Université de Montpellier
Faculté des sciences

Master 2 - IMAGINE

Projet Image, sécurité et deep learning

Sujet 12 - Débruitage ou restauration d'images par CNN

CR5 - Modèles de restauration et modèles de génération automatique de masques binaires

`[HAI918I]`

BES Jean-Baptiste
COMBOT Evan

17/11/2024

Lien GitHub de notre projet image

1 Génération automatique du masque binaire : méthode hybride

Après notre présentation de la semaine dernière, on nous a suggéré d'utiliser notre CNN pour générer le masque binaire qui est utilisé dans la phase d'inpainting de la méthode sans CNN. Nous avons donc décidé de tester cette approche qui consiste à combiner notre méthode d'inpainting classique avec un masque binaire généré avec un CNN.

Pour cela, nous avons réutilisé la structure de notre VAE. Nous l'avons entraîné sur une banque d'images (1000 dans ce cas) qui ont subi une dégradation artificielle. En parallèle, nous avons stocké les dégradations qui ont été appliquées afin qu'elles servent de vérité de terrain.

Nous n'avons pas encore pu entraîner correctement ce modèle, donc on n'obtient pas de résultats satisfaisants.

FIGURE 1 – Image d'entrée

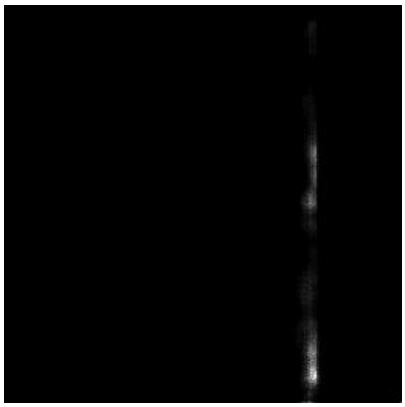

FIGURE 2 – Masque obtenu

Nous avons obtenu ce résultat après avoir fait un entrainement sur 10 époques et avec un batch de taille 16.

2 Modèles de restauration avec VAEs

Comme nous l'avons dit dans le précédent compte-rendu, nous avons mis en place des modèles de restauration utilisant des VAEs.

2.1 Premier modèle : simplifié

Ce 1er modèle est un VAE multi-domaines conçu pour traiter plusieurs domaines de données, ici représentés par des photos synthétisées (domaine X) représentant des images dégradées et bruitées à partir d'images modernes du domaine R, des photos réelles (domaine R) qui proviennent du dataset **Pascal VOC 2012** et des photos anciennes (domaine Y) qui proviennent du dataset **Dating Historical Color Images**.

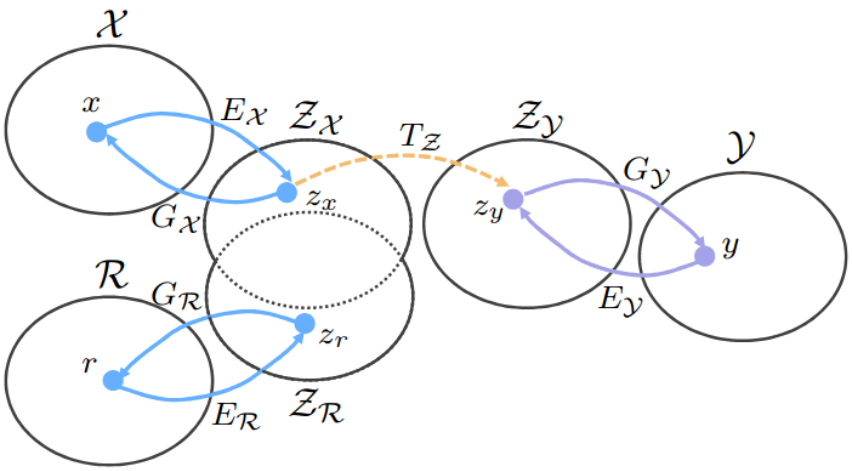

FIGURE 3 – Multi domaines

2.1.1 Variationnal Auto-Encoders (VAE)

Un VAE est un type de réseau neuronal génératif qui encode des données dans un espace latent continu et apprend à générer des échantillons similaires. Contrairement à un autoencodeur classique, il impose une distribution probabiliste sur l'espace latent, souvent une distribution normale.

- **Encodeur** : Mappe les données d'entrée dans un espace latent. Les sorties principales sont :
  - z_{mean} : La moyenne de la distribution latente.
  - $z_{\text{log\_var}}$: Le logarithme de la variance de la distribution latente.
  - z : L'échantillon tiré de cette distribution.
- **Décodeur** : Reconstruit les données d'origine à partir des échantillons de l'espace latent z.

Divergence et perte de reconstruction :

- **Perte de reconstruction** : Compare les données reconstruites et originales (souvent avec une perte de type cross-entropie binaire).
- **Divergence KL (Kullback-Leibler)** : Implique une régularisation pour rapprocher z_{mean} et $z_{\text{log\_var}}$ d'une distribution normale standard $\mathcal{N}(0, 1)$.

$$\text{KL loss} = -0.5 \sum \left(1 + z_{\text{log\_var}} - z_{\text{mean}}^2 - \exp(z_{\text{log\_var}})\right)$$

- **Perte totale** : Addition des deux.

2.1.2 Architecture du VAE multi-domaines

Ce modèle utilise des encodeurs et décodeurs indépendants pour chaque domaine. Cela permet d'apprendre des représentations spécifiques à chaque domaine tout en partageant une architecture similaire.

- **Encodeurs** : Chaque encodeur extrait des représentations latentes $(z_{\text{mean}}, z_{\text{log\_var}}, z)$ pour un type d'image donné. Ils suivent une architecture typique de convolution (4 couches de convolutions avec activation ReLU et stride de 2) suivie d'une projection dense dans l'espace latent.
- **Décodeurs** : Chaque décodeur reconstruit une image à partir de la représentation latente. Il utilise des couches de convolutions transposées pour redimensionner et reconstruire les images.

2.1.3 Résultats

Notre but ici est d'analyser les meilleurs hyperparamètres qui permettent d'obtenir une perte de reconstruction et une perte totale le plus bas possible.

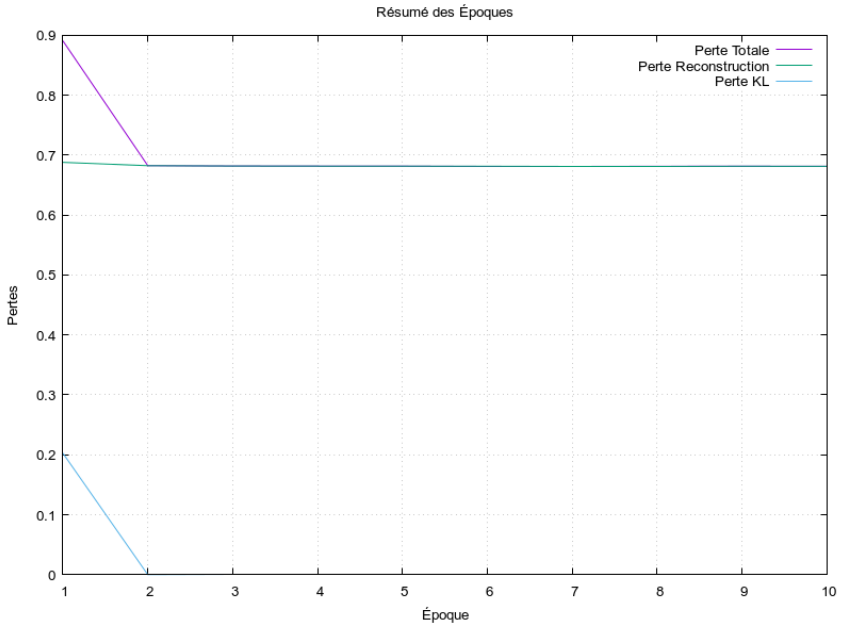

FIGURE 4 – Optimiseur : Adam / Nombre d'epochs : 10 / Adaptation automatique du taux d'apprentissage : Non / Taille du batch : 32

Avec l'optimiseur Adam, on observe qu'après seulement 2 époques, la perte totale converge rapidement et s'aligne avec la perte de reconstruction. La perte KL diminue drastiquement dès la première époque devenant négligeable par la suite. Cependant, après la 2ème époque, la perte totale diminue très lentement ce qui indique une stagnation mais en réalité, la perte totale diminue tout de même mais très lentement.

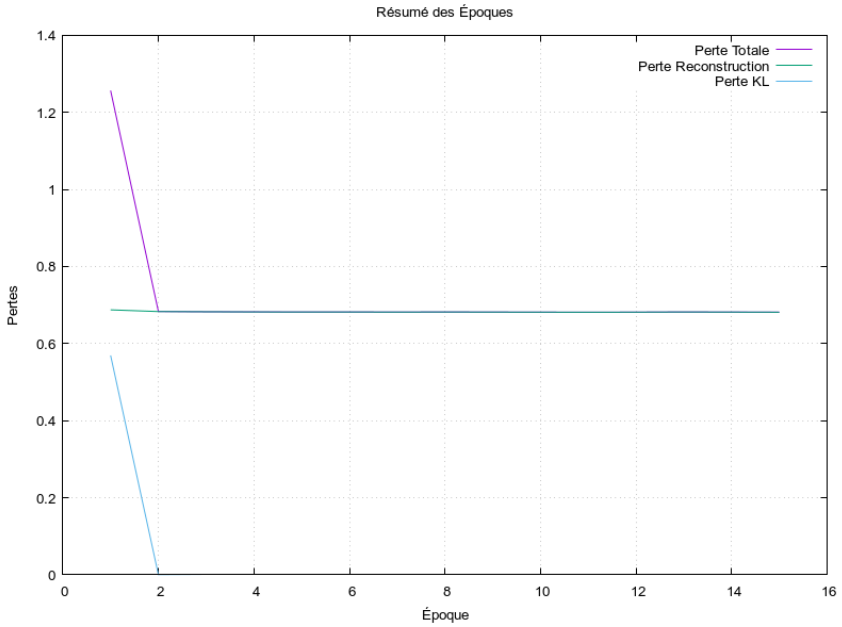

FIGURE 5 – Optimiseur : Adamax / Nombre d'epochs : 15 / Adaptation automatique du taux d'apprentissage : Non / Taille du batch : 32

L'optimiseur Adamax présente un comportement similaire à celui d'Adam. La perte totale converge et s'aligne avec la perte de reconstruction dès la 2ème époque tandis que la perte KL devient presque nulle. Cependant, même avec 15 époques, la diminution de la perte totale reste lente.

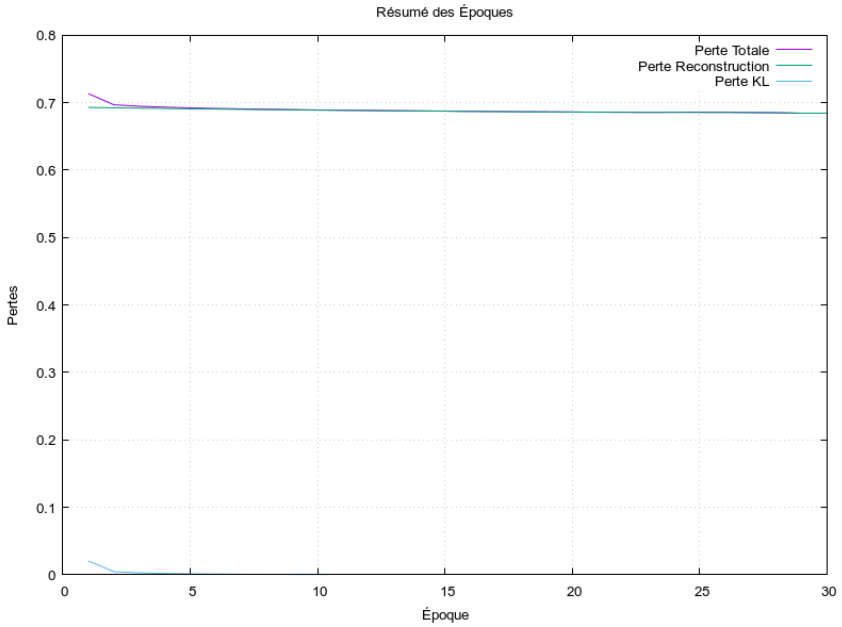

FIGURE 6 – Optimiseur : SGD / Nombre d'epochs : 30 / Adaptation automatique du taux d'apprentissage : Non / Taille du batch : 32

Dans cette configuration, l'optimiseur SGD converge lentement sur 30 époques. La perte totale reste relativement élevée tout au long de l'entraînement, bien que l'on observe une légère diminution progressive. La perte de reconstruction domine et converge très lentement, tandis que la perte KL devient négligeable dès les premières époques.

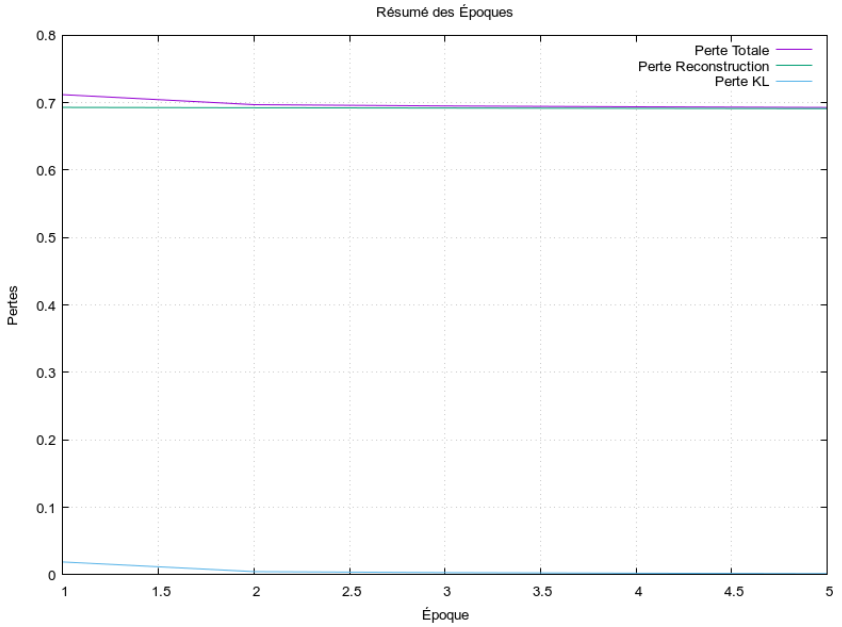

FIGURE 7 – Optimiseur : SGD / Nombre d'epochs : 5 / Adaptation automatique du taux d'apprentissage : Oui / Taille du batch : 32

Avec l'optimiseur SGD et une adaptation automatique du taux d'apprentissage, on observe une convergence rapide en seulement 5 époques. La perte totale diminue rapidement pour s'aligner avec la perte de reconstruction, tandis que la perte KL reste négligeable.

— Les optimiseurs **Adam** et **Adamax** offrent une convergence rapide dès les premières époques mais montrent une stagnation par la suite.
— L'optimiseur **SGD**, sans adaptation, converge très lentement et nécessite un grand nombre d'époques, ce qui le rend moins efficace dans ces conditions.

— L'utilisation de **SGD** avec une adaptation automatique du taux d'apprentissage réduit drastiquement le nombre d'époques nécessaires et offre une performance similaire aux optimiseurs Adam et Adamax.
— La perte **KL** devient négligeable dès les premières époques dans toutes les configurations laissant la perte de reconstruction dominer la perte totale.

2.2 Deuxième modèle : complet

Ce modèle utilise le modèle simplifié précédent et introduit les ResNet dans les encodeurs et décodeurs pour exploiter les chemins de saut (skip connections) qui facilitent l'apprentissage en profondeur.

2.2.1 Les blocs ResNet

Un bloc ResNet ajoute un chemin de saut entre l'entrée et la sortie pour résoudre le problème de dégradation lors de l'entraînement de réseaux profonds.

Etapes :

1. L'entrée subit deux convolutions (avec normalisation par lot et activation ReLU).
2. Un chemin de saut applique une convolution 1×1 pour ajuster les dimensions, si nécessaire.
3. La sortie de la convolution principale est ajoutée à celle du chemin de saut.
4. Une activation ReLU est appliquée à la somme.

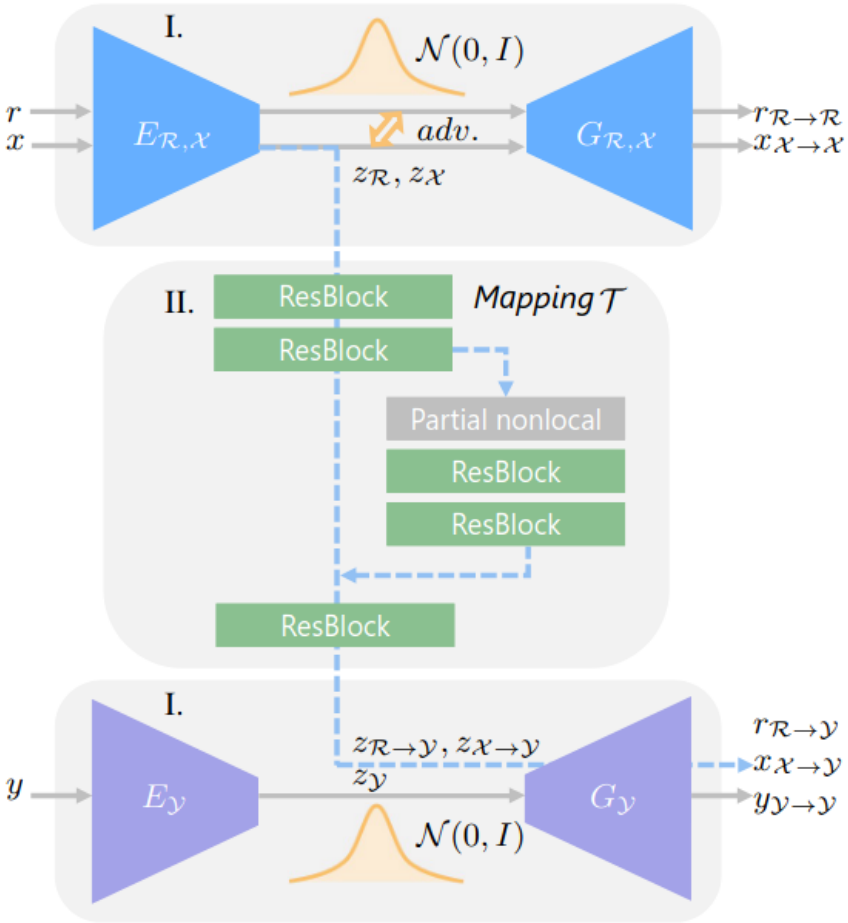

FIGURE 8 – Architecture du modèle complet

2.2.2 Résultats

Nos ordinateurs n'étant pas suffisamment puissants pour entraîner ce modèle, nous envisageons de l'entraîner à l'Université. Si nous constatons que le modèle est trop lourd, nous devrons envisager de le simplifier.

3 Modèle de colorisation

Nous avons commencé à mettre en place un modèle de colorisation basé sur l'architecture U-Net, qui est optionelle dans le processus de restauration. Nous voulons mettre en place, si nous avons le temps d'implémenter la colorisation à partir de ce document : [1]

3.1 Conversion de l'espace RGB vers XYZ

La conversion des valeurs RGB en valeurs XYZ est réalisée avec l'équation suivante :

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

où :

$$R, G, B \in [0, 1], \quad X \in [0, 0.95], \quad Y \in [0, 1], \quad Z \in [0, 1.089].$$

La normalisation des composantes X et Z se fait comme suit :

$$X = \frac{X}{0.95}, \quad Z = \frac{Z}{1.089}.$$

3.2 Conversion de l'espace XYZ vers Lab

Les composantes L, a, b dans l'espace Lab sont définies par les équations suivantes :

Luminance (L) :

$$L = \begin{cases} 116 \cdot Y^{\frac{1}{3}} - 16, & \text{si } Y > 0.008856, \\ 903.3 \cdot Y, & \text{sinon.} \end{cases}$$

Composantes a et b :

$$a = 500 \cdot (f(X) - f(Y)), \quad b = 200 \cdot (f(Y) - f(Z)),$$

où :

$$f(\text{Composante}) = \begin{cases} \text{Composante}^{\frac{1}{3}}, & \text{si Composante} > 0.008856, \\ 7.787 \cdot \text{Composante} + \frac{16}{116}, & \text{sinon.} \end{cases}$$

Les plages des valeurs sont :

$$L \in [0, 100], \quad a, b \in [-127, 127].$$

Pour quantifier les valeurs en 8 bits :

$$L = \frac{L \cdot 255}{100}, \quad a = a + 128, \quad b = b + 128.$$

Quantification des niveaux de gris

1. Trier les intensités des pixels.
2. Diviser récursivement les intensités en sous-groupes selon la médiane.
3. Calculer les moyennes des sous-groupes pour obtenir les niveaux quantifiés.
4. Remplacer chaque pixel par le niveau le plus proche.

3.3 Architecture U-Net

Le réseau U-Net est composé de deux parties principales :

— **Encodage (Downsampling)** : réduction de la taille avec convolution (3×3), activation logistique et max pooling (2×2).

$$f(x) = \frac{1}{1 + e^{-x}}.$$

— **Décodage (Upsampling)** : reconstruction avec interpolation par voisin le plus proche (2×2).

3.4 Conversion inverse de Lab vers RGB

Étape 1 : Lab vers XYZ

$$f_x = \frac{a}{500} + f_y, \quad f_z = f_y - \frac{b}{200},$$

$$X = \begin{cases} f_x^3 \cdot X_n, & \text{si } f_x > 0.008856^{\frac{1}{3}}, \\ \frac{f_x - 16/116}{7.787} \cdot X_n, & \text{sinon.} \end{cases}$$

$$Y = \begin{cases} \left(\frac{L+16}{116}\right)^3, & \text{si } \frac{L+16}{116} > 0.008856^{\frac{1}{3}}, \\ \frac{L}{903.3}, & \text{sinon.} \end{cases}$$

$$Z = \begin{cases} f_z^3 \cdot Z_n, & \text{si } f_z > 0.008856^{\frac{1}{3}}, \\ \frac{f_z - 16/116}{7.787} \cdot Z_n, & \text{sinon.} \end{cases}$$

où $X_n = 0.95, Z_n = 1.089$.

Étape 2 : XYZ vers RGB

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 3.240479 & -1.53715 & -0.498535 \\ -0.969256 & 1.875991 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

où $R, G, B \in [0, 1]$.

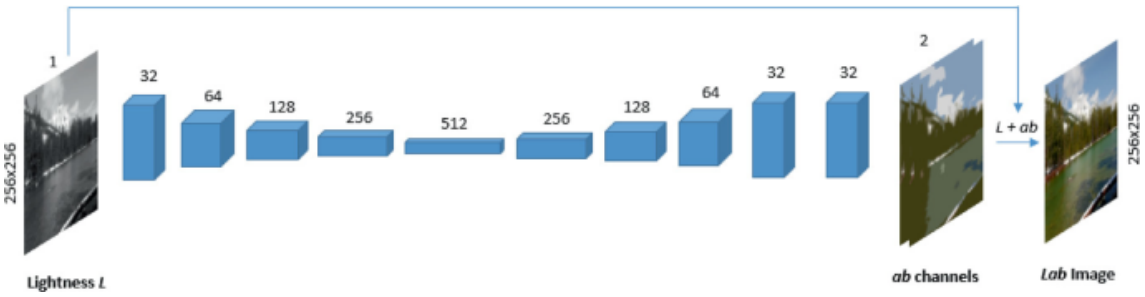

FIGURE 9 – Architecture U-Net pour la colorisation d'images en niveau de gris

4 Développement d'une interface

En parallèle, nous avons commencé à développer l'interface utilisateur de notre application. Nous avons décidé d'utiliser la librairie CustomTkinter en python. Et nous avons pu mettre en place une première version qui permet de charger une image et de générer le masque binaire de cette image.

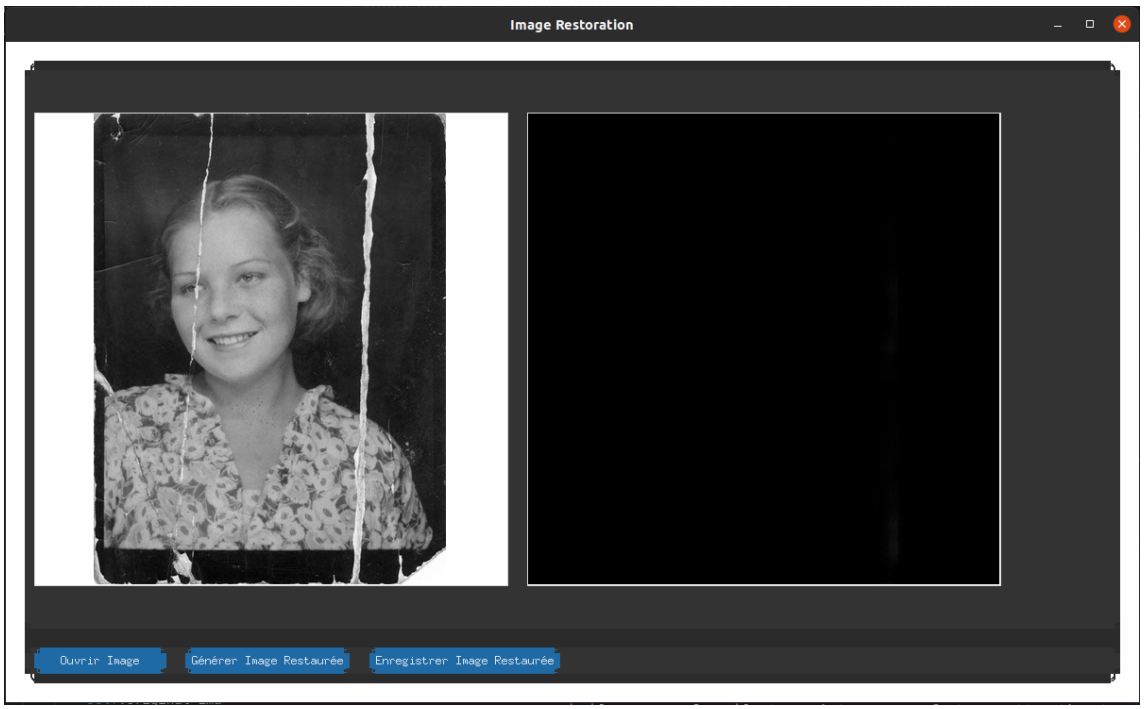

FIGURE 10 – L'interface de notre application

Ce n'est encore qu'une première version mais elle nous permet de visualiser rapidement les résultats de notre CNN. Nous allons rajouter la possibilité de choisir le modèle a utiliser afin de pouvoir les tester facilement.

Références

[1] O. Shkurat Z. HU M. Kasner. "Grayscale Image Colorization Method Based on U-Net Network". In : *Modern Education and Computer Science*. Avr. 2024. URL : https://mecs-press.net/ijigsp/ijigsp-v16-n2/IJIGSP-V16-N2-6.pdf.