

Imagerie Numérique - Synthèse de textures par automates cellulaires

Arthur Calvi
ENS Paris-Saclay

arthur.calvi@ens-paris-saclay.fr

Théophile Rageau
ENS Paris-Saclay

theophile.rageau@ens-paris-saclay.fr

Abstract

Dans ce projet, nous nous intéressons à la méthode de génération de textures par automate cellulaire neuronal décrite dans l'article [6]. Nous allons présenter l'approche générale utilisée par les auteurs puis traiter de plusieurs expériences que nous avons réalisées à partir de cette méthode. Cela nous permettra de comprendre un peu mieux son fonctionnement, son potentiel ainsi que ses limites. Nous verrons en particulier qu'elle est relativement robuste à certains changements (variation des hyperparamètres du modèle ou de l'architecture) et qu'elle dispose d'un potentiel génératif assez important (capacité à s'approprier des informations contenues dans des images et à les restituer sous forme de textures).

1. Introduction

Dans ce projet, nous abordons la génération de texture. Cette tâche compliquée qui vise à reproduire “un subtil équilibre entre répétition et innovation”¹ peut être abordée via un simple automate cellulaire dont la règle de mise à jour est déterminée par un réseau de neurones [6]. Beaucoup de techniques existent pour reproduire des textures : la méthode de réaction-diffusion basée sur des modèles physiques de diffusion et réaction au cours du temps, méthode des fractals; la synthèse par contraintes statistiques avec entre autres le modèle de phases aléatoires qui génère une texture en modifiant la phase de l'image originale dans le domaine de Fourier [4] et son extension avec le Texton [2]; ou encore par décomposition parcimonieuse [9] ou avec la contribution de réseaux de neurones utilisés pour extraire des cartes d'attributs qui remplacent la base d'ondelettes dans une méthode proche de celle de Simoncelli-Portilla [5, 7]; et enfin la synthèse par copier-coller de patchs qui donne de très bons résultats sur les textures structurées, mais qui est extrêmement lente [3].

Parmi toutes ces méthodes, celles qui utilisent des réseaux de neurones convolutifs se sont révélées être

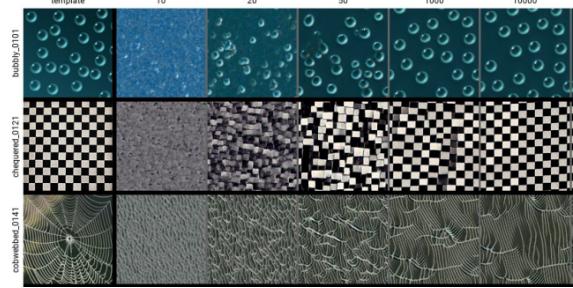


Figure 1: Génération de textures via l'automate cellulaire neuronal [6]

les plus fructueuses. En effet, ces réseaux entraînés sur des banques d'images gigantesques offrent des cartes d'attributs plus intéressantes et plus puissantes que les bases d'ondelettes ou les dictionnaires parcimonieux pour décrire les textures. En revanche, cette nouvelle méthode d'automate cellulaire opte pour une approche complètement différente : un réseau observateur est toujours utilisé pour comparer la texture en cours de génération et la texture cible. Cependant, au lieu de partir d'une image initialisée avec du bruit blanc et d'utiliser une décente de gradient pour générer la texture, un automate cellulaire gouvernée par des règles apprises durant la procédure de génération est utilisé. Cette nouvelle méthode particulièrement simple et facilement parallélisable parvient à générer des textures complexes, notamment les textures organiques telles que les feuilles, les cellules ou encore les bulles. Les auteurs insistent d'ailleurs sur le fait qu'il propose cet algorithme comme modèle biologiquement plausible de la formation distribuée de motifs de texture.

2. Travail proposé

Dans cette étude, nous reprenons une implémentation minimaliste de l'automate cellulaire neuronal sous PyTorch proposé par les auteurs de l'article [6]. À partir de ce code, nous allons d'abord analyser la règle de mise à jour : a) étude des paramètres de la règle asynchrone, b) implémentation d'une règle synchrone , c) implémentation

¹Citation d'Yves Meyer

d'une règle de voisinage, d) comparaison des 3 règles. Puis, nous allons étudier l'influence des hyperparamètres (profondeur, nombre d'unités, nombre de canaux) du réseau de neurones sur la qualité de génération de texture. Par la suite, nous travaillerons sur les filtres utilisés par le réseau : nous allons laisser le réseau apprendre les filtres qu'il souhaite pour modéliser la texture et comparer ces filtres appris aux opérateurs initialement utilisés. Enfin, nous étudierons l'utilisation du réseau Inception [8] pour calculer la fonction de coût durant l'entraînement ainsi que son influence sur la génération de texture.

3. Méthodes

3.1. Automate cellulaire neuronal

La méthode proposée dans l'article repose sur la génération de motifs par des équations aux dérivées partielles. Ces méthodes ont été très souvent utilisées pour modéliser la formation de systèmes dans la nature (par exemple avec les équations de réaction diffusion). On note ici $s(x, t) \in \mathbb{R}^C$ le vecteur d'état du système où $x \in \mathbb{R}^2$ définit sa position et $t \in \mathbb{R}_+$ le temps. Son évolution dans l'espace et le temps est régie par l'équation suivante :

$$\frac{\partial s}{\partial t} = f(s, \nabla_x s, \nabla_x^2 s)$$

La fonction f s'apparente à la règle d'évolution du système. Elle prend en paramètre le vecteur s , son gradient $\nabla_x s$ et son Laplacien $\nabla_x^2 s$.

Il est alors possible d'utiliser une approximation numérique de cette équation pour étudier le comportement de notre système. On discrétise l'espace et le temps de manière à ce que chaque position (x, y) représente les coordonnées d'un pixel. Ainsi, à chaque pas de temps $t \in \mathbb{N}$, l'état de l'automate est représenté par le vecteur s_t et son état au temps $t + 1$ est donné par :

$$p_t = concat(s_t, K_x * s_t, K_y * s_t, K_{lap} * s_t)$$

$$s_{t+1} = s_t + f(p_t) \delta_{x,y,t}$$

Les quantités $\nabla_x s$ et $\nabla_x^2 s$ sont modélisées par des convolutions entre l'automate et les filtres de Sobel. Ces filtres sont couramment utilisés pour la détection de contours dans les images. Ils permettent de prendre en compte l'état des cellules voisines. Le vecteur p_t appelé vecteur de perception dans l'article contient toute l'information de l'automate cellulaire à l'instant t . La quantité $\delta_{x,y,t}$ représente le pas de la discréttisation de l'espace. Dans la suite, on utilisera principalement un pas stochastique suivant une loi de Bernoulli de paramètre 0.5. Son utilisation permet d'avoir une mise à jour asynchrone des cellules de l'automate, et donc d'éviter une certaine dépendance avec l'état initial s_0 .

Architecture : Le modèle numérique développé précédemment définit l'évolution de l'automate cellulaire au cours du temps. Il reste à traiter du choix de la fonction de mise à jour de l'automate, représentée par la fonction f . Cette règle dépend principalement de la tâche que l'automate doit effectuer. L'objectif ici est de générer une image de texture à partir d'une image donnée. La fonction de mise à jour va donc dépendre de l'image que l'on souhaite reproduire et est généralement inconnue en pratique. Pour répondre à ce problème, les auteurs de l'article ont fait en sorte que cette fonction soit apprise par l'automate cellulaire grâce à un réseau de neurones profonds. On la définit de la façon suivante : $f(p) = ReLU(pW_0 + b_0)W_1 + b_1$ (réseau linéaire à une couche cachée avec fonction d'activation ReLU) où les paramètres $W_{0,1}, b_{0,1}$ dépendent de l'image en entrée et sont appris par le modèle. Ainsi, si on représente l'état du système à un instant t par le vecteur s_t , pour obtenir l'état s_{t+1} , on applique une première couche de convolution avec des paramètres fixés (filtres de Sobel) puis une deuxième couche avec des paramètres appris au fur et à mesure de l'entraînement. On répète ensuite ce processus jusqu'à convergence des paramètres du réseau. Le réseau obtenu est donc un réseau récurrent.

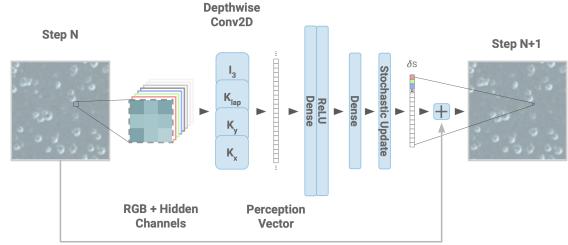


Figure 2: Architecture du modèle [6]

Fonction loss Pour mettre à jour les paramètres du réseau, les auteurs utilisent une fonction de loss qui s'appuie sur un transfert de style entre la texture cible (image que l'on souhaite reproduire) et l'état à un instant t de l'automate cellulaire. Le transfert de style est effectué grâce à un modèle de réseau de neurones pré-entraîné appelé réseau observateur. Les auteurs utilisent ainsi l'architecture VGG-16 entraînée sur la base de données ImageNet. Plusieurs couches de ce réseau sont utilisées pour calculer des matrices de Gram de la texture cible et des canaux RGB de l'automate cellulaire (trois premières composantes du vecteur s_t). Ces représentations par les couches de du réseau sont sensées caractériser le style de l'image. Ces matrices sont ensuite comparées en calculant la distance euclidienne entre chaque coefficient. Ainsi, une valeur faible de cette loss signifie que les deux images (texture cible et canaux

RGB de l'automate cellulaire) ont des représentations similaires dans différentes couches du réseau observateur et donc possèdent des caractéristiques de style commune.

Entraînement : Nous détaillons ici la procédure d'entraînement de l'automate neuronal cellulaire. On commence par initialiser un nombre K d'états de l'automate cellulaire. À chaque itération de l'algorithme, on choisit aléatoirement k états parmi les K possibles et on laisse évoluer ces états pendant $U \sim U([32, 96])$ pas de temps suivant les règles d'évolution présentées précédemment. On calcule ensuite la loss sur les k nouveaux états obtenus, puis on met à jour les paramètres du réseau définissant la fonction f par backpropagation. Pour favoriser une évolution rapide depuis un état initial jusqu'à un état final, les auteurs ajoutent une condition qui réinitialise aléatoirement un état toutes les 8 itérations.

3.2. Base de données

Comme dans l'article sur la génération de texture par automate cellulaire neuronal, la base de données *Describable Textures Dataset* est utilisée [1]. Il s'agit d'une base de données d'images humainement annotée qui contient des catégories d'images de type *damier*, *entrelacé*, *grille*, *bandes* ou encore *nervuré*.

3.3. Qualité de la texture

Dans les parties suivantes, nous allons présenter un certains nombre de tests réalisés à partir de la méthode proposée. Pour savoir si ces tests sont concluants, il faudrait être capable de quantifier les résultats obtenus, ce qui nous permettrait ensuite de les comparer. Cependant, après quelques recherches, nous n'avons pas trouvé de métrique claire permettant d'évaluer la qualité d'une texture. Ainsi, les interprétations des résultats seront réalisées à partir de critères "humains" telles que la ressemblance à une image cible, l'interprétabilité ou encore la quantité de détails.

4. Évaluation

4.1. La règle de mise à jour

Dans cette partie, la façon dont la texture est mise à jour par l'automate cellulaire est étudiée. Les auteurs de l'article proposent une règle asynchrone dans l'esprit de l'auto-organisation où chaque cellule est mise à jour selon une loi de Bernoulli de paramètre $p = 0.5$. En pratique, un masque est créé avec deux états pour mettre à jour ou non la cellule.

D'autres règles peuvent être utilisées pour mettre à jour la texture, nous allons étudier les règles suivantes :

- règle synchrone : toutes les cellules sont mises à jour à chaque itération. L'initialisation de la texture est

réalisée avec du bruit pour empêcher l'apparition de symétries.

- règle de voisinage : comme pour la règle asynchrone un masque est créé avec deux états, cependant l'état d'une cellule à l'itération suivante est dépendant des états voisins. En pratique, cela est fait en convolvant le masque avec un filtre à chaque itération. L'initialisation de la texture est réalisée comme pour la règle asynchrone avec un fond uni.

Les 3 règles donnent des résultats similaires. La figure 3 présente la génération d'une texture avec les 3 différentes règles.

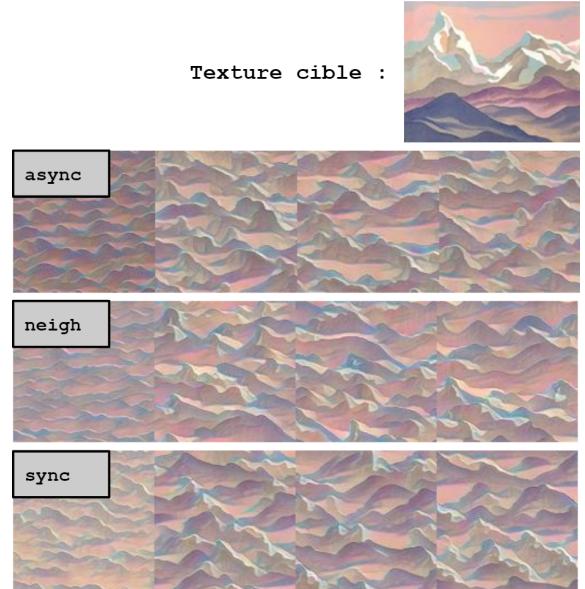


Figure 3: Synthèse d'une texture de montagne peinte avec les différentes règles.

Les 3 règles n'ont pas systématiquement une convergence et des résultats comparables. En effet, selon le filtre utilisé pour la convolution du masque dans la règle de voisinage, l'automate cellulaire peut être perturbé. Pour la figure 3, le filtre F_1 utilisé est le suivant :

$$F_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

. Le masque converge au bout de quelques itérations vers un système à deux états alternatif avec un motif qui ressemble à du bruit. En revanche, si l'on prend le filtre F_2 suivant, le masque prend la forme d'une grille.

$$F_2 = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 0 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

Comme précédemment, le masque converge vers un système alternatif rapidement. Cependant, cette fois, l'automate cellulaire n'est pas capable d'apprendre la texture. La figure 4 présente les résultats ainsi que le masque à une itération donnée.

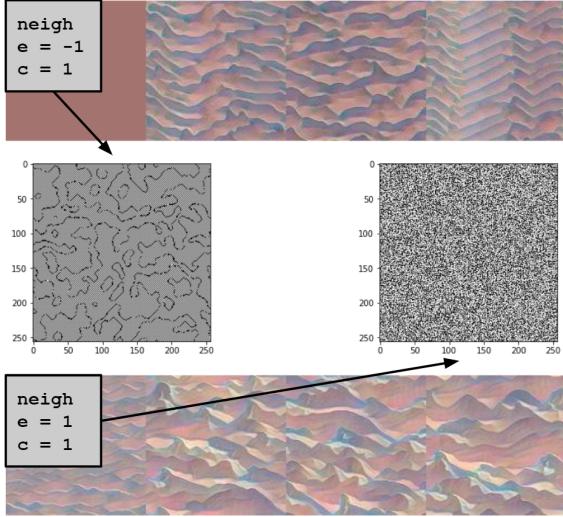


Figure 4: Synthèse d'une texture de montagne peinte avec les filtres F_2 (en haut) et F_1 (en bas) pour la règle de voisinage. Les masques sont également présentés.

De manière générale, nous avons testé des masques de la forme suivante pour avoir une cellule sur deux activée à chaque itération. Ici $o = -4 \times (c + e)$.

$$F = \begin{bmatrix} c & e & c \\ e & o & e \\ c & e & c \end{bmatrix}$$

Nous avons remarqué que tout filtre donnant naissance à un masque avec des régularités nuit à l'apprentissage du motif. Cette expérience permet de voir la robustesse de l'automate cellulaire par rapport à la règle de mise à jour implémentée. Nous étions curieux de voir l'effet d'entraîner l'automate avec une règle de mise à jour elle-même dictée par un automate. Nous pouvons en déduire que a) la méthode synchrone est une manière viable de faire converger l'automate même si cette méthode semble plus conserver la structure globale de l'image et b) tout masque de mise à jour présentant des régularités est néfaste à l'apprentissage. La règle de mise à jour asynchrone est donc à privilégier.

4.2. Hyperparamètres du réseau de neurones

Dans cette partie, l'influence de certains hyperparamètres du réseau de neurones sur la qualité de la texture générée sont analysés. Chaque expérience est répétée sur des textures différentes afin de voir si certains types de

texture sont plus facilement représentables que d'autres par une architecture. Les hyperparamètres suivants sont étudiés :

- le nombre de couches : 2, 3 ou 4 couches de neurones. En apprentissage automatique, au plus un réseau est profond au plus il parvient à modéliser des relations complexes entre l'entrée et la sortie grâce aux non-linéarités créées à chaque couche. Ce paramètre devrait donc permettre à l'automate de modéliser des règles plus ou moins complexes.
- le nombre de neurones par couche : 48, 96 ou 192. Au plus une couche a de neurones au plus elle peut exprimer des relations différentes entre les variables. Ce paramètre devrait ainsi permettre à la règle de mise à jour de modéliser précisément ou non un phénomène.
- le nombre de canaux utilisés dans le vecteur de perception : 3, 12 et 24. Cet hyperparamètre a un impact sur la quantité d'information utilisée pour construire la règle de mise à jour.

Dans un premier temps, nous allons étudier les paramètres de profondeur (L : nombre de couches) et largeur (U : nombre d'unités par couche) du réseau en fixant le nombre de canaux à 12 (C : nombre de canaux) comme décrit dans l'article de l'automate cellulaire neuronal (ACN). D'après nos tests, les textures de type organique avec des motifs auto-organisés comme les nervures d'une feuille sont les moins sensibles à la configuration du réseau. La figure 5 présente la génération d'une texture nervurée d'une feuille, la configuration ne semble pas avoir d'importance dans la qualité visuelle de la texture.



Figure 5: Synthèse d'une texture nervurée d'une feuille.

En revanche, pour des motifs géométriques tels que des damiers ou des motifs entrelacés : le nombre d'unités par couche joue un rôle important dans la restitution du motif

élémentaire et géométrique de la texture. La profondeur du réseau semble avoir plutôt un effet sur les tendances globales de la texture comme des effets de perspective sur le motif élémentaire. Les effets sont donc bien ceux attendus : au plus le réseau est large au mieux il retranscrit le motif élémentaire de la texture et au plus le réseau est profond au mieux il parvient à capturer des nuances dans la texture.

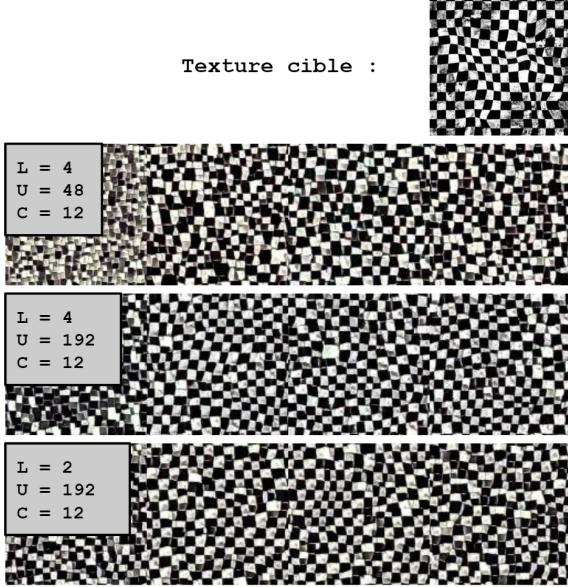


Figure 6: Synthèse d'un damier avec un effet de perspective.

Avec la figure 6, on voit très bien que le nombre d'unités a un impact important sur la restitution du damier, de plus la profondeur du réseau semble jouer sur la capacité du modèle à assurer la cohésion entre le damier et les effets de perspective : avec 4 couches le résultat est plus plaisant à regarder.

Le nombre de canaux utilisés par l'automate cellulaire peut aussi avoir un impact important sur la qualité des textures générées. Bien que la fonction loss n'utilise que les canaux RGB de l'automate cellulaire, la présence de canaux supplémentaires semble permettre à l'automate de mieux comprendre la complexité de certains motifs dans les textures. Intuitivement, si l'on utilise uniquement que trois canaux, toute l'information apprise au cours de l'entraînement sera visualisée, ce qui contraint énormément l'automate dans l'information apprise. L'exemple suivant réutilise une texture de type damier et montre les résultats obtenus pour des nombres différents de canaux.

La figure 7 montre clairement que le nombre de canaux utilisés peut avoir une influence considérable sur la qualité des résultats. Dans le cas de trois canaux, l'automate ne semble pas comprendre la structure géométrique de l'image et l'alternance entre les cases noires et blanches du damier.

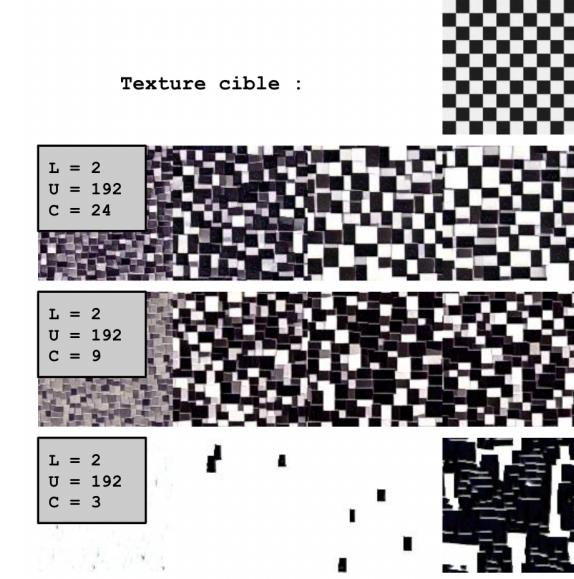


Figure 7: Synthèse d'une texture de damier pour des nombres différents de canaux

Lorsqu'on augmente ce nombre, on observe des motifs plus précis qui correspondent mieux à un damier.

Par ailleurs, le nombre de canaux utilisés semble aussi avoir un impact sur la convergence de l'automate cellulaire vers un état final. On observe d'une part que la vitesse de convergence de l'automate augmente avec le nombre de canaux, c'est-à-dire que l'automate converge vers un état final en un petit nombre d'itérations. D'autre part, avec un grand nombre de canaux, l'état final atteint par l'automate semble statique. En pratique, l'état final de l'automate cellulaire n'est pas unique, il converge vers un état qu'il maintient.

À part les hyperparamètres évoqués précédemment, il peut être intéressant de faire varier les filtres fixes qui permettent une approximation des quantités gradient et Laplacien $\nabla_x s$ et $\nabla_x^2 s$. Les filtres utilisés jusqu'ici sont des filtres de Sobel de tailles 3x3. Nous avons tenté de comprendre l'influence de ces filtres dans l'apprentissage de la règle de mise à jour par l'automate cellulaire en faisant varier leurs tailles. Nous avons pour cela utilisé des filtres de Sobel de tailles 5x5 et 7x7 et comparé les résultats obtenus pour plusieurs textures.

La figure 8 montre que les résultats de textures obtenus dépendent bien de la taille des filtres. Cependant, il n'est pas évident de quantifier clairement l'impact de ces filtres. Ces résultats semblent montrer que l'utilisation de "grands" filtres (taille 7x7) permet d'obtenir plus de détails dans les textures générées. Dans la texture 7x7, il est possible de distinguer les petites bandes contenues à l'intérieur des grandes. Dans la texture 3x3, on distingue clairement les

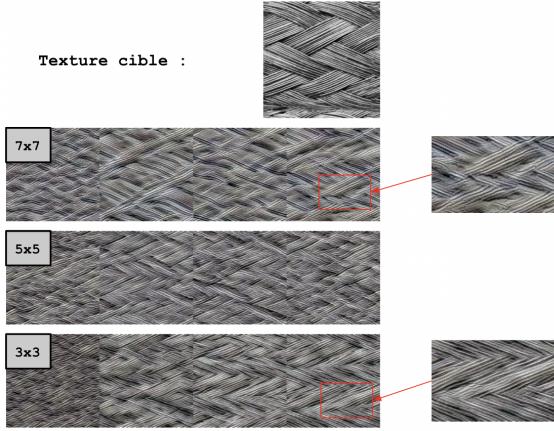


Figure 8: Textures obtenues en faisant varier la taille des filtres de Sobel (3x3, 5x5, 7x7)

grandes bandes mais les petites qui les composent ne sont pas extrêmement nettes.

Dans cette partie, nous proposons une dernière expérience qui consiste à faire varier encore une fois la profondeur et la largeur du réseau, mais cette fois sur une texture plus complexe. Il s'agit en réalité de deux textures juxtaposées.

différentes pour le nombre d'unités par couche. D'abord, nous pouvons remarquer que le réseau apprend une texture plus qu'une autre, cela reste vrai pour toutes les autres juxtapositions testées. En l'occurrence, pour celle-ci le réseau apprend davantage la texture de feuille que le paysage de montagne peint. Le nombre d'unités par couche a un impact clairement visible sur l'apprentissage des deux textures : au plus il y a d'unités au plus l'automate cellulaire est capable de synthétiser une image qui prend en compte les deux textures. La profondeur du réseau a un léger impact sur la cohésion des deux textures.

4.3. Le vecteur de perception

Comme expliquée section 3.1, le vecteur de perception $p \in \mathcal{R}^{4 \times C}$ contient la convolution entre le vecteur d'état du système $s \in \mathcal{R}^C$ et les 4 filtres : I (Identité), K_x (Sobel en x), K_y (Sobel en y) et K_{lap} (Laplacien). Dans cette expérience, deux choses peuvent être faites pour remplacer le vecteur de perception, soit le remplacer par une couche de convolution et ainsi laisser le réseau apprendre $4 \times C$ filtres, soit laisser le réseau apprendre les 4 filtres normalement utilisés et les appliquer sur les C canaux du vecteur d'état s . Pour toutes les expériences, la configuration du réseau est la suivante : $L = 2$, $U = 96$ et mise à jour asynchrone.

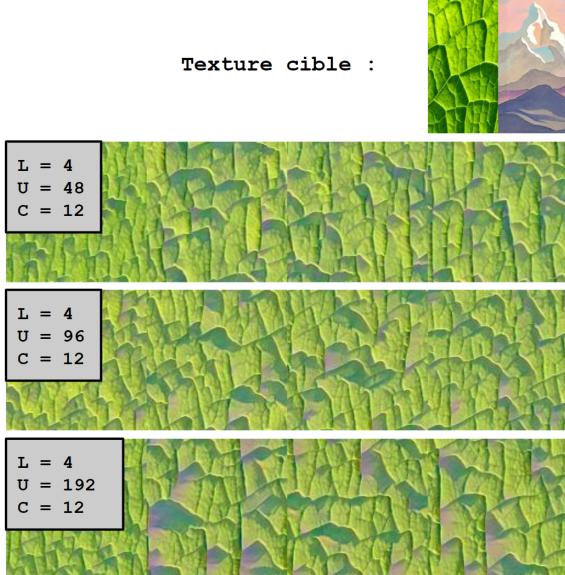


Figure 9: Synthèse de deux textures juxtaposées selon plusieurs configuration réseau.

Pour cette expérience, la règle asynchrone est utilisée et le nombre de canaux est fixé à 12. Le nombre de couches ainsi que nombre d'unités par couches sont les deux paramètres étudiés ici. La figure 9 présente les résultats pour un réseau de 4 couches avec 3 configurations

Remplacement de p par une couche de convolution. Cette expérience donne le plus de liberté au réseau puisqu'il peut apprendre $C \times 4$ filtres différents. Dans les prochaines expériences, nous avons essayé plusieurs valeurs de $C \in [4, 6, 12]$ ainsi que plusieurs tailles de filtre : $f = 3 \times 3$, 5×5 et 7×7 .

Comme vu dans la section 4.2, la figure 10 montre que le nombre de canaux joue un rôle important dans la restitution de la texture. Même si le réseau a plus de liberté avec cette configuration, 12 canaux et donc 48 filtres sont nécessaires pour créer une texture qui s'approche de l'originale. La taille des filtres de convolution a une moindre influence comme le montre la figure 11. Si de manière générale nous pouvons dire qu'augmenter le nombre de canaux est bénéfique pour représenter n'importe quelle texture, la taille des filtres de convolutions est un paramètre dépendant de la texture. Pour certaine texture comme le damier ci-dessous, les filtres de taille 5×5 semblent être les plus performants, mais pour d'autres textures les filtres 3×3 ou 7×7 étaient plus adaptés.

La diversité des 48 filtres produit parfois des résultats visuellement meilleurs mais ce n'est pas systématique. Le temps d'entraînement est également allongé. Face à ce constat, il est préférable d'utiliser les opérateurs utilisés par les auteurs.

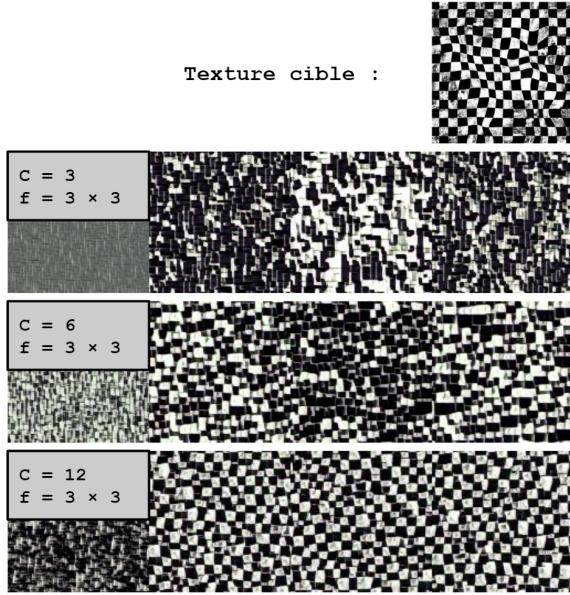


Figure 10: Synthèse d'une texture de damier pour des nombres différents de canaux

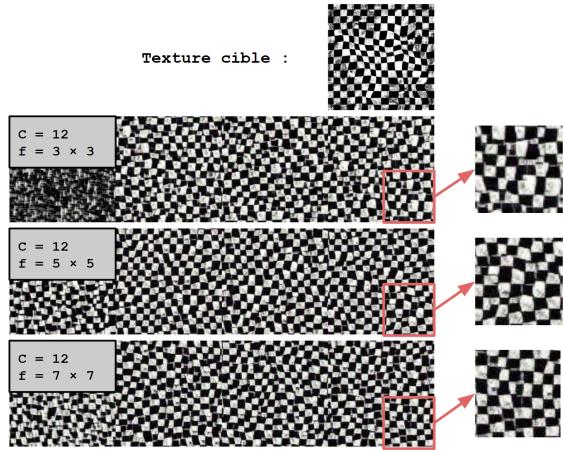


Figure 11: Synthèse d'une texture de damier pour différentes tailles de filtre de convolution

Apprentissage des filtres de convolution du vecteur p . Cette méthode plus contraignante mais plus facile à interpréter consiste à apprendre les 4 filtres de convolution initialement fixés à l'identité (I), les filtres de Sobel (K_x , K_y) et le Laplacien (K_{lap}). De la même façon, le nombre de canaux et la taille des filtres sont étudiés.

De manière générale, les résultats ont été moins bons que dans le cas de filtres fixés. Comme auparavant, la qualité de l'image est très dépendante du nombre de canaux et la taille des filtres est dépendante de la texture apprise. La figure 12 présente une comparaison entre les filtres fixés et les filtres appris par le réseau pour l'une des meilleures configurations

retenues lors des essais. Les 4 images représentent la convolution par canaux des 3 couches RGB par les 4 différents filtres. L'automate a accès à toutes ces couches durant son entraînement.

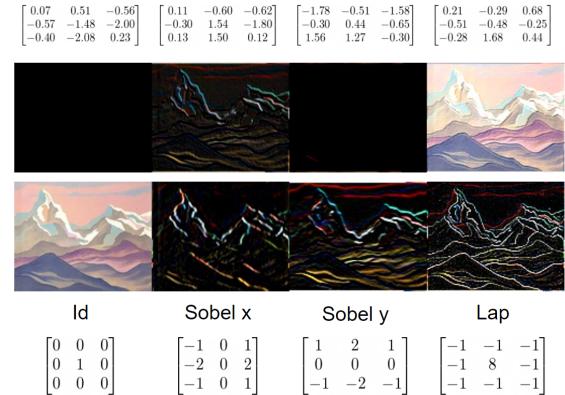


Figure 12: Comparaison des filtres appris et des filtres fixés pour une texture de montagne peinte.

Deux filtres semblent n'avoir que très peu d'intérêts. En revanche, un filtre est sensiblement similaire à l'identité et l'autre est un mélange entre le Laplacien et le filtre de Sobel en y . La figure 10 présente également le résultat de l'application des filtres retenus pour la meilleure configuration entraînée sur la texture de damier. Les coefficients ne sont pas présentés ici mais il est aisément de comprendre leur effet via les images. Cette fois, les 4 filtres sont très différents. Le premier et dernier filtres semblent être équivalents aux filtres de Sobel mais avec une rotation de 45. Le deuxième filtre détecte les carrés noirs du damier et le troisième semble détecter les bandes diagonales du motif dans une certaine orientation.

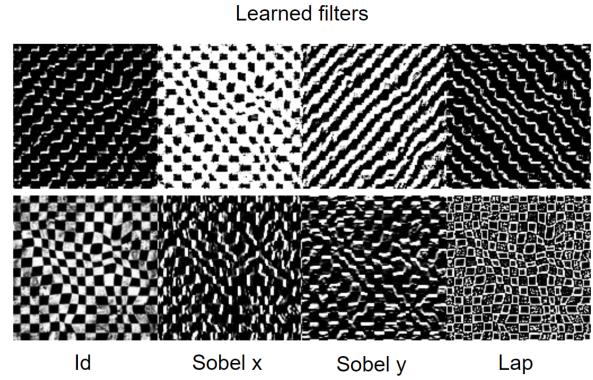


Figure 13: Comparaison des filtres appris et des filtres fixés pour une texture de damier.

Finalement, les filtres appris sont proches des opérateurs originalement utilisés. D'après nos tests, apprendre ces 4

filtres n'a pas d'intérêt : l'entraînement est légèrement plus long et les résultats moins satisfaisants. De plus, nous avons remarqué que l'automate ne converge pas toujours selon la taille des filtres utilisés.

4.4. Le réseau Inception

Les auteurs proposent dans l'article une autre méthode de génération de textures qui reposent sur le même schéma numérique et la même architecture que développés dans la section 3.1. Dans cette méthode, il n'est plus question de transfert de style entre une image cible et les canaux RGB de l'automate cellulaire. Dans ce cas, les paramètres de la fonction de mise à jour notée f sont appris en maximisant l'activation d'une couche d'un réseau observateur. Ainsi, l'automate cellulaire va converger vers un état final qui maximise l'activation d'une couche du réseau. Les auteurs utilisent le réseau Inception entraîné sur ImageNet, comme observateur car certaines couches profondes de ce réseau fournissent des interprétations et des motifs complexes appris sur ImageNet. Cette méthode permet d'obtenir de nouvelles textures et offre un nouveau potentiel créatif.

Cette méthode nécessite de changer le mode d'apprentissage du réseau. Pour cela, les auteurs utilisent une fonction loss qui va calculer l'activation d'une couche donnée du réseau Inception pour un état de l'automate cellulaire.

La figure 14 montre différents résultats de textures en maximisant l'activation de certaines couches profondes du réseau Inception. On peut admirer le côté artistique et original proposé par cette méthode.

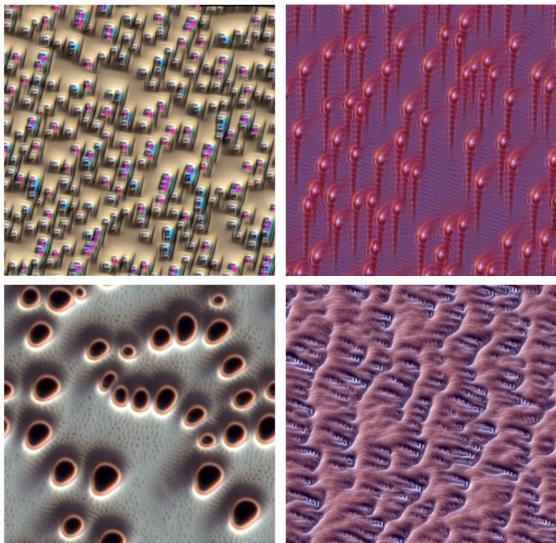


Figure 14: Textures obtenus par la loss *Inception* pour différentes couches

5. Conclusion

Dans ce projet, nous avons présenté et testé la méthode de génération de textures par automate cellulaire neuronal décrite dans l'article [6]. Nous avons testé cette méthode au travers de plusieurs expériences (variations des paramètres de l'automate et du réseau, quantification de leurs impacts, changement de loss...). Cela nous a permis de comprendre son potentiel et ses limites. Les résultats obtenus nous indiquent que cette méthode de génération de textures semble robuste à différents changements de paramètres ou d'architecture, et qu'elle dispose d'un potentiel génératif assez conséquent. Les textures obtenues peuvent être à la fois diverses et relativement complexes, ce qui signifie que cette méthode parvient à extraire certaines caractéristiques des images et à les restituer de façon appropriée.

References

- [1] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014. [3](#)
- [2] Agnès Desolneux, Lionel Moisan, and Samuel Ronsin. A compact representation of random phase and gaussian textures. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1381–1384, 2012. [1](#)
- [3] A.A. Efros and T.K. Leung. Texture synthesis by non-parametric sampling. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1033–1038 vol.2, 1999. [1](#)
- [4] Bruno Galerne, Yann Gousseau, and Jean-Michel Morel. Micro-Texture Synthesis by Phase Randomization. *Image Processing On Line*, 1:213–237, 2011. https://doi.org/10.5201/ipol.2011.ggm_rpn. [1](#)
- [5] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. [1](#)
- [6] Alexander Mordvintsev, Eyyvind Niklasson, and Ettore Randazzo. Texture generation with neural cellular automata, 2021. [1, 2, 8](#)
- [7] 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. [1](#)
- [8] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. [2](#)
- [9] Guillaume Tartavel, Yann Gousseau, and Gabriel Peyré. Variational texture synthesis with sparsity and spectrum constraints. *Journal of Mathematical Imaging and Vision*, 52(1):124–144, 2015. [1](#)