utilisant des techniques de rééchantillonnage sur les pixels. La deuxième approche est une approche paramétrique basée sur un ensemble de mesures statistiques prises sur l'étendue spatiale de l'image.

Comme nous le savons, le réseau neuronal a prouvé sa capacité à effectuer de nombreuses tâches avec une grande précision. Par exemple, il a été démontré que le CNN a une bonne précision dans la classification d'images. Cet article propose donc une nouvelle méthode de synthèse de texture basée sur le réseau de neurones convolutifs, en particulier le réseau de neurones VGG-19 qui a été formé à la reconnaissance d'objets. Cette nouvelle méthode vise à combiner le cadre conceptuel des statistiques de résumé spatial sur les réponses aux caractéristiques avec l'espace caractéristique (feature space) du réseau VGG-19.

2 Méthodes classiques d'analyse de texture

Il existe deux principales classes de méthodes de synthèse de textures :

2.1 Méthodes non paramétriques :

Il s'agit ici notamment des algorithmes de synthèses de textures par patchs. Dans cette approche, on génère des nouvelles textures par échantillonnage des pixels ou encore des patchs de la texture originale. Autrement dit l'algorithme calcul séquentiellement une texture en sortie telle que chaque patch de la sortie corresponde à un patch de la texture d'entrée. La texture de sortie est remplie avec des pixels(ou patches) aléatoires de l'image d'entrée, les transitions entre pixels(ou patches) sont gérés par optimisation. L'avantage de cette méthode est qu'il synthétise bien les macro-textures. Son désavantage est qu'il peut s'avérer lent et peut présenter des problèmes de stabilité, difficulté à régler les paramètres...

2.2 Méthodes paramétriques :

Il s'agit ici des algorithmes de synthèses de textures en se basant sur des contraintes statistiques. Extraction de "statistiques" pertinentes de l'image d'entrée (e.x. distribution des couleurs, des coefficients de Fourier, des coefficients d'ondelette,. . .). Calcul d'une image de sortie "aléatoire", ayant les mêmes statistiques : partir d'une image de bruit blanc et imposer alternativement chaque "statistique" de l'image d'entrée. L'avantage de cette méthode est qu'il est perceptuellement stable. Son désavantage est généralement pas assez bon pour les macro-textures.

3 La méthode de Portilla-Simoncelli

L'algorithme de texture de Portilla et Simoncelli (PS) est l'une de ces méthodes qui est devenue très populaire et a influencé les études de perception visuelle. Pour de nombreuses raisons, il peut encore être considéré comme un algorithme de synthèse de texture de pointe : (i) il génère des textures qui sont souvent impossibles à distinguer de l'original sans examen ; (ii) il repose sur peu de paramètres par rapport aux méthodes récentes deep learning ; (iii) les algorithmes récents s'y comparent souvent.

Brièvement, l'algorithme PS synthétise une nouvelle texture en imposant itérativement à une image de bruit blanc gaussien un ensemble de statistiques d'ordre élevé de coefficients d'ondelettes pré-calculés sur un exemple de texture. Après quelques itérations, l'image de bruit blanc initiale est transformée en une texture similaire à la texture originale.

L'algorithme PS se compose de deux étapes décrites comme suit

3.1 Étape 1 : Analyse

— Décomposition et reconstruction d'images : On construit les filtres pyramidaux orientables de manière à permettre de décomposer une image en sous-bandes et de reconstruire une image à partir de sous-bandes. En d'autres termes, les filtres ont des propriétés qui font de la décomposition en pyramide orientable un opérateur pseudo-inversible.

— Calculez la décomposition pyramidale orientable complexe de la texture original à partir duquel de nouveaux échantillons de texture seront générés.

— Calculer le résumé statistique des sous-bandes de la pyramide.

3.1.1 Comment les contraintes statistiques sont obtenues.

Afin de trouver les contraintes statistiques pertinentes, Portilla et Simoncelli ont utilisé une méthode empirique basée sur un large jeu de données de texture composé de textures artificielles et naturelles couvrant la plupart des caractéristiques trouvées dans les textures. En ne considérant que les statistiques marginales et croisées, ils ont ajouté différentes contraintes statistiques sur les sous-bandes une à une tout en testant l'algorithme sur jeu de données de texture. Après ajout d'une contrainte, les résultats ont été subjectivement classés selon le type de fonctionnalités manquantes. Une nouvelle contrainte statistique a ensuite été choisie si les caractéristiques étaient correctement synthétisé ou non. Enfin, chaque contrainte a été vérifiée comme étant nécessaire en exhibant échecs de synthèse. Les contraintes superflues ont été supprimées.

3.2 Étape 2 : Synthèse

La texture de sortie est synthétisée à l'aide du schéma itératif suivant.

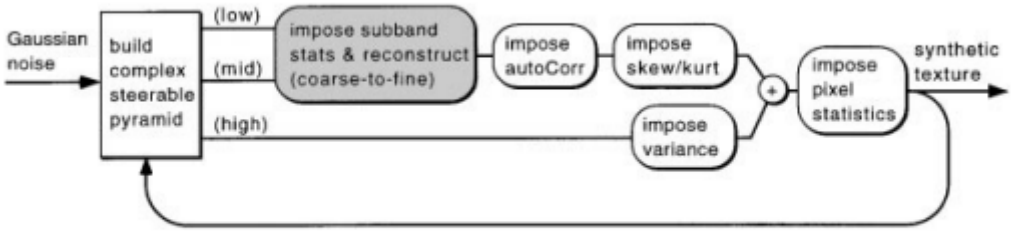

Figure 9. Top level block diagram of recursive texture synthesis algorithm. See text.

— Le processus est initialisé avec une image contenant des échantillons de bruit blanc gaussien.

— L'image est décomposée en une pyramide orientable complexe.

— Une procédure récursive grossière à fine impose les contraintes statistiques sur les sous-bandes passe-bas et passe-bande, tout en reconstruisant simultanément une image passe-bas.

— L'auto-corrélation de l'image passe-bas reconstruit est ensuite ajustée, ainsi que le biais et l'aplatissement, et le résultat est ajouté à la bande passe-haut ajustée en variance pour obtenir l'image de texture synthétisée.

— Les statistiques marginales sont imposées aux pixels de cette image et tout le processus est répété.

— La convergence est atteinte après environ 50 itérations, pour les centaines de textures que les auteurs ont synthétisés.

— De plus, une fois la convergence atteinte (à une certaine tolérance près), les textures synthétiques sont assez stables, oscillant peu dans leurs paramètres.

4 Réseau de neurones convolutif : VGG-19

A la différence des méthodes présentées précédemment, les auteurs de cet article proposent un modèle de synthèse de texture utilisant un réseau de neurones convolutif. Dans cette partie nous commencerons par décrire brièvement l'architecture générale de ces réseaux de neurones puis nous nous pencherons sur les particularités du réseau choisi et son utilisation.

4.1 Architecture d'un réseau de neurones convolutif

Un réseau de neurone artificiel est une modélisation mathématique simplifié du fonctionnement des neurones biologiques composé de plusieurs couches de noeuds connectés. Un réseau de neurones convolutif est composé de trois types de couches : des couches de convolutions, des couches de pooling et des couches entièrement connectées. Le premier et le dernier type de couche sont généralement couplés avec une fonction d'activation non linéraire.

Couche de convolution
Avant de détailler le fonctionnement de ce type de couche commençons par définir formellement la convolution.

Définition 4.1 (Convolution 2D périodique) *Le produit de convolution de deux images u et v de $\mathbb{C}^{M\times N}$ est l'image $u * v \in \mathbb{C}^{M\times N}$ définie pour tout $(m,n) \in 0,\dots,M-1 \times 0,\dots,N-1$ par*

$$(u*v)_{m,n} = \sum_{p=0}^{M-1}\sum_{q=0}^{N-1} u_{p,q} v_{(m-p)\ mod\ M,(n-q)\ mod\ N}$$

Dans une couche de convolution le volume d'entrée (une image ou la sortie d'une autre couche) de dimension $N \times N \times k$ (longueur×largeur×profondeur) est convoluer avec différents filtres de même dimension, aussi appelés noyaux de convolution, avec une dimension spatiale $F \times F$ petite et une profondeur égale à k. Le filtre ayant (toujours) une dimension spatiale plus petite que celle du volume d'entrée, la convolution entre ces deux objets se fait d'une manière particulière que l'on peut visualiser de la façon suivante : on superpose le filtre K sur le coin gauche du volume d'entrée V ensuite on multiplie chaque nombre superposé canal par canal puis on additionne le tout et on recommence en déplaçant K d'un certain rang jusqu'à ce qu'on ait parcouru tout le volume. Le résultat d'une telle convolution est une matrice carrée (2D), appelé feature map, dont les dimensions sont contrôlées par trois hyperparamètres :

— La dimension F du filtre.

— Le pas de déplacement horizontal et vertical du filtre souvent appelé Stride.

— La marge p ou zero-padding qui correpond au nombre de lignes/colonnes de 0 rajoutées à la frontière des canaux du volume d'entrée avant la convolution avec le filtre.

La longueur du feature map est alors donnée par la formule suivante : $\frac{N-F+2p}{Stride}+1$. Enfin, la sortie de cette couche est un empilement des différentes feature maps auxquelles on a appliqué coordonnées par coordonnées une fonction d'activation non linéaire. La figure 1 résume les étapes décrites ci-dessus.

Couche de pooling

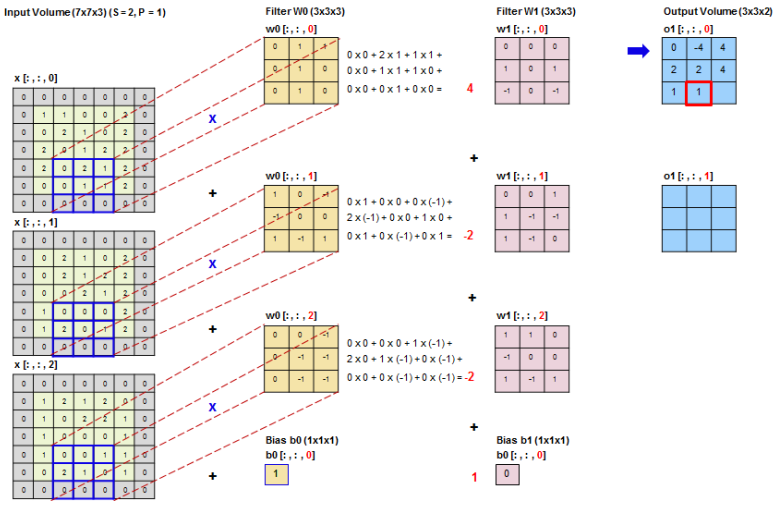

FIGURE 1 – Exemple de convolution d'un volume de profondeur 3 par deux filtres $3 \times 3 \times 3$.

Une couche de pooling souvent placée entre deux couches de convolution permet de réduire la dimension spatiale du volume d'entrée V (sortie d'une couche de convolution) et ainsi diminuer le temps de calcul dans les couches plus en profondeur, on verra plus tard que cette couche a d'autres avantages. Le pooling consiste à subdiviser chaque canal (feature map) de V en un ensemble de carrés de pixels de même taille et qui ne se chevauchent pas puis à compresser les données de chaque carré en un pixel en appliquant une certaine opération. On peut donner deux exemples d'opérations fréquemment utilisées :

— Max-pooling : consiste à sélectionner la valeur maximale des pixels du carré.

— Average-pooling : consiste à faire la moyenne des pixels du carré.

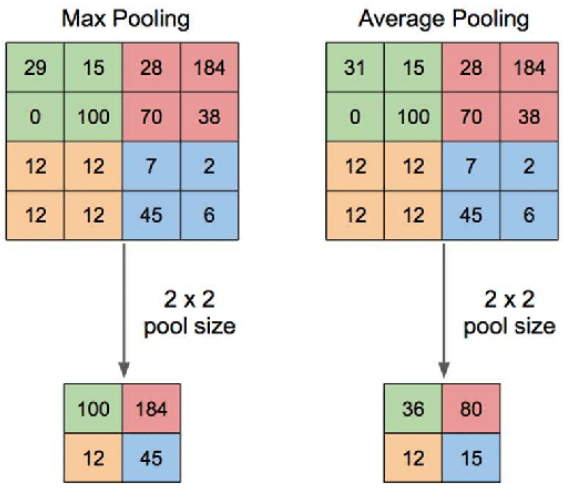

FIGURE 2 – Deux exemples d'opérations de pooling.

Comme annoncé, la couche de pooling a des avantages autres que la réduction du temps de calcul, en effet, on peut imaginer que les plus grandes valeurs d'un feature map correspondent aux caractéristiques les plus importantes détectées et donc en appliquant une opération de max-pooling on réduit la dimension tout en préservant ces caractéristiques ce qui permet de baisser le risque de sur-apprentissage. On peut aussi remarquer que l'opération de max-pooling permet de sélectionner les plus grandes valeurs indépendamment de leur position dans les carrés (voir fig.1) ce qui rend le réseau moins sensible aux transformations comme les translations et les rotations et ainsi améliore son efficacité.

Couche entièrement connectée

Les couches entièrement connectées constitue les dernières couches d'un réseau de neurones convolutif, elles correspondent aux couches que l'on retrouve dans les réseaux de neurones non-convolutifs.

Une telle couche prend en entrée un vecteur, obtenu en redimensionnant le volume de sortie de la couche qui la précède (flattening) s'il s'agit d'une couche de convolution ou de pooling, et lui applique une application affine puis éventuellement une fonction d'activation. La sortie est un autre vecteur de longueur égale au nombre de neurones composant la couche suivante.
Pour leur modèle les auteurs de l'article ont choisi le réseau de neurones VGG-19 que nous présentons ci-dessous.

4.2 Le réseau de neurones convolutif VGG-19

Le réseau de neurone convolutif VGG-19 a été construit et entraîné pour la détection et la classification d'objets dans une image. Les chercheurs à l'origine de ce réseau ont construit et testé plusieurs réseaux de neurones convolutifs de profondeur différente dans l'objectif d'étudier l'influence de la profondeur (nombre de couches de convolution) sur le taux d'erreur d'un réseau. VGG-19 est le plus profond parmi ceux construits et a montré de meilleures performances ce qui leurs a valu la première place dans la catégorie localisation et la deuxième place dans la catégorie classification du coucours ILSVRC (ImageNet Large Scale Visual Recognition Competition) en 2014.

4.3 Architecture du réseau VGG-19

Définissons d'abord la fonction ReLU et la fonction softmax qui sont utilisées dans les différentes couches du réseau VGG-19.

Définition 4.2 (fonction ReLU) *La fonction ReLU (Rectified Linear Unit) est une fonction d'activation non-linéaire R définie par*

$$\begin{array}{rccc} R: & \mathbb{R} & \to & \mathbb{R}^+ \\ & x & \longmapsto & max(0,x) \end{array}$$

C'est la fonction d'activation la plus utilisée dans les réseaux de neurones convolutifs du fait de la simplicité de sa dérivée, qui vaut 1 sur $\mathbb{R}_+^*$ et 0 sinon (on accepte en général la valeur 0 comme dérivée au point 0 même si la fonction n'est pas dérivable en ce point) , ce qui réduit le temps d'entraînement par rapport aux autres fonctions d'activations (tanh par exemple).

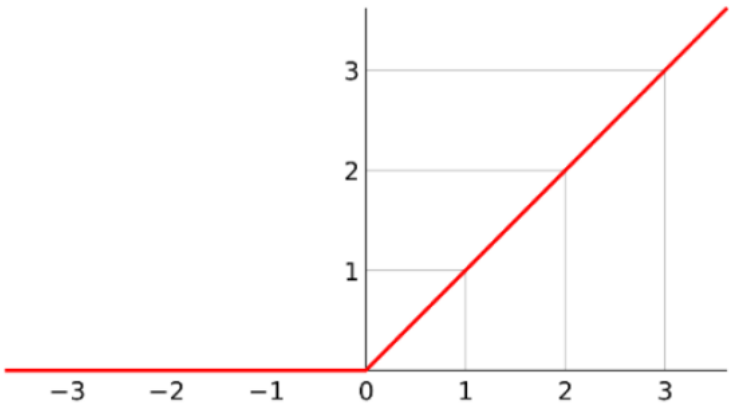

FIGURE 3 – Représentation graphique de la fonction ReLU.

Définition 4.3 (fonction softmax) *La fonction softmax est la fonction σ définie par*

$$\begin{array}{rccc} \sigma: & \mathbb{R}^K & \to & \mathbb{R}^K \\ & x=(x_1,\ldots,x_K) & \longmapsto & (\sigma(x)_1,\ldots,\sigma(x)_K)=(\frac{e^{x_1}}{\sum_{j=1}^{K} e^{x_j}},\ldots,\frac{e^{x_K}}{\sum_{j=1}^{K} e^{x_j}}) \end{array}$$

Elle transforme un vecteur de K réels en un vecteur de K réels compris entre 0 et 1 et dont la somme fait 1. Elle est utilisée dans la dernière couche des réseaux de neurones convolutifs spécialisés dans

les taches de classification et de détection d'objet, auquel cas K correspond au nombre de classes et $\sigma(x)_i$ est interprété comme la probabilité que l'objet de classe i soit présent dans l'image.

Le réseau VGG-19 est composé de 16 couches de convolution, 5 couches de pooling, 3 couches entièrement connectées et une couche softmax. Les hyperparamètres contrôlant la dimension spatiale des volumes de sortie sont les mêmes dans toutes les couches de convolution avec $F = 3$ (filtres de dimension 3×3), un pas de déplacement des filtres égale à 1 et une marge qui vaut 1 également. De cette façon la dimension spatiale du volume de sortie d'une couche de convolution est la même que celle du volume d'entrée. Dans toutes les couches de pooling, un max-pooling est opérer sur des carrés 2×2 qui ne se superposent pas (sur chaque canal du volume d'entrée) ce qui produit en sortie un volume dont la dimension spatial est réduit de moitié par rapport au volume d'entrée.
Les deux premières couches du réseau sont des couches de convolution composées de 64 filtres chacune, elles sont suivies d'une couche de max-pooling. Ensuite, on a une alternance entre des couches de convolution et des couches de max-pooling avec un nombre de filtres qui est doublé après chaque couche de pooling pour arriver à 512 dans la dernière couche de convolution. Puis on a les trois couches entièrement connectées dont les deux premières produisent en sortie des vecteurs de même longueur et la dernière un vecteur de longueur 1000 ce qui correspond au nombre de classe considérer dans le concours ILSVRC et enfin on a la couche softmax. La figure 4 schématise

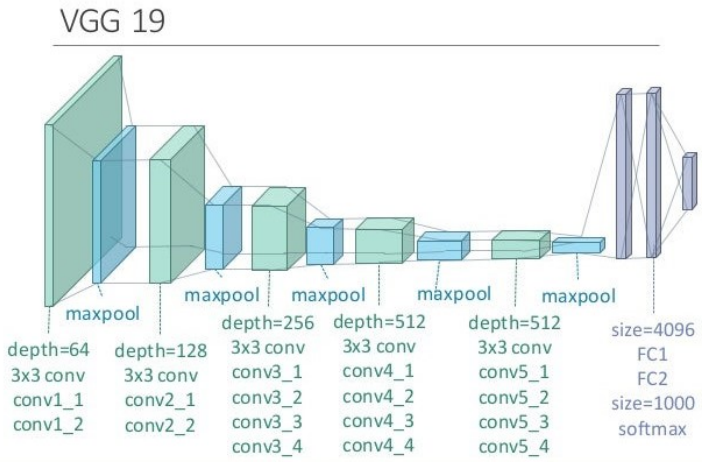

FIGURE 4 – Schéma de l'architecture du réseau de neurones convolutif VGG-19.

l'architecture décrite ci-dessus et permet d'y voir un peu plus clair. Pour plus d'informations sur ce réseau vous pouvez consulter l'article original ([SZ15]).

4.4 Transfer learning

En général, on a besoin d'une énorme base de données pour entraîner un réseau de neurones convolutif bout à bout et cette phase d'entraînement peut prendre des jours voir des semaines. Le transfer learning consiste à utiliser un réseau de neurones pré-entraîné, pour une tâche spécifique, pour accomplir une nouvelle tâche. En pratique, les poids appris dans les premières couches et les couches centrales sont figés et seules quelques-unes des dernières couches sont ré-entraînées (il existe d'autres stratégies expliquées plus en détail dans [Uni21]). Cette méthode permet de gagner beaucoup de temps pendant la phase d'entraînement et avec peu de données le réseau final montre parfois de meilleure performance par rapport à un autre réseau entraîner bout à bout pour la même tâche ([Gug21]). On a pu observer ces avantages dans le tp sur le transfer learning avec ResNet où on a construit un réseau de neurones pour classifier des images de fourmis et de d'abeilles à partir du réseau ResNet qui prend en compte 1000 classes différentes.
Ce que les auteurs de l'article qu'on étudie ont fait peut être qualifié de transfer learning (même si les dernières couches, les couches entièrement connectées, sont supprimées et non ré-entraînées) car ils ont réutilisé les poids appris dans les 16 couches de convolution du réseau VGG-19 pour construire leur modèle de synthèse de texture. Ils ont également remplacé les couches de max-pooling par des couches de average-pooling mais seulement parce que cela permet d'obtenir des images plus nettes (voir fig. 9).

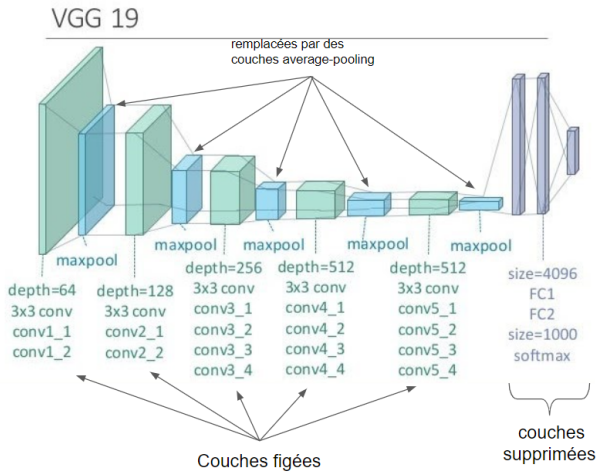

FIGURE 5 – Schéma des modifications apportées au réseau de neurones convolutif VGG-19.

5 Modèle de synthèse

Définissons la composante principale du modèle de synthèse : la matrice de Gram.

Définition 5.1 (Matrice de Gram) *Soient $v_1, \dots, v_m$ des vecteurs de $\mathbb{R}^n$, leur matrice de Gram est la matrice $G \in \mathbb{R}^{m \times m}$ telle que*

$$G_{i,j} = \langle v_i, v_j \rangle, \forall (i,j) \in [\![1, m]\!]^2$$

où $\langle \cdot, \cdot \rangle$ désigne le produit scalaire canonique de $\mathbb{R}^n$.
Si les vecteurs $v_1, \dots, v_m$ sont les lignes d'une matrice X, alors $G = XX^T$.

Dans leur modèle, les auteurs caractérisent une texture par un ensemble de matrices de Gram calculées sur l'espace des feature maps après passage de la texture dans le réseau. Soit $\vec{x}$ une texture vectorisée, elle est passé en entrée au réseau de neurones et ils récupèrent les volumes de sortie des couches de convolution. Pour une couche de convolution l avec N_l filtres, les N_l feature maps de longueur M_l une fois vectorisée sont stockées dans une matrice $F^l \in \mathbb{R}^{N_l \times M_l}$, où ${F^l}_{j,k}$ est le résultat de la convolution avec le $j^{\text{ième}}$ filtre (+ReLU) à la position k dans la couche l. La matrice de Gram $G^l \in \mathbb{R}^{N_l \times N_l}$ correspondante est alors donnée par $G^l = F^l {F^l}^T$, il s'agit de la corrélation entre les différentes feature maps de la couche l. La texture $\vec{x}$ est entièrement caractérisée par ses matrices de Gram et donc synthétiser la même texture revient à trouver une image qui produit les mêmes matrices de Gram que $\vec{x}$ quand elle passe à travers le réseau de neurones. Nous verrons dans la partie suivante comment les auteurs ont procédé.

6 Synthèse de texture

Pour générer une nouvelle texture à partir d'une image de texture initiale $\vec{x}$, les auteurs font passer $\vec{x}$ à travers le réseau puis calculent ses matrices de Gram. Ensuite, ils font de même avec une image de bruit blanc $\hat{\vec{x}}$. Après, ils font une descente de gradient sur l'image $\hat{\vec{x}}$ pour minimiser la distance entre les matrices de Gram de $\vec{x}$ et celles de l'image en cours de synthèse. Cette distance est calculée séparément dans chaque couche de convolution l, elle est définie par

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

où G^l et $\hat{G}^l$ sont les matrices de Gram respectives de $\vec{x}$ et $\hat{\vec{x}}$ dans la couche l. La fonction de perte globale est donnée par

$$\mathcal{L}(\vec{x}, \hat{\vec{x}}) = \sum_{l=1}^{16} w_l E_l$$

où w_l est le poids associé à la couche l.
La descente de gradient nécessite de calculer les différentielles de E_l (pour chaque couche l) et

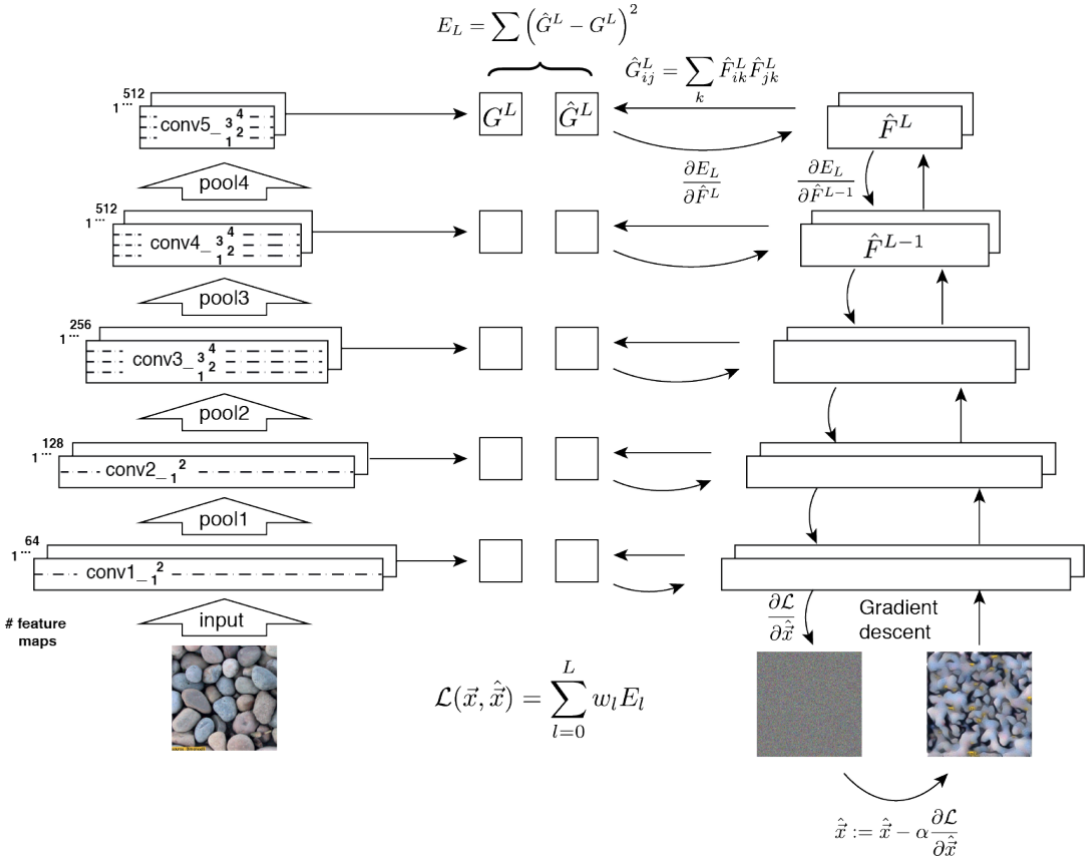

FIGURE 6 – Étapes de la synthèse de texture .

de $\mathcal{L}(\vec{x}, \hat{\vec{x}})$ par rapport à $\hat{\vec{x}}$ et les auteurs soulignent qu'une partie de ces différentielles peut être calculée analytiquement. En effet, la dérivée partielle de E_l par rapport à $\hat{F}^l_{p,q}$ (quantité positive ou nulle à cause de la fonction ReLU) est donnée par l'expression suivante :

$$\frac{\partial E_l}{\partial \hat{F}^l_{p,q}} = \begin{cases} \frac{1}{N_l^2 M_l^2}\left((\hat{F}^l)^T(\hat{G}^l - G^l)\right)_{q,p} & \text{si } \hat{F}^l_{p,q} > 0 \\ 0 & \text{si } \hat{F}^l_{p,q} = 0. \end{cases}$$

Preuve
soient $l \in [\![1, 16]\!]$ et $(p, q) \in N_l \times M_l$. Si $\hat{F}^l_{p,q} > 0$,

$$\begin{aligned}
\frac{\partial E_l}{\partial \hat{F}^l_{p,q}} &= \frac{1}{4N_l^2 M_l^2} \frac{\partial}{\partial \hat{F}^l_{p,q}} \left[\sum_{i=1}^{N_l} \sum_{j=1}^{N_l} (G^l_{i,j} - \hat{G}^l_{i,j})^2 \right] \\
&= \frac{1}{4N_l^2 M_l^2} \frac{\partial}{\partial \hat{F}^l_{p,q}} \left[\sum_i (G^l_{i,i} - \hat{G}^l_{i,i})^2 + \sum_i \sum_{j \neq i} (G^l_{i,j} - \hat{G}^l_{i,j})^2 \right] \\
&= \frac{1}{4N_l^2 M_l^2} \frac{\partial}{\partial \hat{F}^l_{p,q}} \left[\sum_i (G^l_{i,i} - \hat{G}^l_{i,i})^2 + 2 \sum_i \sum_{j<i} (G^l_{i,j} - \hat{G}^l_{i,j})^2 \right] \qquad \text{car } G^l \text{ et } \hat{G}^l \text{ sont symétriques.} \\
&= \frac{1}{4N_l^2 M_l^2} \left(\underbrace{\frac{\partial}{\partial \hat{F}^l_{p,q}} \left[\sum_i (G^l_{i,i} - \hat{G}^l_{i,i})^2 \right]}_{A} + 2 \underbrace{\frac{\partial}{\partial \hat{F}^l_{p,q}} \left[\sum_i \sum_{j<i} (G^l_{i,j} - \hat{G}^l_{i,j})^2 \right]}_{B} \right)
\end{aligned}$$

$$
\begin{aligned}
A &= \sum_i \frac{\partial}{\partial \hat{F}^l_{p,q}} \left[(G^l_{i,i} - \hat{G}^l_{i,i})^2 \right] \\
&= \frac{\partial}{\partial \hat{F}^l_{p,q}} \left[(G^l_{p,p} - \hat{G}^l_{p,p})^2 \right] && \text{car } \hat{G}^l_{i,i} = \sum_{k=1}^{N_l} \hat{F}^l_{i,k} \hat{F}^l_{i,k} = \sum_{k=1}^{N_l} (\hat{F}^l_{i,k})^2 \text{ et donc } \hat{F}^l_{p,q} \text{ n'apparaît que dans } \hat{G}^l_{p,p}. \\
&= \frac{\partial}{\partial \hat{G}^l_{p,p}} \left[(G^l_{p,p} - \hat{G}^l_{p,p})^2 \right] \frac{\partial}{\partial \hat{F}^l_{p,q}} \left[\hat{G}^l_{p,p} \right] && \text{Chain-rule.} \\
&= 4(\hat{G}^l_{p,p} - G^l_{p,p}) \hat{F}^l_{p,q}
\end{aligned}
$$

$$
\begin{aligned}
B &= \sum_i \sum_{j<i} \frac{\partial}{\partial \hat{F}^l_{p,q}} \left[(G^l_{i,j} - \hat{G}^l_{i,j})^2 \right] \\
&= \sum_{j<p} \frac{\partial}{\partial \hat{F}^l_{p,q}} \left[(G^l_{p,j} - \hat{G}^l_{p,j})^2 \right] + \sum_{i>p} \frac{\partial}{\partial \hat{F}^l_{p,q}} \left[(G^l_{i,p} - \hat{G}^l_{i,p})^2 \right] && \text{car } \hat{G}^l_{i,j} = \sum_{k=1}^{N_l} \hat{F}^l_{i,k} \hat{F}^l_{j,k} \text{ et donc } \hat{F}^l_{p,q} \text{ apparaît dans } \hat{G}^l_{i,j} \\
&\quad \text{que si } i = p \text{ ou } j = p. \\
&= \sum_{i \neq p} \frac{\partial}{\partial \hat{F}^l_{p,q}} \left[(G^l_{i,p} - \hat{G}^l_{i,p})^2 \right] && \text{car } G^l \text{ et } \hat{G}^l \text{ sont symétriques.} \\
&= \sum_{i \neq p} \frac{\partial}{\partial \hat{G}^l_{i,p}} \left[(G^l_{i,p} - \hat{G}^l_{i,p})^2 \right] \frac{\partial}{\partial \hat{F}^l_{p,q}} \left[(\hat{G}^l_{i,p} - G^l_{i,p}) \right] && \text{Chain-rule.} \\
&= \sum_{i \neq p} 2(\hat{G}^l_{i,p} - G^l_{i,p}) \hat{F}^l_{i,q}
\end{aligned}
$$

Et finalement,

$$
\begin{aligned}
\frac{\partial E_l}{\partial \hat{F}^l_{p,q}} &= \frac{1}{4 N_l^2 M_l^2} \left(4(\hat{G}^l_{p,p} - G^l_{p,p}) \hat{F}^l_{p,q} + 4 \sum_{i \neq p} (\hat{G}^l_{i,p} - G^l_{i,p}) \hat{F}^l_{i,q} \right) \\
&= \frac{1}{N_l^2 M_l^2} \sum_i (\hat{G}^l_{i,p} - G^l_{i,p}) \hat{F}^l_{i,q} \\
&= \frac{1}{N_l^2 M_l^2} \sum_i (\hat{F}^{lT})_{q,i} (\hat{G}^l - G^l)_{i,p} \\
&= \frac{1}{N_l^2 M_l^2} \left((\hat{F}^l)^T (\hat{G}^l - G^l) \right)_{q,p}
\end{aligned}
$$

□

En pratique, les gradients sont calculés en utilisant le procédé de back-propagation qui est le procédé utilisé pendant la phase d'entraînement des réseaux de neurones pour mettre à jour les poids des différentes couches.

7 Implémentation et résultats

7.1 Implémentation

Pour implémenter et tester la méthode de l'article pour la synthèse de texture, nous avons décidé d'utiliser un autre code basé sur le framework pytorch car le code fourni est basé sur le framework caffe et nous avons rencontré beaucoup de problèmes pour l'installer car le framework utilise certaines dépendances qui n'existent plus.

L'implémentation de la méthode a ensuite été faite en utilisant le framework Pytorch. Nous vous invitons à lire le jupyter notebook qui est très détaillé.

7.2 Résultats

Dans un premier temps, nous allons voir l'influence du nombre de couches utilisées dans le *VGG-19* sur les textures générées. Cela signifie que nous allons voir la texture générée dans chaque couche à partir de la couche **conv1_1** jusqu'à la couche **pool4** en utilisant *LBFGS* comme optimiseur.

Nous pouvons remarquer, dans la figure 7 que lorsque nous augmentons le nombre de couches, nous avons une texture générée qui est indiscernable de l'image originale. Mais quand on utilise seulement la couche basse (**conv1_1**) on obtient des images qui ne contiennent pas les structures de l'image originale, et qui sont proches du bruit. Donc, l'utilisation de plusieurs couche de convolution et de pooling permet d'améliorer la qualité de la texture synthétisé.

De même, si nous utilisons une image originale qui n'est pas proche de la texture dans sa structure, comme l'image du chat, nous pouvons remarquer que lorsque nous augmentons le nombre de couches, la méthode préserve les informations locales de l'image, mais elle les réorganise,ce qui signifie qu'elle ne conserve pas l'ordre spatial des informations.

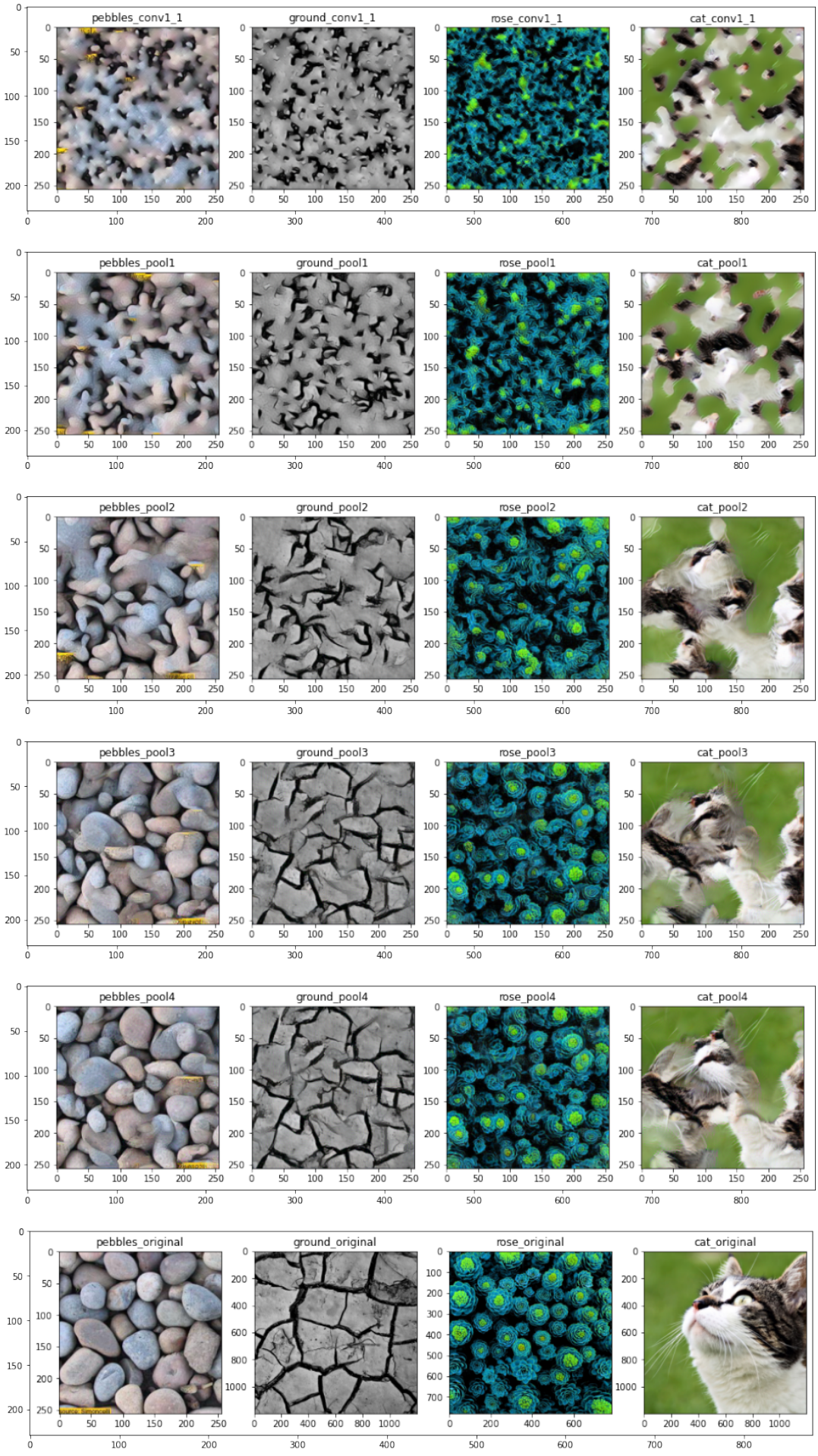

FIGURE 7 – Résultats obtenus en augmentant le nombre de couche.

Maintenant nous allons voir l'influence de l'optimiseur sur le résultat de chaque couche. Nous allons essayer l'optimiseur *Adam* sur l'image *pebbles*. Nous pouvons dire que l'utilisation de l'optimiseur *Adam* n'a pas permet d'améliorer les résultats, car on peut faire la différence entre la texture synthétisé et l'image originale. Donc, l'otpimiseur a bien une influence sur les résultats obtenues et l'optimiseur *LBFGS*, utilisé précédemment, semble de donner des résultats améliorés

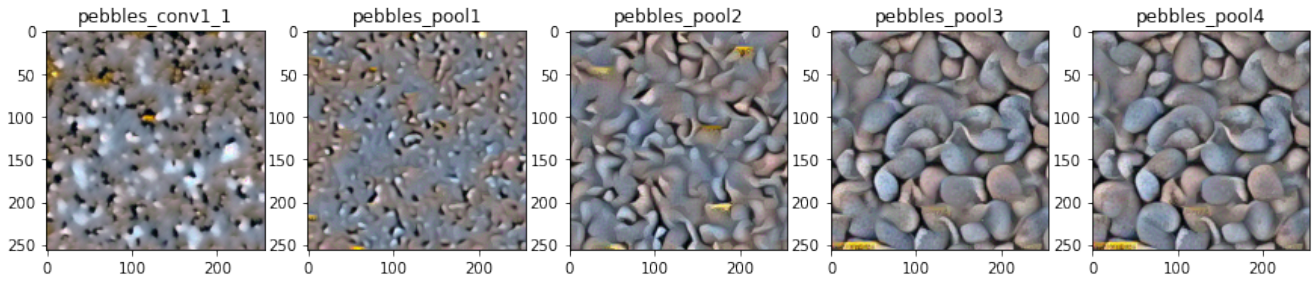

FIGURE 8 – Résultats obtenus en utilisant Adam comme optimiseur.

Nous avons essayé de changer le nombre d'itérations (Epoch) en l'augmentant et de voir l'influence, mais cela n'a apporté aucune amélioration, donc utiliser seulement 2000 itérations semble être suffisant pour générer de bonnes textures synthétisées.

Comme mentionné dans l'article, le problème derrière la perte de qualité des textures synthétisées est le nombre de paramètres du modèle. En utilisant uniquement la couche **conv1_1** et en regroupant les couches de 1 à 4, on obtient un modèle avec environ $177k$ de paramètres à estimer. Ce grand nombre de paramètres peut conduire à une perte de qualité des textures. Nous pouvons le voir sur les textures synthétisées obtenues dans la figure 7, qui n'ont pas une si bonne qualité. La méthode proposée pour résoudre ce problème est de réduire le nombre de paramètres en faisant une ACP du vecteur caractéristique dans les différentes couches du réseau et en construisant ensuite la matrice de Gram uniquement pour les k premières composantes principales. La figure suivante (figure 9 tirée de l'article) montre que l'ACP améliore les résultats (première et deuxième images).

FIGURE 9 – Résultat pris en augmentant de l'article montre l'influence de l'ACP.

8 Conclusion

La nouvelle méthode paramétrique, qui repose sur le réseau neuronal *VGG-19*, peut générer une texture synthétisée de bonne qualité. Mais la méthode est coûteuse en termes de calcul. La méthode est capable de produire une nouvelle texture en calculant la matrice de *Gram* sur les cartes de caractéristiques (la sortie des couches convolutionnelles de *VGG-19*) pour l'image originale et l'image en cours de synthèse, et ensuite, la fonction de perte à minimiser est la distance quadratique normalisée entre ces deux matrices de *Gram*.

La nouvelle méthode semble générer des textures synthétisées de bonne qualité, notamment en utilisant l'optimiseur **LBFGS** au lieu d'**Adam**. Dans l'article, les auteurs ont montré que faire une ACP pour réduire le nombre de paramètres à estimer, qui est d'environ 177k paramètres si seulement les couches **Conv1_1** et **pool_1** à **pool_4** ont été utilisées. Le grand nombre de paramètres du modèle est responsable de la perte de qualité de la texture synthétisée.

Références

[Gug21] Sylvain Gugger. What is transfer learning, why and when is it useful ? https://huggingface.co/course/chapter1/4?fw=pt, 2021.

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2015.

[Uni21] Stanford University. Convolutional neural networks for visual recognition. https://cs231n.github.io/transfer-learning/, 2021.