Compte-rendu Inpainting de micro-textures

ROUILLE Nathan, KHOSROWSHAHI Sacha

Novembre 2024

1 Introduction

L'inpainting est une méthode de complétion d'images à partir des données disponibles. Dans ce projet, nous nous intéressons à l'inpainting de micro-textures. Ces dernières sont particulières en ce qu'elles peuvent être modélisées par des champs aléatoires, permettant d'approcher une micro-texture connue à l'aide de réalisations stochastiques. Nous utilisons, dans cette étude, des champs gaussiens pour modéliser les micro-textures.

L'objectif est d'inpainter une micro-texture à partir d'une image complète en niveaux de gris, puis d'appliquer cette méthode à des images en couleur masquées par des formes quelconques. Nous décrivons la méthode utilisée, les problèmes rencontrés, les solutions retenues,les résultats ainsi que les pistes d'amélioration pour évaluer la validité de cette méthode.

L'expérience acquise lors du module Proj104 l'année dernière, où nous avons travaillé sur un article de recherche, nous a permis d'organiser plus efficacement ce projet. L'erreur principale que nous avions comise était de vouloir analyser intégralement un article avant d'entamer des expérimentations. Cette fois-ci, nous avons adopté une approche parallèle entre théorie et pratique, vérifiant ainsi notre compréhension au fur et à mesure. Cela a été particulièrement pertinent pour des sujets complexes impliquant des images et des couleurs afin de mieux comprendre les objets mathématiques en les manipulant.

Dans ce document, nous introduisons d'abord le modèle ADSN (*Asymptotic Discrete Spot Noise*), qui permet de synthétiser des micro-textures. Nous expliquons ensuite le conditionnement appliqué à l'ADSN, notamment à travers la création de masques. Enfin, nous présentons les coefficients de Kriging, qui permettent de projeter le modèle généré sur les contraintes imposées par le conditionnement.

Nous travaillons avec des images, modélisées comme des fonctions $\Omega \to \mathbb{R}^d$, où $\Omega \subset \mathbb{Z}^2$, $d = 1$ pour les images en niveaux de gris et $d = 3$ pour les images en couleur, d représentant le nombre de canaux.

2 Modèle ADSN

2.1 Introduction à l'ADSN

La méthode fondamentale pour simuler une micro-texture consiste à convoluer une image centrée, normée avec un bruit blanc W. Pour des raisons d'efficacité, nous utilisons la transformée de Fourier pour effectuer le produit au lieu d'une convolution classique.

2.2 Modèle Oracle

On appelle modèle Oracle le fait de générer une synthèse de la microtexture en prenant en compte l'intégralité des données de l'image. C'est à dire que nous prenons $u : \Omega \to \mathbb{R}^d$, notre image et d'après le modèle ADSN en introduisant

$$t_u = \frac{1}{\sqrt{|\Omega|}}(u - \overline{u})$$

avec

$$\overline{u} = \frac{1}{|\Omega|} \sum_{x \in \Omega} u(x)$$

on obtient le vecteur centré, normalisé que nous voulons convoluer avec le bruit blanc W.
Il est important de préciser les dimensions des objets utilisés. On peut choisir la taille de la texture que l'on veut synthétiser en agrandissant la taille du bruit blanc d'un facteur k et en 0-paddant l'image t_u d'un même facteur.

2.2.1 Niveaux de Gris

La première implémentation du projet est de synthétiser une micro-texture à partir d'une image non-masquée et en niveaux de gris.

Figure 1: Micro-texture 1 en niveaux de gris

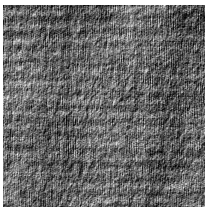


Figure 2: Modèle oracle 1 generé

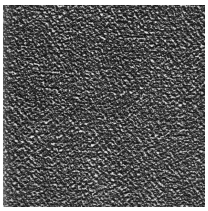


Figure 3: Micro-texture 2 en niveaux de gris

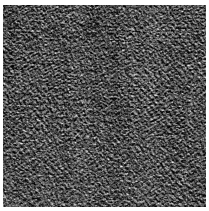


Figure 4: Modèle oracle 2 generé

Afin de vérifier la cohérence de nos résultats on regarde la moyenne $\mathbb{E}$ on trouve des moyennes similaires à 10^{-6} près. On trouve des résultats satisfaisants visuellement et qui sont cohérents statistiquement. Nous avons donc généré une synthèse d'une micro-texture à un canal et non masquée sans conditionnement.

Cette méthode plutôt élémentaire nous a permis de comprendre l'utilisation des objets, des dimensions et de comprendre aussi le but de travailler avec des champs gaussiens quant il en vient à des micro-textures. En effet ces dernières ne présentant pas de motifs géométriques forts ou de régularités marquées, on peut les synthétiser grâce à des champs aléatoires.

2.2.2 Adaptation à la couleur

Il nous faut maintenant pouvoir adapter cette méthode à la couleur. Nous nous interessons donc à une image $u : \Omega \to \mathbb{R}^3$. Afin de suivre le modèle oracle et de maintenir la corrélation entre les canaux ce qui est important pour notre perception des micro-textures, il est important de convoluer chaque canal de couleur par la même gaussienne.

Figure 5: Micro-texture en couleur

Figure 6: Modèle oracle en couleur

	Channel	Image (Mean, Q1, Q3)	Estimation (Mean, Q1, Q3)
0	Red	(137.12, 113.0, 164.0)	(137.12, 114.31, 160.14)
1	Green	(171.15, 147.0, 198.25)	(171.15, 148.25, 194.18)
2	Blue	(147.02, 123.0, 174.0)	(147.02, 124.25, 170.05)

Figure 7: Comparaison des statistiques des 2 images

Un des problèmes que nous avons rencontré avec ce premier modèle est l'apparition de pixels aberrants sur la micro-texture synthetisée. En effet comme on le voit sur les Figure 7 et Figure 8 , des pixels roses non présents dans notre image originale. Nous avons pu résoudre ce problème en comprenant ce qu'il se passait lors de l'affichage de la micro-texture synthétisée.

Cette erreur venait du fait que dans la création de notre synthèse on initiait l'image synthétisée avec `np.zeros_like()`, or si on ne précise pas le type de valeurs que nous voulons dans notre image, elle initie automatiquement les valeurs à des entiers. Or l'image que nous calculons avec la convolution d'une Gaussienne prends des valeurs flottantes, certaines valeurs float étaient donc

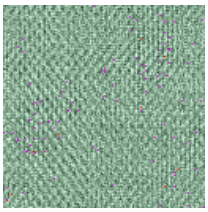


Figure 8: Apparition d'aberrations

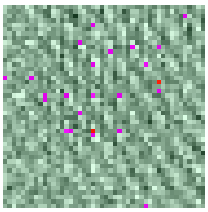


Figure 9: Zoom sur la micro-texture

arrondies à des entiers d'où l'apparition de pixels non existants dans le modèle de base. On a donc modifié le code ainsi : `np.zeros_like(,dtype='float')`.

3 Conditionnement

Maintenant que l'on sait synthétiser une microtexture semblable, il est possible de commencer à s'attaquer à la vraie nature de notre projet : l'inpainting. En effet, on peut essayer de combler les "trous" d'une image masquée par une texture que l'on sythétise avec les données disponibles. Nous verrons les problèmes induit par une telle méthode et comment y remédier.

3.1 Le masque

Nous avons donc commencé par faire de l'inpainting sur une image que l'on a masquée avec un masque carré en son centre. On applique le modèle ADSN que nous avons vu précédemment sur la partie de l'image disponible, en prenant garde à ne normaliser que par le nombre de pixels disponibles et en gardant le masque sur l'image centrée, normée. Puis on comble les parties manquantes de l'image par cette texture synthétisée. On remarque sur la Figure 11 que cette méthode présente un défaut : on peut distinguer des discontinuités sur les bords du masque car la synthèse est ressemblante mais ne se supperpose pas parfaitement avec l'image originale.

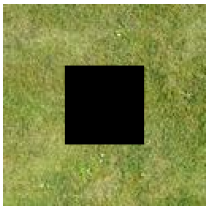


Figure 10: Image masquée

Figure 11: ADSN + image masquée

Figure 12: délimitation marquée

Nous avons également implémenté la possibilité de dessiner n'importe quel masque pour avoir plus de liberté sur l'inpainting.

Figure 13: Interface de sélection du masque

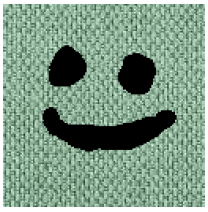


Figure 14: Image masquée

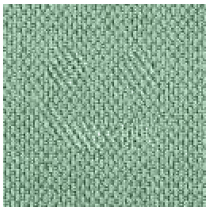


Figure 15: Inpainting de l'image avec ADSN

La solution pour remédier à ce problème de discontinuité est de conditionner la texture synthétisée par l'ADSN pour qu'elle soit égale aux valeurs de la microtexture originale sur les pixels disponibles, ce qui assurerait la continuité de l'inpainting. Un tel conditionnement sur la totalité des pixels disponibles serait très chronophage, nous avons donc généré une bordure des masques en les convoluant par un noyau moyenneur puis en seuillant tous les pixels positifs à 1 et en supprimant le masque original. On obtient alors un contour du masque de taille variable en choisissant la taille du noyau de convolution. Nous verrons par la suite que le conditionnement sur cette bordure est très satisfaisante pour assurer la continuité et permet d'économiser beaucoup de temps.

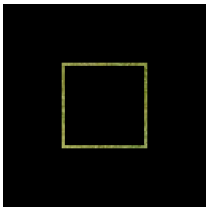


Figure 16: contour de conditionnement carré

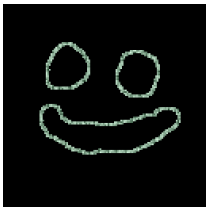


Figure 17: contour de conditionnement libre

3.2 Coefficients de Kriging

Nous avons donc un nouvel objectif : créer de la continuité au bord de notre masque. Afin de faire cela, il faut pouvoir projeter notre modèle ADSN sur le conditionnement autour de notre masque. Cela passe par introduire la projection dans un espace Hermitien avec comme produit scalaire : Cov. On de-

terminera pour un vecteur Gaussien centré F sa projection sur le contour C, noté

$$F^*(x) = \sum_{c \in C} \lambda_c(x) F(c).$$

Les $(\lambda_c)_{c \in C}$ sont les coefficients de Kriging, on les regroupera dans la matrice Λ. Le but est donc de determiner ces coefficients à partir du contour que l'on a crée et de la microtexture. Afin d'obtenir le modèle voulu on utilise la formule du papier :

$$(u - \overline{v})^* + F - F^* = F + \Lambda \left((u - \overline{v} - F)|_C \right. \quad (1)$$

Ce modèle pourra remplir l'image masquée et grâce au conditionnement la limite ne sera plus visible.

On peut obtenir Λ grâce à l'équation de Kriging :

$$\sum_{d \in C} \lambda_d(x) \Gamma(d, c) = \Gamma(x, c), \quad \text{i.e.} \quad \lambda(x) \Gamma|_{C \times C} = \Gamma|_{\{x\} \times C}.$$

Avec Γ la matrice de covariance. Et alors $\Lambda = \Gamma|_{\Omega \times C} \Gamma|_{C \times C}^{\dagger}$

Ainsi le calcul du modèle peut se faire en 2 étapes :

1. Calcul de $\psi = \Gamma^{\dagger}\big|_{C \times C} \varphi$ avec $\varphi = ((u - \overline{v} - F)|_C$
2. Calcul des composantes d'innovation et de Kriging à partir de ψ

3.2.1 Matrice de covariance

Ainsi pour la première étape, il faut calculer $\Gamma\big|_{C \times C}$. Il est, ici aussi, important de s'intéresser aux dimensions des objets utilisés. En effet Γ n'est pas une simple matrice, c'est un tenseur car Γ est défini sur $\Omega \times \Omega$ et à valeur dans $\mathbb{R}^{d \times d}$ tel que $\Gamma((i,k),(j,l)) = Cov(F(i,k), F(j,l))$.
Comme on suppose que la microtexture est stationnaire, alors sa covariance ne dépend que de la distance entre les pixels. Ainsi on peut calculer la matrice de covariance à partir de l'autocovariance cv : $\Gamma((i,k),(j,l)) = cv(j-i, l-k)$.
Pour calculer Γ il nous suffit donc de calculer cv or on sait d'après le théorème d'Herglotz que pour une microtexture on a :

$$c_v(x, y, i, j) = \mathcal{F}^{-1}\Big(\mathcal{F}(t_v(\cdot, \cdot, i)) \cdot \overline{\mathcal{F}(t_v(\cdot, \cdot, j))} \Big)(x, y)$$

avec t_v la microtexture centrée normée (sur les pixels non masqués). Ainsi c_v est définie sur Ω et à valeur dans $\mathbb{R}^{d \times d}$.
Pour calculer ψ, on commence par linéariser φ sur le contour. Pour ce faire, on linéarise le contour qui est un masque binaire, puis on garde les indices où le masque linéarisé vaut 1. Puis on linéarise φ de la même manière et on garde uniquement les valeurs de φ aux indices précédents. Enfin on garde un bijection

entre les indices de φ linéarisé et ses anciens indices dans Ω.
Il faut ensuite créer la matrice $\Gamma_{C\times C}$, pour ce faire on exploite la relation entre Γ et c_v en utilisant la bijection entre les indices du contour et les indices de Ω (dont on fait la différences dans c_v).

Enfin on transforme le tenseur $\Gamma_{C\times C}$ de dimension ($|C|$,$|C|$,d,d) en un matrice de dimension (d*$|C|$,d*$|C|$) et la matrice φ de dimension ($|C|$,d) en un vecteur de dimension (d*$|C|$). Ainsi on peut calculer : $\psi = np.linalg.lstsq(\Gamma_{C\times C},\varphi)$ (si on sait que $\Gamma_{C\times C}$ est inversible on peut utiliser np.linalg.solve qui implémente alors la méthode de Chloesky étant donné que $\Gamma_{C\times C}$ est symétrique définie positive si la $|C|$ est grand, on peut utiliser une descente de gradient pour accélérer la résolution).

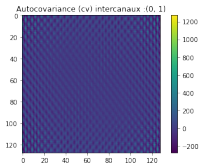


Figure 18: Exemple d'une autocovariance intercanaux

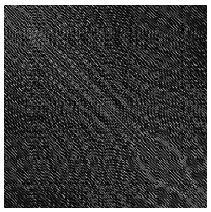


Figure 19: Exemple d'une matrice de covariance pour une image en niveaux de gris

3.2.2 Composantes d'Innovation et de Kriging

Après avoir calculé ψ on peut maintenant calculer les composantes de Kriging et d'innovation. Pour ce faire on commence par transformer ψ qui est un vecteur de dimension (d*$|C|$) en un matrice de dimension ($|C|$,d) puis on fait du zéro-padding pour le transformer en une matrice Ψ de dimension ($|\Omega|$,d) à l'aide de la bijection d'indices entre C et Ω. On peut ensuite calculer :
$composante_{kriging} - composante_{innovation} = c_v * \Psi$
La difficulté ici est que cv est de dimension ($|\Omega|$,d,d) et Ψ de dimension ($|\Omega|$,d).
On fait donc :

$$\text{comp}_i(x,y) = \sum_{j=1}^{C} \left(\Psi_j * c_v^{j,i}\right)(x,y)$$

où pour tout i allant de 1 à d $comp_i$ est le canal numéro i de $composante_{kriging} - composante_{innovation}$ on obtient donc une matrice de dimension ($|\Omega|$,d).

3.2.3 Algorithme complet

Enfin, il suffit d'ajouter à comp la microtexture F synthétisée avec l'ADSN sur la partie disponible de l'image masquée puis on ajoute la moyenne de l'image masquée sur cette même partie. On obtient donc une microtexture cohérente (grâce à l'ADSN) est qui est égale à l'image de base sur tout le contour (grâce à comp). Il ne reste plus qu'à combler les "trous" de l'image masquée grâce à cette microtexture synthétisée.

3.2.4 Problèmes rencontrés

Nous avons rencontré de nombreux problèmes en réalisant cette partie. Dans un premier temps nous avions un problème dans le calcul de $\Gamma_{C\times C}$ car nous faisions : $c_v(abs(j-i), abs(l-k))$ car nous nous représentions c_v comme une image du plan Ω et donc nous ne voulions pas aller dans les indices négatifs, cependant c'était une erreur car notre implémentation revenait à symétriser c_v avant de le périodiser au lieu de seulement le périodiser si on enlève les valeurs absolue.
Le second problème que nous avons recontré été au sujet du calcul même de c_v pour les images en couleur. Les images en niveau de gris avaient un inpainting très bon mais les images en couleur ne fonctionnaient pas bien. On a identifié que le problème venait probablement de c_v car les sous matrices de taille (d,d) n'était pas symétriques alors que la covariance intercanal devrait être identique pour (0,1) et (1,0) par exemple. Ce problème venait sûrement d'approximations dans la transformée de Fourier, nous avons décidé de les symétriser manuellement en calculant uniquement la matrice triangulaire supérieur et en imposant l'inférieure. Cette méthode a très bien fonctionné et nous avons ainsi eu des inpaintings cohérents, cependant nous avons perdu la symétrie de $\Gamma_{C\times C}$ en faisant cela mais ça ne semble pas avoir d'impact sur le résultat.

4 Résultats

Nous obtenons finalement le modèle voulu après avoir calculé la matrice de covariance qui nous a permis d'appliquer Λ. La complétion de nos images masquées trouve alors de la continuité aux bords du masque. Il est dans l'objet de cette partie de discuter des résultats obtenus grâce au modèle final, c'est à dire si on a bien de la continuité aux bords, si la synthèse coïncide bien avec le reste de la micro-texture etc. Enfin on pourra voir que différents facteurs entrent en compte quant à la qualité et l'efficacité de l'inpainting.
Tout d'abord voici quelques micro-textures que nous avons inpaintés :
Nous voyons donc que le résultat est satisfaisant car on ne peut pas distinguer le carré qui était la forme de notre masque. Afin de quantifier l'erreur sur la continuité au bord de notre masque nous avons calculé la MSE au contour entre l'image originale et notre micro-texture synthétisée. En calculant la moyenne des différences entre canaux linéarisés au carré on obtient la MSE qui est de l'ordre de 10^{-21} pour nos résultats, ce qui est très satisfaisant. Intéressons nous

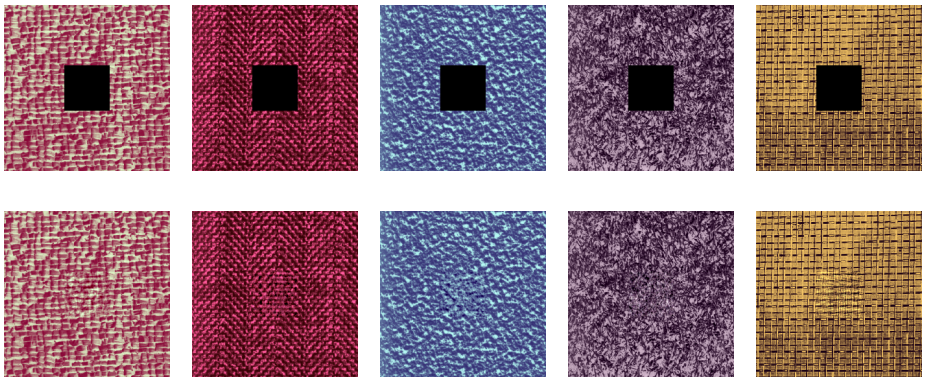


Figure 20: Resultats d'inpainting avec masque carré

alors aux différents types de facteurs qui influent sur nos résultats.

4.1 Type de texture

Tout d'abord nous avons remarqué plutôt tôt dans notre démarche qu'il faut expérimenter qu'avec des micro-textures. En effet certaines images sur lequelles nous travaillions ne pouvaient pas être considérées comme micro-texture car elles présentaient trop de régularité ou de motifs géométriques. On peut voir que la méthode d'inpainting que nous avons implémenté est inefficace pour ce genre de textures comme le montre la figure 18.

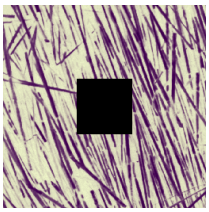


Figure 21: Mauvaise texture

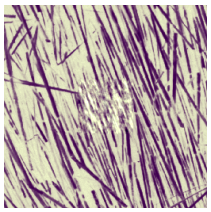


Figure 22: Inpainting

On voit que le centre de l'image est un peu trop saturé et que les formes géométriques présente dans l'image originale, c'est à dire les longues lignes violettes se ressèrent dans notre texture synthétisée.

4.2 Taille de la bordure

Etant donné que nous conditionnons notre projections de Kriging sur une bordure C de taille w il est naturel de se demander si w influe beaucoup sur la qualité du rendu final. En effet on se dit naïvement que plus la bordure est grande, plus on a d'information, plus la texture de complétude sera cohérente avec les bords. Cependant on voit que en prenant $w = 19$ et $w = 2$ les résultats sont tout aussi satisfaisants.

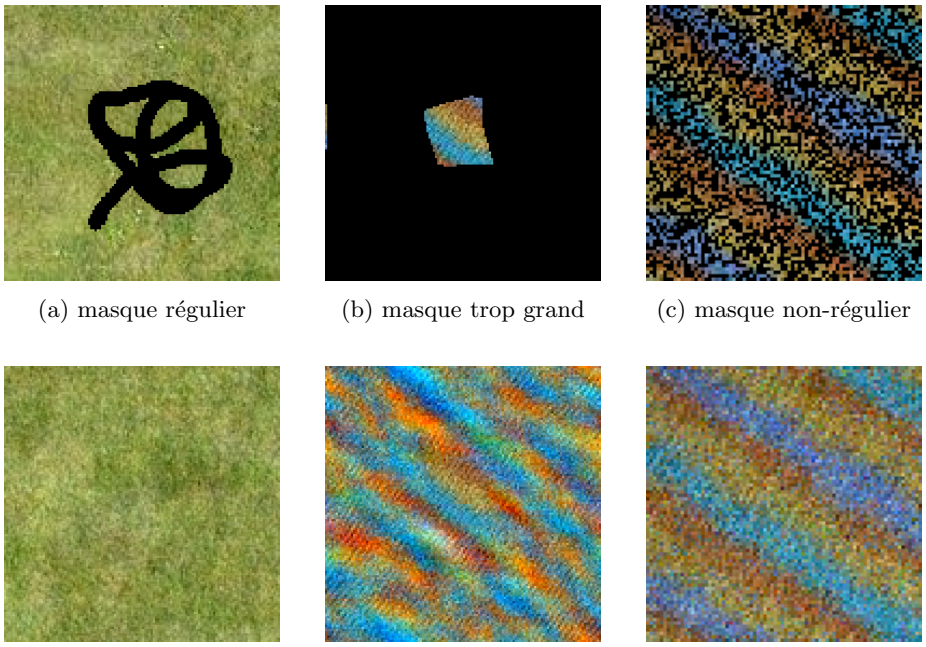
Figure 23: $w = 19$

Figure 24: $w = 2$

Ainsi, la seule différence entre ces deux tailles de bordure est le temps de calcule qui est bien plus important pour $w = 19$ que pour $w = 2$.

4.3 Type de masque

Enfin nous avons pu tester différents types de masques grâce à l'implémentation que nous avons faite dans le notebook. Nous avons vu que les masques les plus réguliers et suffisamment petits pour laisser une grande part de l'information permettent un meilleur inpainting.



(a) masque régulier (b) masque trop grand (c) masque non-régulier

(d) Inpainting correct (e) Inpainting raté (f) Effet bruité

Figure 25: Inpainting en fonction de la nature du masque

5 Conclusion

Nous avons finalement pu implémenter l'algorithme d'inpainting de micro-texture et obtenir des résultats plutôt satsfaisants. Le projet nous obligeait à faire des aller-retours entre théorie et pratique car les objets mathématiques devaient être compris afin de pouvoir même tester la validité de nos implémentations (par exemple le caractère symétrique définit positif de la matrice de covariance). Les difficultés étaient multiples et il était compliqué de faire des tests car les objets avaient des dimensions que nous n'avions pas rencontré auparavant. Il fallait donc adapter nos méthodes et nous avons passé une partie conséquente du temps à débugger notre programme.

Pour ce qu'il en est des améliorations que nous aurions voulu implémenter; nous avions remarqué lors de nos tests que lorsque nous voulions faire de l'inpainting sur des images plus grandes, le temps de calcul de l'inverse de la matrice $\Gamma|_{C\times C}$ était bien plus important. En effet pour une image de taille $600\times600\times3$ on doit attendre une minute avant d'avoir un résultat alors que pour une image de taille $128\times128\times3$ ce n'est que 5 secondes. Ce problème d'optimisation de temps aurait pu être approché avec la méthode de descente de gradient comme le développe le papier, mais nous n'avons malheureusement pas eu le temps de nous y intéresser. De plus il aurait été pertinent pour nous de pouvoir implémenter la distance Optimal Transport afin de pouvoir comparer nos modèles d'inpainting. Finalement

ce projet nous a permis d'en apprendre plus sur un domaine qui nous intéresse, de progresser dans la lecture de papiers de recherche, de mieux manipuler des objets mathématiques et leur implémentation. On remercie Arthur Leclaire pour ses explications sur ce sujet et d'avoir répondu aux maintes questions que nous lui avons posé.