Advanced Methods for Image Processing
TP2: Deep learning for texture synthesis

Hugo Trarieux - Florian Dayre

Présenté à Aurélie Bugeau

5 Décembre 2021

1 Question 1

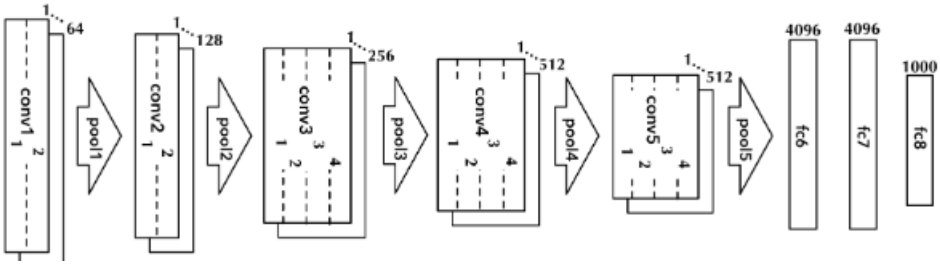

FIG. 1 – *Architecture du réseau VGG-19*

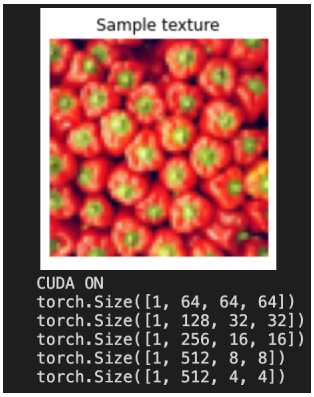

FIG. 2 – *Image de taille 256x256, passée à travers le réseau VGG*

Ce que l'on peut voir sur la Figure 2 ci-dessus correspond aux résultats des differentes couches après un passage de cette image à travers le réseau VGG-19. Le premier nombre correspond au batch size, disons plutôt le nombre d'images que l'on envoie dans le réseau. Ensuite, on retrouve la dimension des descripteurs, qui correspond aux nombres de filtres. À noter que ce nombre correspond, comme on peut le constater sur la Figure 1, aux profondeurs des couches du réseau VGG. 2 couches avec des descripteurs de dimension 64, 2 couches avec des descripteurs de dimension 128, et ainsi de suite. Puis, les deux derniers nombres des résultats indiquent la taille de l'image, qui diminuera au fil des couches du réseau.

2 Question 2

Tout d'abord, expliquons la formule de la Figure 3. Cette dernière n'est d'autre qu'un calcul de loss avec une MSE (Mean-squared error). On cherche à minimuser l'erreur entre 2 éléments qui, dans ce cas, sont des matrices de Gramm. La première, $\Phi(G_\theta(z))$ correspond à la matrice de Gramm de l'image générée par le générateur. La seconde correspond elle à la matrice de Gramm de l'image de référence, celle vers laquelle on souhaite tendre.

$$\min_\theta \mathbb{E}_{z\sim Z} \sum_l \|\Phi_l(G_\theta(z)) - \Phi_l(s)\|_2^2$$

FIG. 3 – *Fonction MSE appliquée sur 2 matrices de Gramm*

La matrice de Gramm permet de donner plus de poids aux descripteurs les plus importants. On retrouve son calcul sur la Figure 4 ci-dessous. Prenons par exemple une image avec beaucoup de rouge, la matrice de Gramm accordera beaucoup d'importances au canal rouge.

$$\Phi_l(u) = \frac{1}{H_l W_l D_l} \phi_l(u) \phi_l^T(u)$$

FIG. 4 – *Fonction de calcul de la matrice de Gramm pour une image* $\boldsymbol{u}$ *au niveau* $\boldsymbol{l}$

3 Question 4

Une fois notre modèle entrainé avec les meilleurs paramètres trouvés, nous l'avons utilisé pour générer une image de poivrons à partir d'une image de bruit. Lors de l'entrainement, il fallait donner en entré une image vers laquelle tendre. Mais lorsque ce que le modèle est entrainé, plus besoin de donner cette image de référence qui servait d'entrainement. Lancer le modèle à partir d'une image de bruit suffit. On constate sur la figure 16 différents résultats obtenus avec notre modèle entrainé avec les paramètres ci-dessous.

- Loss function : MSE
- Learning Rate = 0.01
- Batch Size = 50
- Iterations = 500

FIG. 5 – *Images générées à partir de notre modèle entrainé*

4 Question 5

FIG. 6 – *Images de Test, peppers.jpg et raddish.jpg*

4.1 Différentes fonction de perte

Nous avons réalisé ces tests en changeant la fonction de perte pour le calcul de la loss mais en conservant les paramètres de base, à savoir 300 itérations, un batch_size de 10 et un learning_rate de 0.1.

4.1.1 Cross Entropy

La cross entropy ne donne clairement pas des résultats satisfaisants. On peut voir qu'on ne retrouve pas les couleurs de l'image d'origine, ni même les formes [Fig.7, Fig.8]. Cette fonction de perte est surtout utilisée pour la classification d'image et non pour la comparaison, ce qui peut expliquer ces résultats aberrants.

FIG. 7 – *Résultats avec la fonction Cross Entropy sur l'image Peppers.jpg*

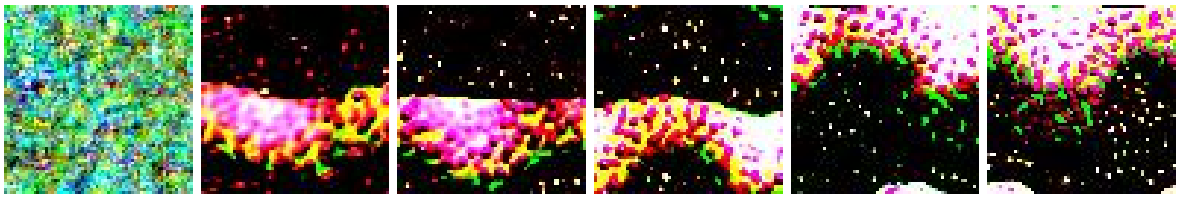

FIG. 8 – *Résultats avec la fonction Cross Entropy sur l'image raddish.jpg*

4.1.2 MAE

La fonction de perte MAE (Mean Absolute Error) est utilisée pour comparer l'erreur absolue entre deux images, elle convient donc pour notre programme. On peut voir qu'elle donne des résultats satisfaisants. On retrouve la bonne répartition des couleurs. En effet, pour peppers.jpg on voit bien que les couleurs verts/jaunes se situe au centre des formes rondes de couleurs rouges[Fig.9]. Pour raddish.jpg, on retrouve des formes rondes de couleur rouge et du vert disséminé entre[Fig.10].

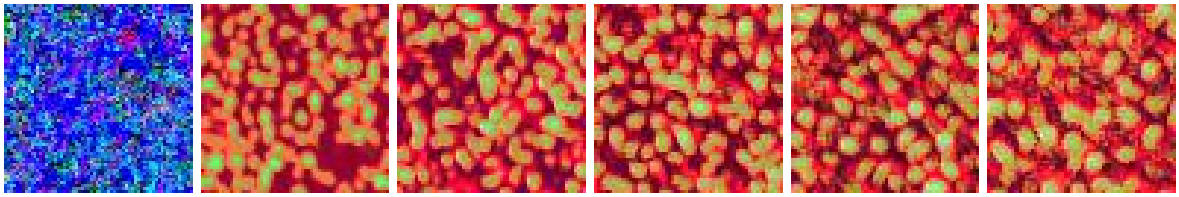

FIG. 9 – *Résultats avec la fonction MAE sur l'image Peppers.jpg*

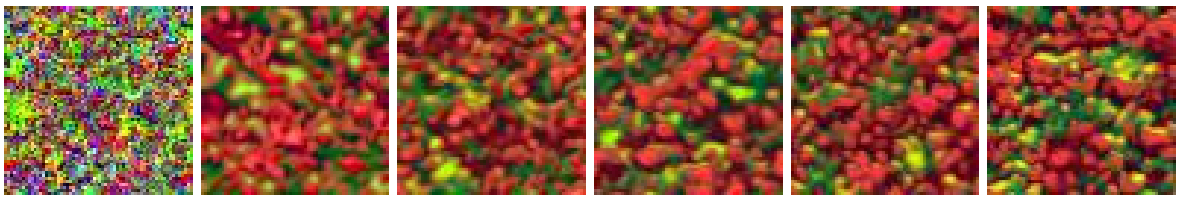

FIG. 10 – *Résultats avec la fonction MAE sur l'image Raddish.jpg*

4.1.3 MSE

La MSE (Mean Squared Error) est la fonction de perte qui nous été donné dans le sujet. Il est donc normal d'obtenir les meilleurs résultats avec celle-là. En effet, les résultats obtenus sont beaucoup plus nets qu'avec la MAE. Avec peppers.jpg, on voit bien que les frontières entre la couleur rouge et la couleur verte sont beaucoup mieux définies[Fig.11]. En ce qui concerne raddish.jpg, l'amélioration des résultats est moins voyante mais les résultats restent aussi satisfaisants qu'avec la MAE [Fig.12].

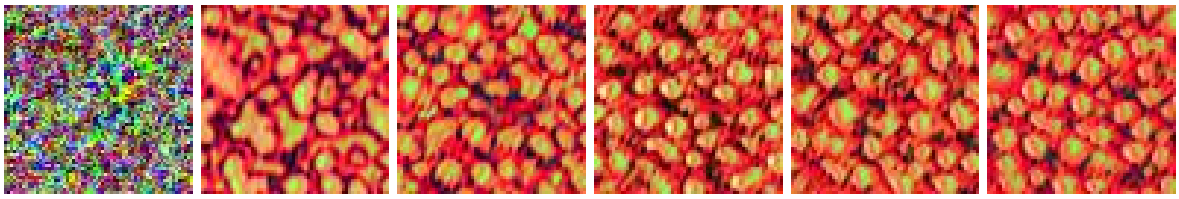

FIG. 11 – *Résultats avec la fonction MSE sur l'image Peppers.jpg*

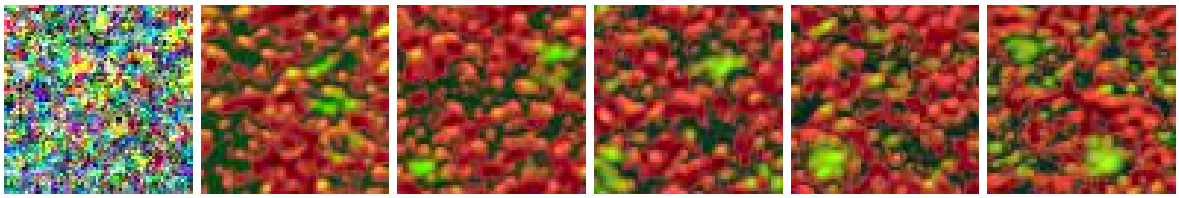

FIG. 12 – *Résultats avec la fonction MSE sur l'image Raddish.jpg*

4.2 Modification du learning_rate et du batch_size

Nous avons conclut que la MSE était la fonction de perte qui donnait les meilleurs résultats. Nous sommes donc repartis de cette dernière pour effectuer de nouveaux tests. Nous avons modifié le learning_rate qui correspond au pas lors de la descente de gradient. Si il est trop élevé, on risque de ne jamais converger et si il est trop bas la convergence risque d'être beaucoup trop longue. Nous avons également modifié le batch_size qui correspondant à la taille des données d'entraînement.

Pour les deux figures ci-dessous, nous avons donné un pas d'apprentissage de 0.1 [Fig.13] et de 0.01 [Fig.14] et fixé le batch_size à 50. Nous pouvons voir qu'avec un learning_rate plus petit, les résultats obtenus ont plus de détails. La dernière image de la Fig.14 est moins flous que celle de la Fig.13.

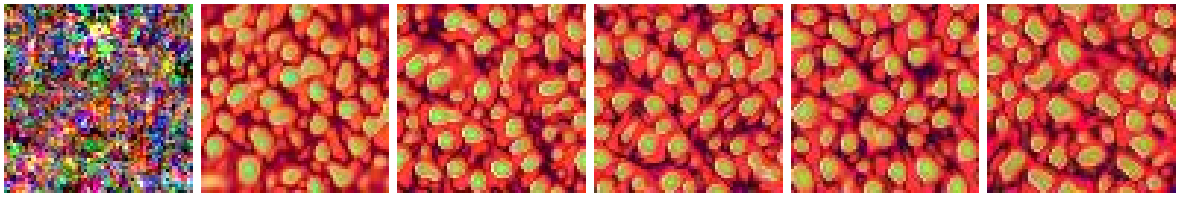

FIG. 13 – *Résultats avec un learning_rate de 0.1 et un batch_size de 50 pour 300 itérations sur l'image Peppers.jpg*

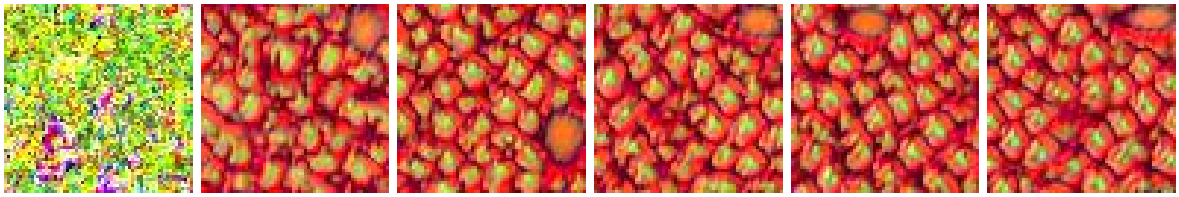

FIG. 14 – *Résultats avec un learning_rate de 0.01 et un batch_size de 50 pour 300 itérations sur l'image Peppers.jpg*

Pour les deux figures ci-dessous, nous avons fixé le pas d'apprentissage à 0.01 et donné un batch_size de 10 [Fig.15] et de 50 [Fig.16]. Nous pouvons voir que l'augmentation du batch_size donne des résultats plus satisfaisants, avec moins d'artefacts.

FIG. 15 – *Résultats avec un learning_rate de 0.01 et batch_size de 10 pour 500 itérations sur l'image Peppers.jpg*

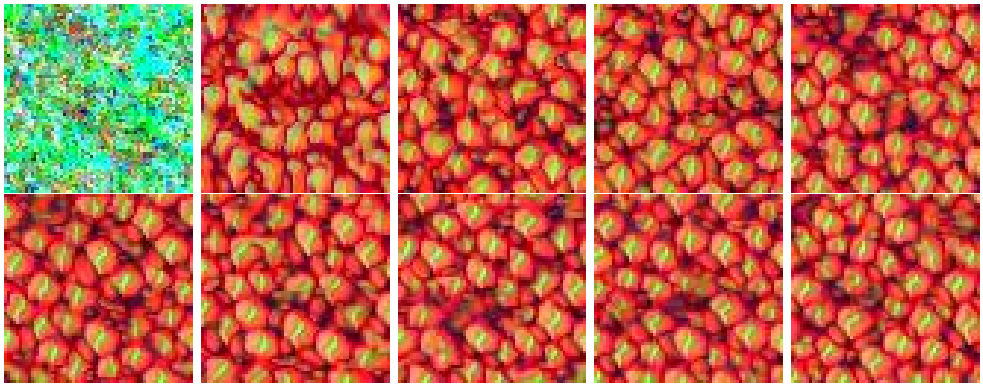

FIG. 16 – *Résultats avec un learning_rate de 0.01 et batch_size de 50 pour 500 itérations sur l'image Peppers.jpg*