Université de Montpellier
Faculté des sciences

Master 2 - IMAGINE

Projet Image, sécurité et deep learning

Sujet 12 - Débruitage ou restauration d'images par CNN

CR6 - Avancées globales et optimisation des différents modèles

[HAI918I]

BES Jean-Baptiste
COMBOT Evan

24/11/2024

Lien GitHub de notre projet image

1 Avancées de la méthode hybride

Cette semaine, nous avons entrainé et testé le modèle VAE afin de générer le masque binaire pour notre méthode hybride. Avant l'entrainement, nous avons fait un pré-traitement de la banque d'images (2000 images). Nous avons généré aléatoirement des dégradations dans les images et nous avons sauvegardé à coté ces dégradations comme des masque binaire afin d'avoir une vérité de terrain

Nous avons essayé trois méthodes de génération de dégradations.

1.1 Dégradations linéaires

Pour cette méthode, des segments horizontaux, verticaux et diagonaux sont tracés entre deux points selectionnés aléatoirement dans l'image et avec des largeurs, elles aussi, aléatoires.

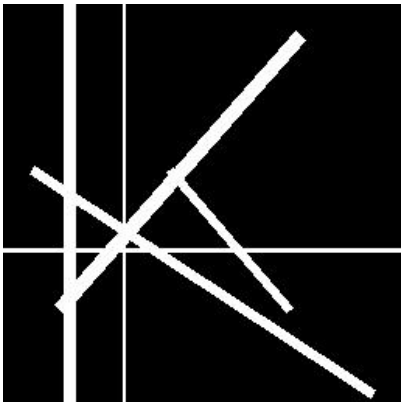

FIGURE 1 – Masque généré (exemple 1)

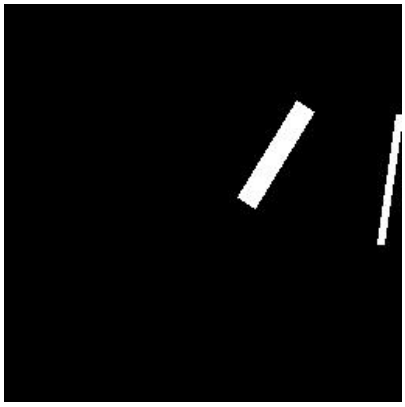

FIGURE 2 – Masque généré (exemple 2)

Nous avons fait deux entrainements chacun sur 100 époques avec des tailles de batch différent.

Entrainement avec un batch de taille 16

FIGURE 3 – Image originale

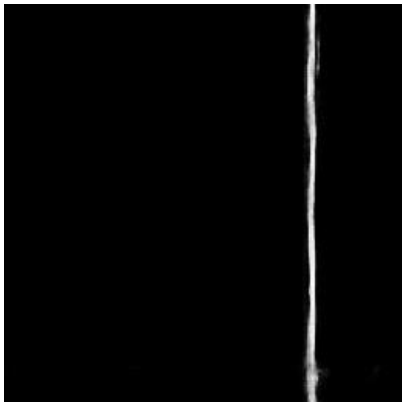

FIGURE 4 – Masque obtenu (batch taille 16)

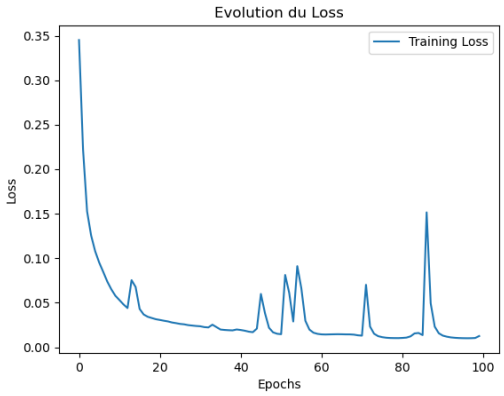

FIGURE 5 – Evolution de la fonction de perte (batch taille 16)

Entrainement avec un batch de taille 64

FIGURE 6 – Image originale

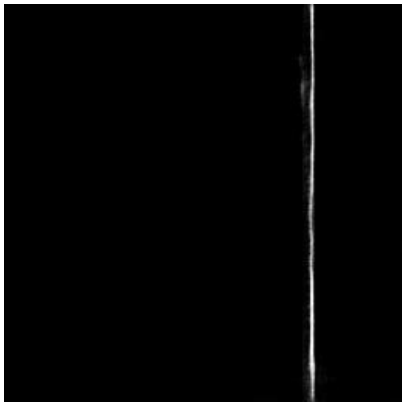

FIGURE 7 – Masque obtenu (batch taille 64)

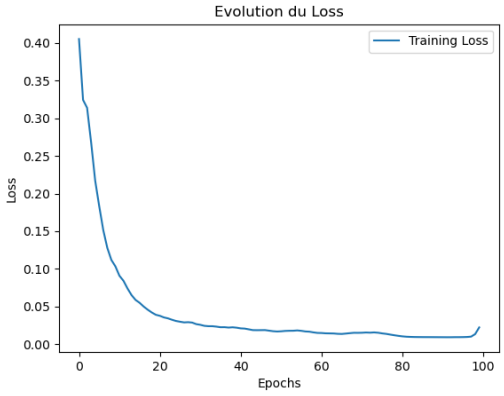

FIGURE 8 – Evolution de la fonction de perte (batch taille 64)

Nous remarquons qu'après ces deux entrainements nous obtenons des résultats similaires. Le modèle ne détecte uniquement la dégradation verticale épaisse et pas les dégradations discontinues ou celles ayant des formes spécifiques, comme des tâches. Donc nous avons voulu modifier la génération des dégradations en rajoutant de la discontinuité.

1.2 Dégradations non-linéaires

Pour cette méthode, nous avons réutilisé le même procédé que pour la première méthode mais en découpant le segment en plusieurs sous-segment et en applicant une translation / rotation à chacun de ces sous-segments.

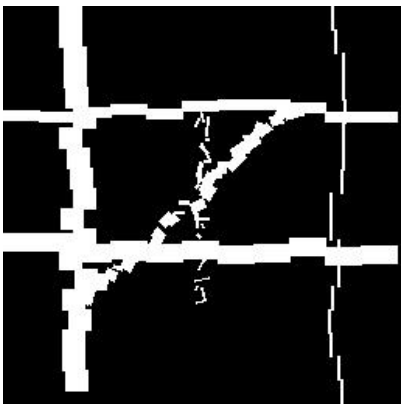

FIGURE 9 – Masque généré (exemple 1)

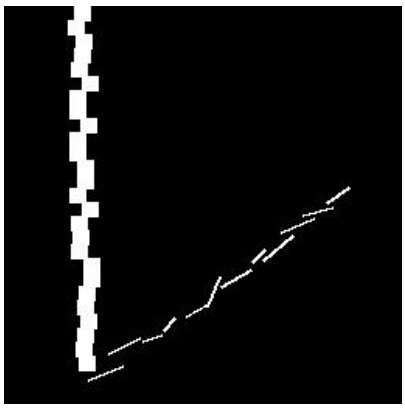

FIGURE 10 – Masque généré (exemple 2)

Nous avons fait les mêmes tests que pour la méthode 1 :

Entrainement avec un batch de taille 16

FIGURE 11 – Image originale

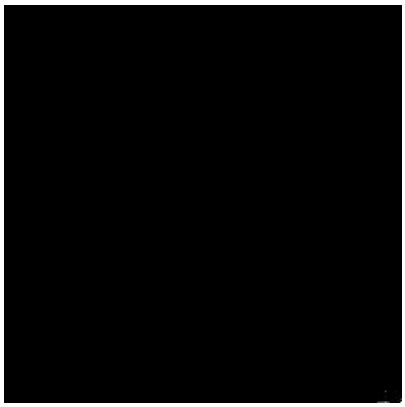

FIGURE 12 – Masque obtenu (batch taille 16)

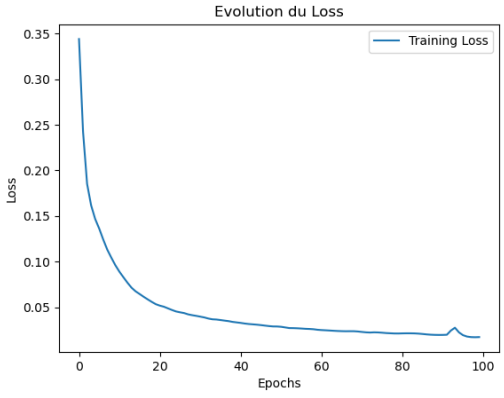

FIGURE 13 – Evolution de la fonction de perte (batch taille 16)

Entrainement avec un batch de taille 64

FIGURE 14 – Image originale

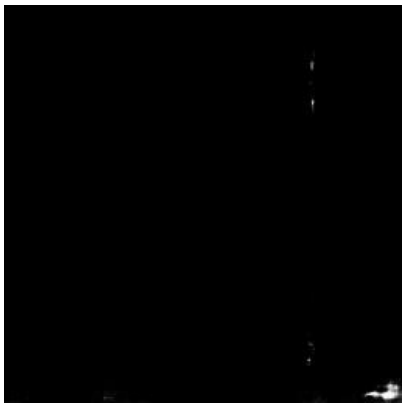

FIGURE 15 – Masque obtenu (batch taille 64)

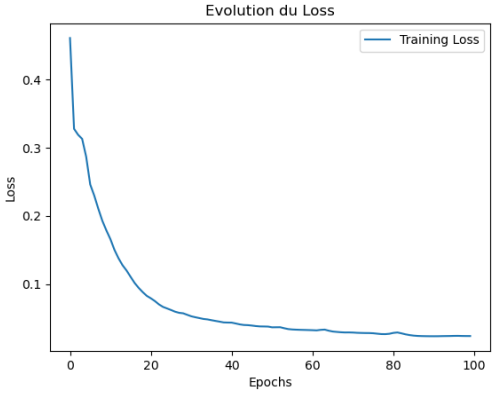

FIGURE 16 – Evolution de la fonction de perte (batch taille 64)

Nous remarquons une baisse de performance, même s'il y a un début de détection de la dégradation dans le coin en bas à droit pour le batch de taille 64. Mais avec l'étude de l'évolution de la fonction de perte on constate que le modèle ne pourra plus s'améliorer. Donc nous avons décider de tester une dernière méthode.

1.3 Dégradations courbées

Pour cette méthode, nous avons amélioré la génération des dégradations utilisée dans la méthode 2. Chaque sous-segment est généré en utilisant le point de fin du sous-segment précédent comme point de départ.

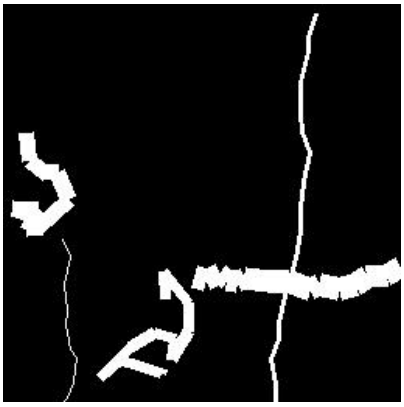

FIGURE 17 – Masque généré (exemple 1)

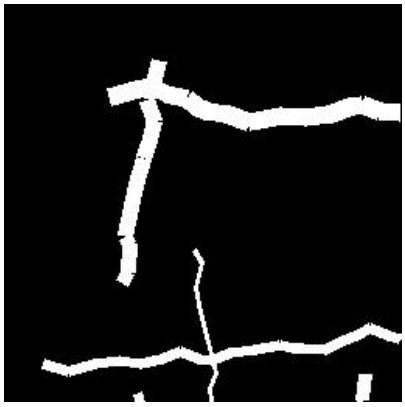

FIGURE 18 – Masque généré (exemple 2)

Nous avons fait les mêmes tests que pour les méthodes 1 et 2 :

Entrainement avec un batch de taille 16

FIGURE 19 – Image originale

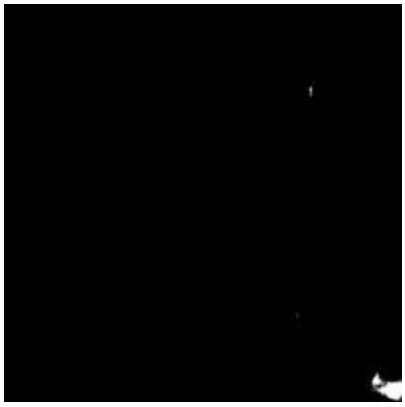

FIGURE 20 – Masque obtenu (batch taille 16)

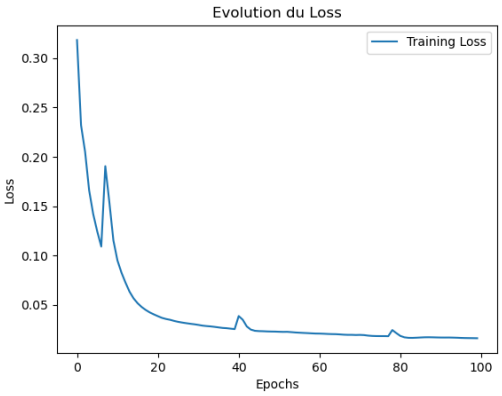

FIGURE 21 – Evolution de la fonction de perte (batch taille 16)

Entrainement avec un batch de taille 64

FIGURE 22 – Image originale

FIGURE 23 – Masque obtenu (batch taille 64)

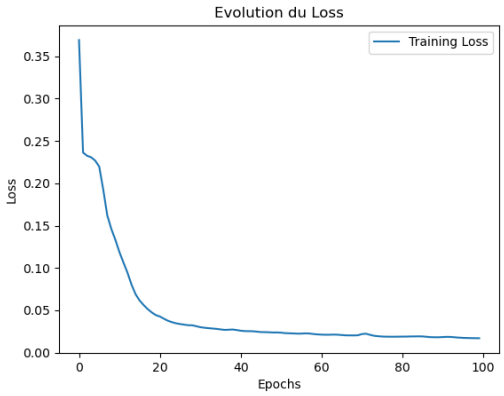

FIGURE 24 – Evolution de la fonction de perte (batch taille 64)

Nous pouvons faire le même constat que pour la deuxième méthode, avec une légère amélioration. Cependant, les résultats restent clairement insatisfaisants. Et il semblerait que ces modèles ne démontrent aucun signe d'amélioration quand nous regardons la fonction de perte à chaque époque.

1.4 Conclusions des tests

Après avoir analysé les résultats des entraînements, il est clair que nous devons changer de méthode. Nous avons décidé d'utiliser un modèle CNN basé sur un VAE car il était notre premier modèle implémenté conçu pour la restauration d'image. Cependant, dans ce contexte de méthode hybride, nous nous orientons désormais vers une problématique de segmentation où notre objectif est de dissocier notre image des dégradations afin de l'utiliser dans notre méthode classique (sans CNN). Pour cela, nous avons penser utiliser un modèle U-Net qui est conçu spécifiquement pour de la segmentation.

2 Améliorations et corrections des modèles

Nous avons corrigé les modèles depuis le dernier compte-rendu car ils présentaient des cas de surapprentissage principalement dus à un taux d'apprentissage trop élevé. Nous avons également ajouté un modèle avec une architecture U-Net. Tous les modèles sont des VAE multi-domaines.

2.1 Modèle simple

- **Encodeurs par domaine :** Chaque domaine (X, Y, Z) dispose d'un encodeur indépendant, constitué de trois couches convolutives ($32 \to 64 \to 128$) suivies de couches denses pour produire des vecteurs latents (z_mean, z_log_var).
- **Reparamétrage :** Un échantillonnage différentiable permet de générer un vecteur latent z à partir des paramètres statistiques (z_mean, z_log_var).
- **Décodeurs par domaine :** Chaque domaine possède un décodeur dédié, symétrique aux encodeurs, avec des convolutions transposées pour reconstruire les images d'origine ($256 \times 256 \times 1$).

2.1.1 1er entraînement

Voici les paramètres :

```
BATCH_SIZE = 16
LATENT_DIM = 256
EPOCHS = 10
BETA = 0.001
MAX_BETA = 1.
ANNEAL_EPOCH = 3
TAUX_APPRENTISSAGE = 1e-6
PONDERATION_X_TRAIN = 1.
PONDERATION_Y_TRAIN = 1.
PONDERATION_Z_TRAIN = 1.
```

Nous utilisons une taille de batch de 16, car une taille de 32 s'avère trop coûteuse en termes de mémoire. La dimension de l'espace latent est fixée à 256 ce qui permet d'obtenir un équilibre entre la capacité de représentation et la complexité du modèle.

Le paramètre β, qui intervient dans le calcul de la perte de reconstruction est utilisé avec une valeur maximale définie à 1. La valeur de β est augmentée progressivement selon une méthode linéaire toutes les 3 époques (*annealing*). Cela permet une montée en puissance contrôlée du terme régulier dans la perte.

Le taux d'apprentissage est fixé à 1×10^{-6}. Ce paramètre a été particulièrement critique car une valeur trop élevée (1×10^{-4} avant) entraînait une convergence trop rapide, provoquant un surapprentissage.

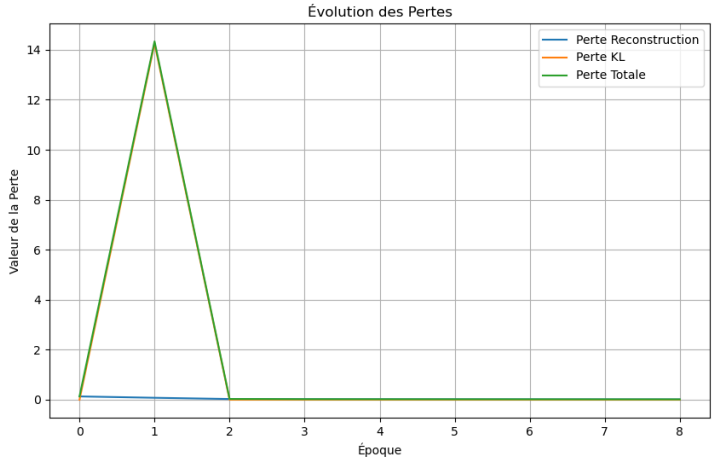

FIGURE 25 – Exemple de surapprentissage observé avec l'ancien modèle

Enfin, nous pouvons décider de prioriser un domaine spécifique dans le modèle. Les pondérations des différents domaines sont définies ici comme égales à 1 ce qui signifie qu'aucune priorité particulière n'est donnée à un domaine spécifique lors de l'entrainement.

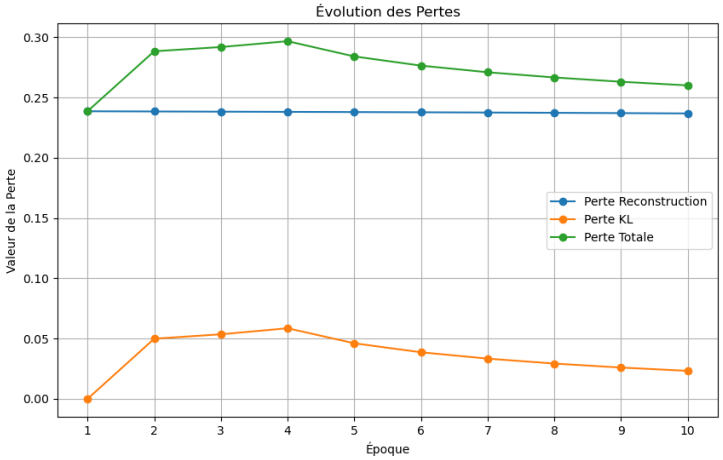

FIGURE 26 – Résultats

On observe une perte totale initiale de 0,24 à la 1ère époque. Par la suite, la divergence de Kullback-Leibler augmente d'environ 0,06 jusqu'à la 4ème époque, ce qui fait passer la perte totale à une valeur proche de 0,30. À partir de la 5ème époque, la divergence KL commence à diminuer, ce qui impacte directement la perte totale. En revanche, la perte de reconstruction reste stable tout au long des 10 époques avec une valeur approximative de 0,24. On observe que la perte totale diminue au fil des époques.

2.1.2 2ème entraînement

Voici les paramètres :

```
BATCH_SIZE = 16
LATENT_DIM = 256
EPOCHS = 30
BETA = 0.001
MAX_BETA = 1.
ANNEAL_EPOCH = 2
TAUX_APPRENTISSAGE = 1e-6
PONDERATION_X_TRAIN = 1.
PONDERATION_Y_TRAIN = 1.
PONDERATION_Z_TRAIN = 1.
```

Pour confirmer la diminution de la perte totale observée lors du premier entraînement, nous avons augmenté le nombre d'époques à 30 et défini l'intervalle de calcul de β à 2 époques.

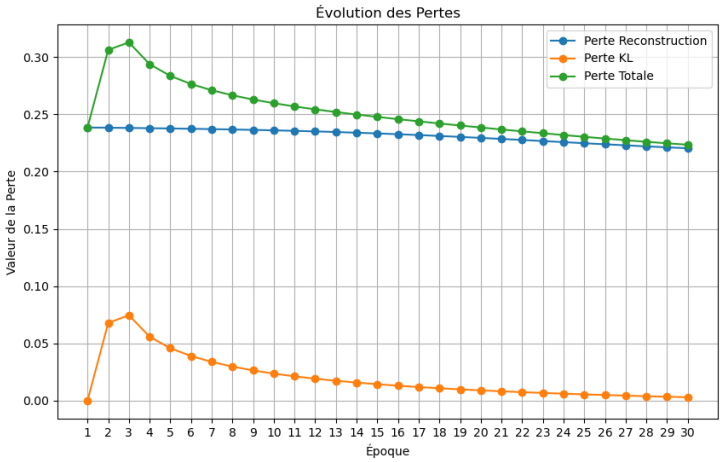

FIGURE 27 – Résultats

Comme pour le 1er entraînement, on observe que la divergence KL augmente. De plus, au bout de 30 époques, la perte totale diminue de manière significative, ce qui suggère que, si le modèle était entraîné sur un plus grand nombre d'époques, la perte totale pourrait continuer à diminuer. En revanche, nous pensons que le calcul de β toutes les 2 époques n'est pas optimal. En effet, la valeur de β semble augmenter de manière plus importante lorsque l'intervalle est faible, comparé à un intervalle plus grand. Cela pourrait indiquer qu'il serait préférable d'augmenter l'intervalle de calcul de β à 3 époques ou davantage.

Tests du modèle (restauration)

Pour obtenir un résultat satisfaisant en termes de restauration d'images, il est essentiel de bien sélectionner les données d'entrée et d'entraînement car elles influencent directement la capacité du modèle à généraliser et à restaurer efficacement des images anciennes.

1. **Domaine Z comme données d'entrée :** Le domaine Z représente les images anciennes naturellement dégradées, c'est-à-dire des images issues de conditions réelles de vieillissement, telles que la détérioration physique (usure, décoloration) ou chimique. Ces données sont utilisées comme entrées pour le modèle, car elles reflètent directement le problème réel que l'on cherche à résoudre : la restauration d'images dégradées par le temps.
2. **Raison du choix des domaines :** L'utilisation combinée des domaines Y et Z pour la restauration améliore la robustesse du modèle. Le domaine Y fournit une large gamme de dégradations synthétiques qui permettent un apprentissage général tandis que le domaine Z affine les performances pour des scénarios réels.

Nous avons utilisé le modèle entrainé sur 30 époques. Pour chercher un compromis entre le débruitage et la réparation de dégradations, on utilise une pondération de 0.5 entre les domaines Y et Z.

FIGURE 28 – Image originale

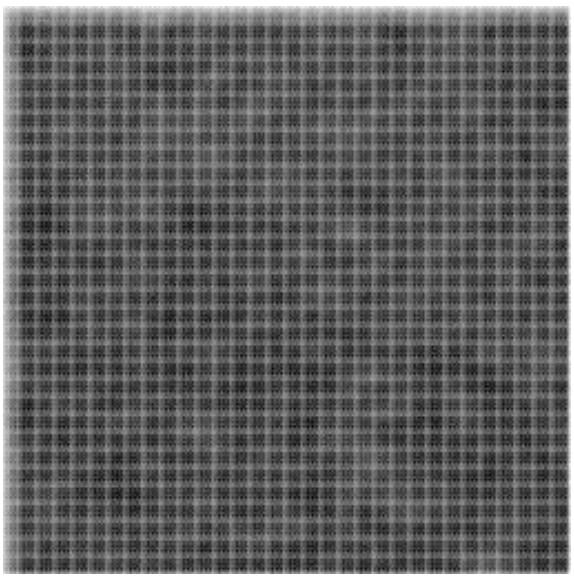

FIGURE 29 – Poids domaine Y = 0.5 / Poids domaine Z = 0.5

PSNR : 11.77 dB
SSIM : 0.3020

Peu importe la pondération attribuée, nous n'obtenons pas de résultats satisfaisants avec ce premier modèle "simpliste". Il serait nécessaire d'entraîner le modèle sur un plus grand nombre d'époques et de l'améliorer. Le quadrillage observé est dû à la taille des batchs qui est très petite ce qui rend ce phénomène visible.

2.2 Modèle avec une architecture U-Net

- **Encodeurs par domaine :** Chaque domaine (X, Y, Z) dispose d'un encodeur indépendant, constitué de trois couches convolutives ($32 \to 64 \to 128$) avec activations ReLU et strides de 2, suivies de couches denses produisant les paramètres latents (z_mean, z_log_var). Les connexions de saut (*skip connections*) sont conservées pour le décodeur.
- **Reparamétrage :** Un échantillonnage différentiable est appliqué pour générer un vecteur latent $\mathbf{z}$, défini comme :

$$\mathbf{z} = \mathbf{z\_mean} + \exp\left(0.5 \cdot \mathbf{z\_log\_var}\right) \cdot \epsilon,$$

  où ϵ est un bruit gaussien.
- **Décodeurs par domaine :** Chaque domaine possède un décodeur dédié qui utilise :
  - Une couche dense suivie d'un `reshape` pour recréer une carte spatiale initiale ($16 \times 16 \times 128$).
  - Trois convolutions transposées ($128 \to 64 \to 32$) avec *skip connections* pour combiner les informations des encodeurs.
  - Une couche finale reconstruisant l'image d'origine ($256 \times 256 \times 1$) avec une activation sigmoid.

2.2.1 1er entraînement

Voici les paramètres :

```
BATCH_SIZE = 32
LATENT_DIM = 256
EPOCHS = 10
BETA = 0.001
MAX_BETA = 1.
ANNEAL_EPOCH = 3
TAUX_APPRENTISSAGE = 1e-6
PONDERATION_X_TRAIN = 1.
PONDERATION_Y_TRAIN = 1.
PONDERATION_Z_TRAIN = 1.
```

Les paramètres sont les mêmes que pour le premier modèle à l'exception de la taille des batchs, qui est ici augmentée à 32.

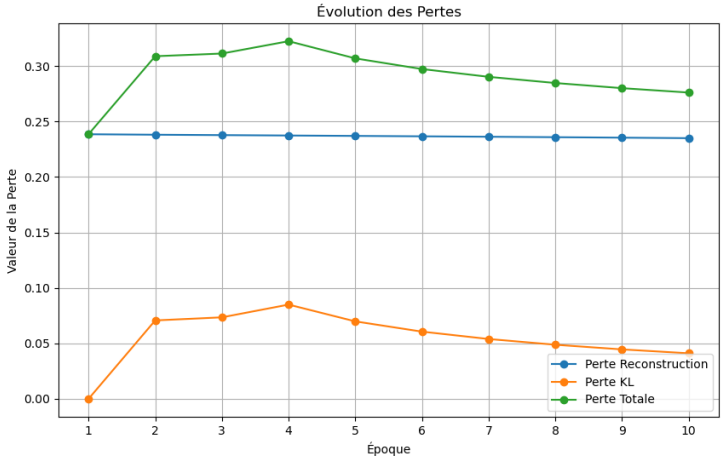

FIGURE 30 – Résultats

On remarque que le graphique suit la même tendance que pour le 1er modèle.

2.2.2 2ème entraînement

Voici les paramètres :

```
BATCH_SIZE = 32
LATENT_DIM = 256
EPOCHS = 10
BETA = 0.001
MAX_BETA = 1.
ANNEAL_EPOCH = 2
TAUX_APPRENTISSAGE = 1e-6
PONDERATION_X_TRAIN = 1.
PONDERATION_Y_TRAIN = 1.
PONDERATION_Z_TRAIN = 1.
```

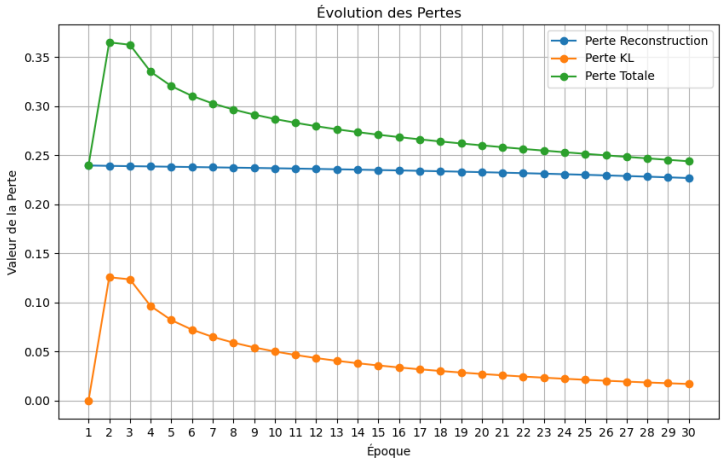

FIGURE 31 – Résultats

Comme on pouvait s'y attendre, le graphique suit également la même tendance que pour le 1er modèle.

Tests du modèle (restauration)

Nous avons utilisé le modèle entrainé sur 30 époques. Pour obtenir un compromis entre la réparation des dégradations et le débruitage, il faut utiliser une pondération de 0.5 entre les domaines Y et Z.

FIGURE 32 – Image originale

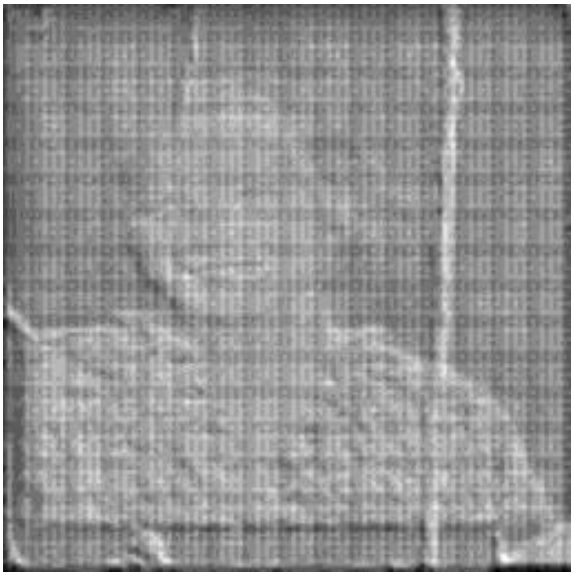

FIGURE 33 – Poids domaine Y = 0.5 / Poids domaine Z = 0.5

PSNR : 10.87 dB
SSIM : 0.2913

Pour privilégier la réparation des dégradations, il faut donner une pondération plus importante au domaine Z.

FIGURE 34 – Image originale

FIGURE 35 – Poids domaine Y = 0.3 / Poids domaine Z = 0.7

PSNR : 10.92 dB
SSIM : 0.2920

Pour privilégier le débruitage, il faut donner une pondération plus importante au domaine Y.

FIGURE 36 – Image originale

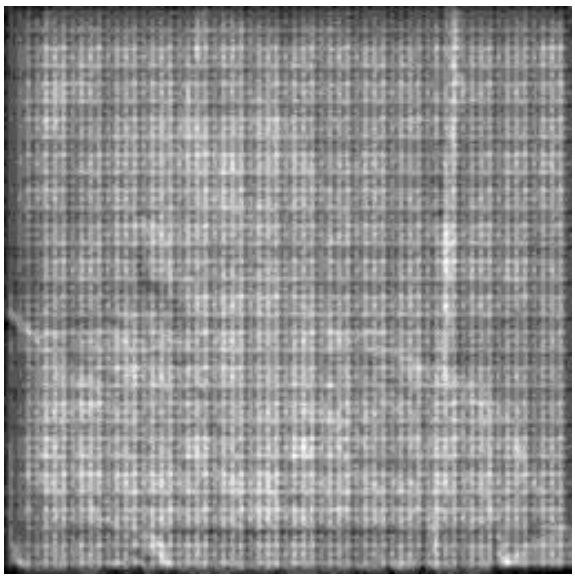

FIGURE 37 – Poids domaine Y = 0.7 / Poids domaine Z = 0.3

PSNR : 10.81 dB
SSIM : 0.2901

Avant d'ajouter des images anciennes naturellement dégradées, nous obtenions des résultats plus propres. Nous allons analyser le problème pour déterminer si une simple pondération lors de l'entraînement suffirait, ou s'il serait nécessaire de réduire le nombre d'images anciennes dégradées dans le dataset. Cependant, les résultats obtenus restent très cohérents en fonction de la pondération attribuée à chaque domaine lors de la restauration, ce qui nous satisfait pleinement. Le plus dur est de trouver le juste millieu pour les données synthéthiques afin qu'elles ne soient pas trop bruitées et en même temps qu'elles ne soient également pas trop dégradées afin que le modèle ne privilégie l'un ou l'autre.

2.3 Modèle avec des ResBlocks

- **Blocs résiduels (ResBlocks) :** Les blocs résiduels ajoutés dans les encodeurs et les décodeurs améliorent la capacité du modèle à capturer des relations complexes. Chaque bloc comprend :
  - Une connexion shortcut qui adapte les dimensions si nécessaire.
  - Deux convolutions principales avec des activations ReLU, suivies d'une addition avec le shortcut.
  - Une activation finale ReLU pour la sortie.
- **Encodeurs par domaine :** Chaque domaine (X, Y, Z) dispose d'un encodeur distinct qui suit cette structure :
  - Trois couches convolutives ($32 \to 64 \to 128$) avec strides de 2.
  - Une étape de *flattening* pour réduire les dimensions.
  - Calcul des vecteurs latents **z_mean** et **z_log_var** à l'aide de couches denses.
  - Connexions de saut (*skip*) pour les résolutions intermédiaires ($x1, x2, x3$).
- **Reparamétrage :** Le vecteur latent **z** est échantillonné de manière différentiable à l'aide de la formule :
$$\mathbf{z} = \mathbf{z\_mean} + \exp(0.5 \cdot \mathbf{z\_log\_var}) \cdot \epsilon,$$
où ϵ est un bruit gaussien.
- **Mappage latent (`mapping_t`) :** Une étape supplémentaire reformate les vecteurs latents pour les transformer en une carte spatiale ($16 \times 16 \times 128$). Cette carte passe à travers trois ResBlocks pour capturer des interactions complexes avant d'être envoyée au décodeur.
- **Décodeurs par domaine :** Les décodeurs, symétriques aux encodeurs, utilisent les connexions de saut et les ResBlocks pour reconstruire les images originales. Leur structure inclut :
  - Une carte initiale obtenue à partir de **z** ($16 \times 16 \times 128$).
  - Trois convolutions transposées ($128 \to 64 \to 32$) avec connexions de saut.

— Une dernière convolution transposée pour produire l'image de sortie ($256 \times 256 \times 1$).

Le modèle, étant trop gourmand en mémoire GPU, que ce soit chez nous ou à l'Université, nécessite une réflexion sur son allègement sans qu'il devienne inefficace.

2.4 D'autres entraînements

Nous avons testé d'autres optimiseurs dans le but d'évaluer leur potentiel à améliorer nos résultats.

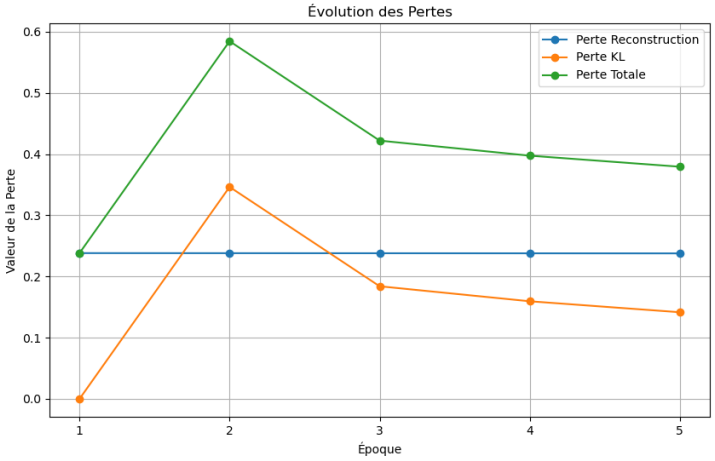

FIGURE 38 – Optimiseur : Adamax

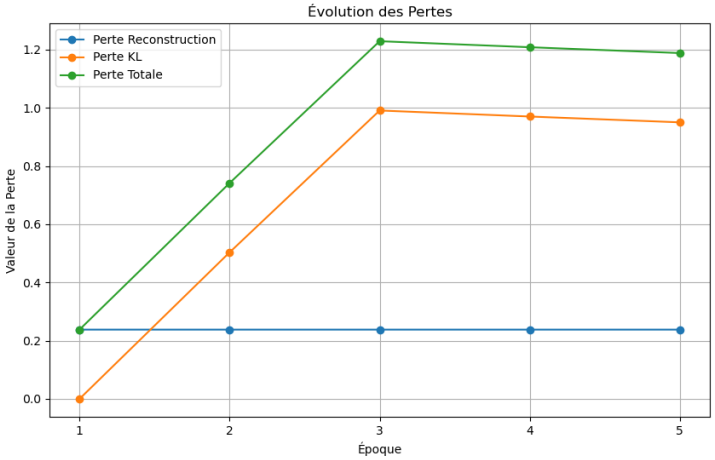

FIGURE 39 – Optimiseur : SGD

D'après les graphiques obtenus, l'optimiseur Adamax semble plus prometteur que SGD. Cependant, pour les deux optimiseurs, la perte totale augmente considérablement dès les premières époques, tandis que la reconstruction totale ne s'améliore que très peu par la suite. Bien qu'Adamax paraisse meilleur que SGD, aucun des deux optimiseurs ne semble viable en raison de cette forte augmentation de la perte totale qui ne diminue que marginalement par la suite. L'optimiseur privilégié reste Adam utilisé pour les tests précédents.

2.5 Les améliorations possibles

D'après nos tests, plusieurs améliorations ou modifications sont envisageables :

— Amélioration du prétraitement des données.

— Tests de différents optimiseurs.

— Ajout de couches de convolution (évaluer les effets sur le modèle simple et, si les résultats sont prometteurs, appliquer cette modification aux autres modèles).
— Modification du mode de calcul de β (explorer un compromis entre la divergence KL et la perte de reconstruction).
— Ajustement des fonctions d'activation (par exemple, remplacer ReLU par LeakyReLU, etc.).

3 Avancées de l'application

Nous avons amélioré notre application afin de permettre la sélection de plusieurs modèles. Pour cela, l'application exécute directement les fonctions de génération d'images des scripts Python associés à chaque modèle. Ces scripts prennent en entrée un dossier contenant les images à restaurer et génèrent un dossier contenant les images restaurées. Ensuite, l'application récupère les images restaurées et les affiche à côté de leurs images d'origine. L'utilisateur peut finalement naviguer entre les images générées à l'aide des deux boutons situés en bas de la fenêtre.

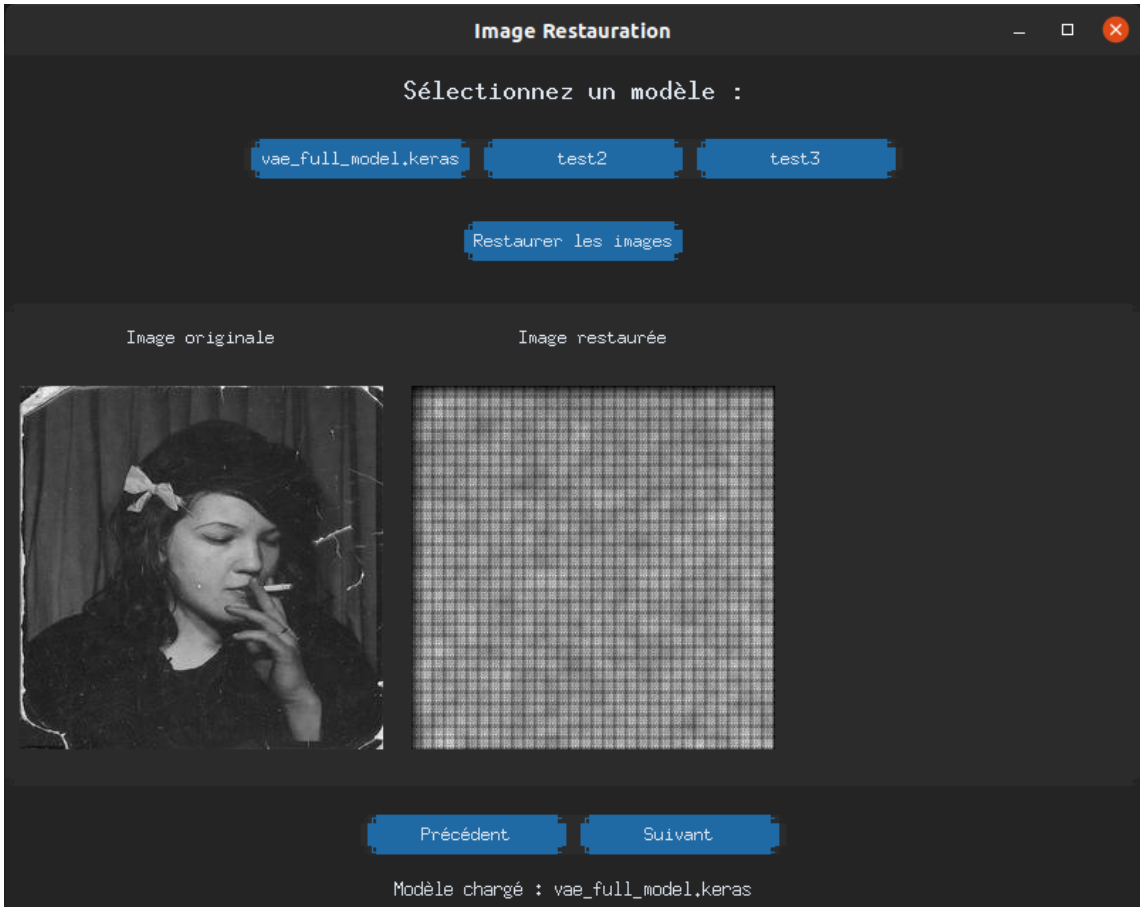

FIGURE 40 – Interface de l'application

4 Nos prochaines avancées

Nous allons optimiser les modèles de restauration, tester le modèle U-Net pour la génération de masque et nous allons essayer de finaliser l'application.