Projet Image - Compte-rendu 6

Sécurité Visuelle - Obscuration d'image

Loïc Kerbaul - Valentin Noyé

25 novembre 2024

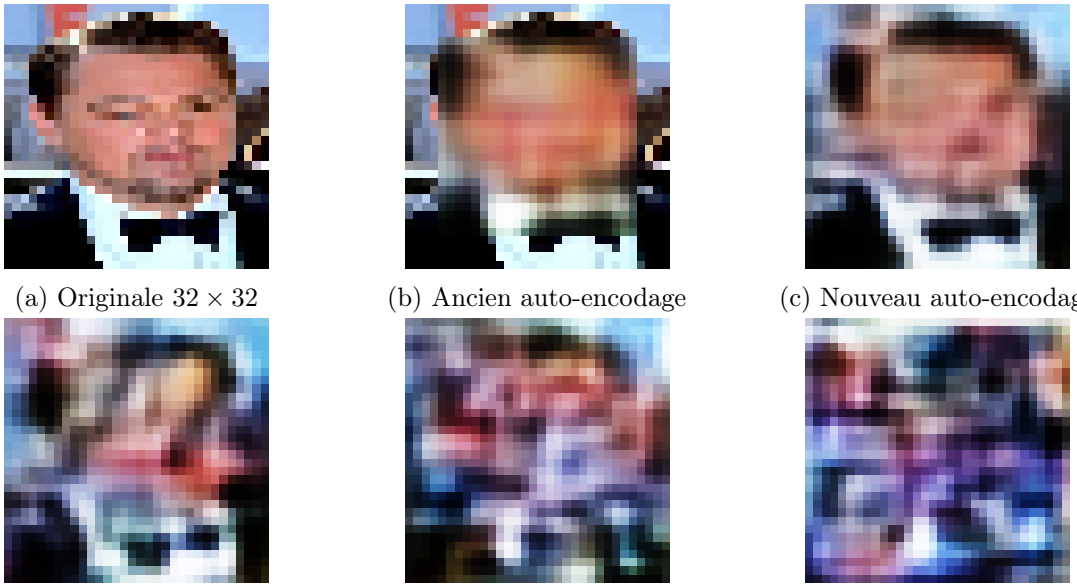
1 Introduction

Cette semaine, nous avons amélioré notre méthode d'obscuration par auto-encodeur à partir de l'utilisation de la base de données LFW, et nous avons commencé l'implémentation de l'interface.

2 Amélioration de l'auto-encodeur

2.1 Sur la base de données CIFAR-10

Nous voulions à l'origine améliorer les résultats obtenus sur la base de données CIFAR-10. L'une des observations que nous avions fait était que l'espace de caractéristiques est de seulement 4. Or, il s'agissait d'une erreur puisqu'il était en realité de taille $4 \times 4 \times 64$, soit 1 024 caractéristiques à la sortie de l'encodeur. Mais ces caractéristiques n'ont pas été aplaties et n'ont pas subi de réduction de leur dimensionalité, ce qui fait que les certains changements dans cet espace de caractéristiques étaient localisés. À présent, le modèle a été amélioré pour accueillir 2 048 de ces caractéristiques, en multipliant par 2 le nombre de convolutions sur la dernière couche. Les résultats sont légèrement meilleurs, et l'obscuration obtenue par la modification de l'espace latent sont plus intéressantes.



(a) Originale 32×32 (b) Ancien auto-encodage (c) Nouveau auto-encodage

(d) Obscuration à 25% (e) Obscuration à 50% (f) Obscuration à 100%

FIGURE 1 – Obscuration par modification de l'espace latent

Le tableau 1 montre l'évolution du PSNR et du SSIM en fonction de l'obscuration, qui s'avère être cohérent avec ce que nous percevons dans les images. Par exemple, l'image obtenue sans obscuration n'est pas de bonne qualité, et les manipulations qui en suivent le sont naturellement encore moins. Nous pouvons d'ailleurs attribuer en partie l'erreur obtenue à la base de données qui ne possède pas de visage humain.

Pourcentage obscurci	0%	25%	50%	75%	100%
PSNR (dB)	17,03	13,34	11,37	9,77	7,96
SSIM	0,738	0,540	0,422	0,218	0,045

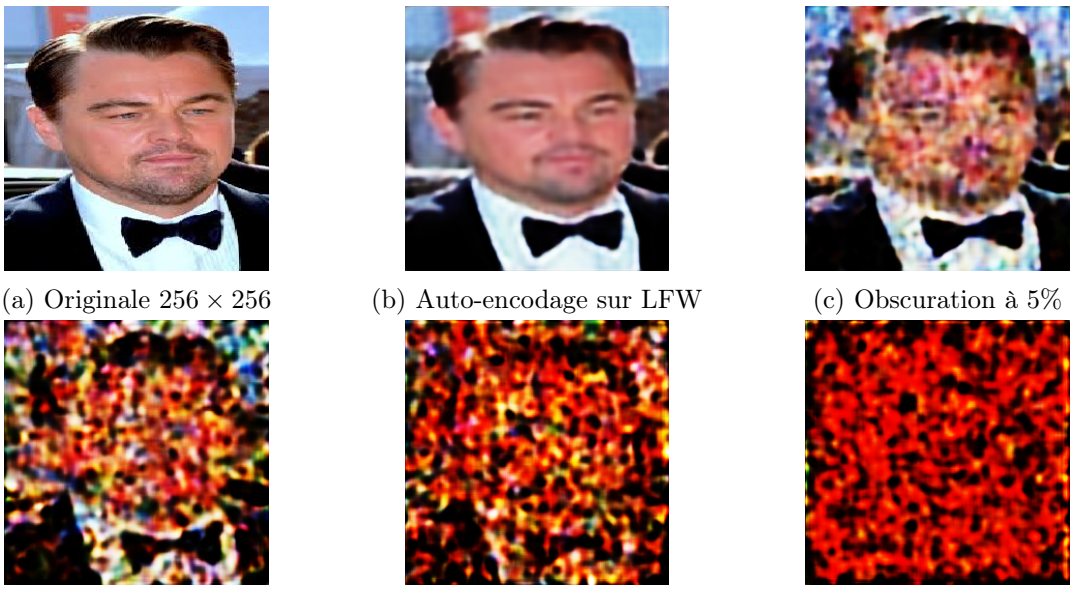
TABLE 1 – Évolution du PSNR et du SSIM en fonction de l'obscuration

2.2 Sur la base de données LFW

Nous avons utilisé à présent la base de données LFW (Labelled Faces in the Wild) pour entraîner notre modèle sur des visages et sur une taille d'images 250x250. Nous limitons la taille de la base de données à 500 personnes afin de pouvoir faire fonctionner notre modèle sur WSL et sur nos ordinateurs sans problème mémoire ou de temps de traitement trop longs. 90% des images récoltées sont utilisées pour l'entraînement, les 10% restants sont utilisés pour le test. Nous avons d'abord implémenté un modèle à trois couches tel que nous avions précédemment décrit dans le compte-rendu 4, mais avec 32, 64 puis 128 convolutions pour nos couches pendant l'encodage ; et inversement pour le décodage.

2.2.1 Obscuration par modification de l'espace latent

Comme pour CIFAR-10, nous obscurons certaines caractéristiques de notre espace latent. Comme constaté précédemment, la sortie de l'encodeur n'est pas un vecteur aplati mais un ensemble de $32 \times 32 \times 128$ caractéristiques, soit 131 072 d'entre elles. Nous les obscurons alors de la même manière que précédemment. Toutes les images sont redimensionnées en 256×256 pour faciliter nos tests.



(a) Originale 256×256 (b) Auto-encodage sur LFW (c) Obscuration à 5%

(d) Obscuration à 25% (e) Obscuration à 50% (f) Obscuration à 100%

FIGURE 2 – Obscuration par modification de l'espace latent

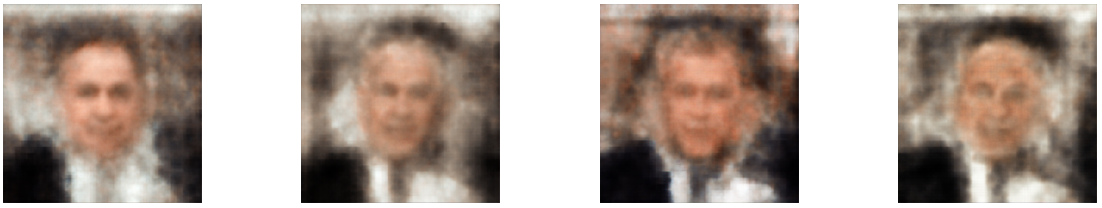
Le fait d'utiliser une base de visages de plus grandes dimensions améliore la qualité visuelle de l'image, ce que nous pouvons déduire quantitativement. Le PSNR est au-delà de 22 dB, ce qui représente une amélioration de plus de 5 dB par rapport à notre précédent modèle sur CIFAR-10. Avec l'ajout de bruit dans l'espace des caractéristiques, on constate que le PSNR et le SSIM se dégradent assez rapidement.

Pourcentage obscurci	0%	5%	25%	50%	75%	100%
PSNR (dB)	22,73	16,98	10,49	7,65	6,44	5,85
SSIM	0,753	0,455	0,218	0,126	0,085	0,065

TABLE 2 – Évolution du PSNR et du SSIM en fonction de l'obscuration

2.2.2 Obscuration par réduction de la dimensionalité

Notre première idée en voyant le nombre de caractéristiques obtenu est de procédé à une réduction de la dimensionalité par l'ajout d'une couche en fully-connected avec la sortie. Or nous avons là 131 072 caractéristiques par neurones de cette couche-ci, ce qui représente un nombre impressionnant de poids à garder en mémoire. Pour cela, nous faisons deux choses. Premièrement, nous réduisons la dimension de sortie de notre encodeur à $32 \times 32 \times 64$ caractéristiques, soit 65 536. Deuxièmement, nous limitons la couche en fully-connected en un maximum de 512 caractéristiques. Il est clair que nous obtiendrons de meilleurs résultats avec plus de caractéristiques, seulement, la phase d'entraînement plante durant la sauvegarde des poids lorsque nous en attribuons plus que ça. Une étude plus approfondie viserait à réduire de manière plus efficace la dimension de cet espace en utilisant un pooling global ou un pooling plus large.



(a) 64 caractéristiques (b) 128 caractéristiques (c) 256 caractéristiques (d) 512 caractéristiques

FIGURE 3 – Obscuration par réduction de la dimensionalité

Le tableau 3 nous fait comprendre qu'un plus grand nombre de caractéristiques influe sur le PSNR de manière assez proportionnelle. Par ailleurs, le SSIM possède des difficultés à faire des correspondances entre l'image de base et offusquée.

Nombre de caractéristiques	64	128	256	512
PSNR (dB)	11,16	11,81	11,96	12,45
SSIM	0,198	0,226	0,219	0,225

TABLE 3 – Évolution du PSNR et du SSIM en fonction du nombre de caractéristiques

3 Interface

Cette semaine, nous avons commencé à implémenter notre application, qui permet d'utiliser nos méthodes d'obscurations (uniquement les méthodes dites "classiques" pour l'instant) et d'évaluer les résultats obtenus. L'interface graphique est réalisée en Python, grâce à la librairie `Tkinter`. De plus, il est nécessaire d'avoir la bibliothèque `pillow` installée, celle-ci permettant la manipulation des images dans l'application. Pour lancer l'application, placez-vous simplement dans le répertoire *ProjetImage2_KERBAUL_NOYE/Code/application/*, et exécutez la commande suivante dans un terminal : `python3 main.py`. La fenêtre ci-dessous va alors s'ouvrir. Pour la fermer, vous pouvez utiliser la touche 'Echap', et la touche 'f' pour activer/désactiver le mode plein écran.

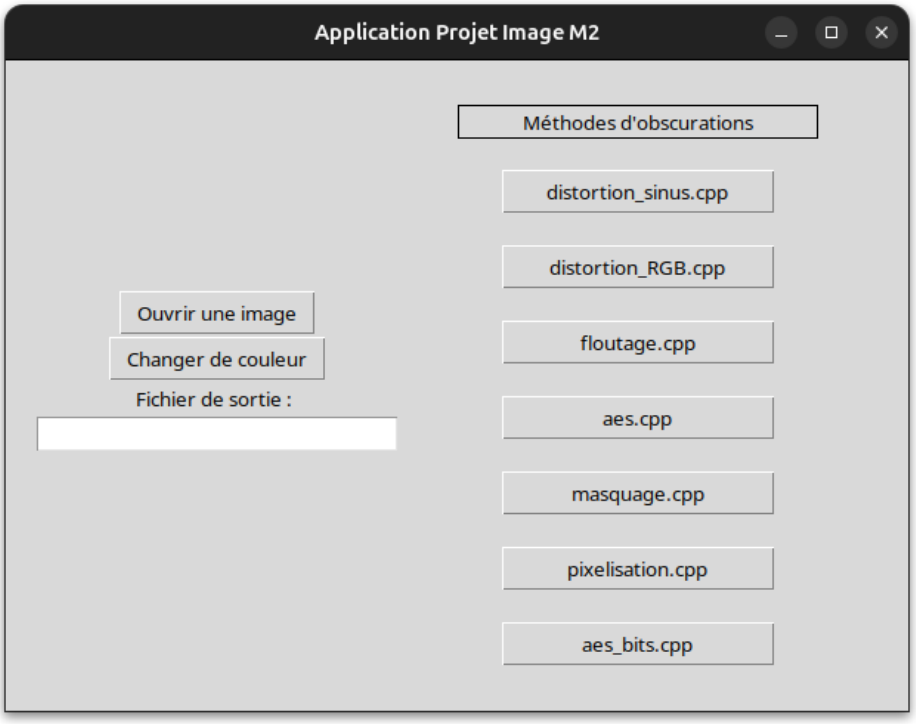


FIGURE 4 – Etat de l'interface au lancement

Afin d'obscurcir une image grâce à notre application, voici les étapes à suivre :

— Choisir une image source via l'explorateur de fichier, en appuyant sur le bouton 'Ouvrir une image'.
— Sélectionner la zone sur l'image où les pixels seront obscurcis, en maintenant le clic gauche de la souris. Si nécessaire, la couleur du cadre dessiné peut être modifié directement dans l'application, en appuyant sur le bouton 'Changer de couleur'.
— Entrer dans le champ de saisie le nom de l'image produite en sortie (cette image se trouvera dans le répertoire *ProjetImage2_KERBAUL_NOYE/Code/results/*).
— Choisir l'une de nos sept méthodes d'obscuration en cliquant sur l'un des boutons correspondants. Il faudra alors préciser les paramètres d'obscurations via les widgets interactifs de l'interface. Par exemple, si la méthode choisie est celle de distorsion sinusoïdale :

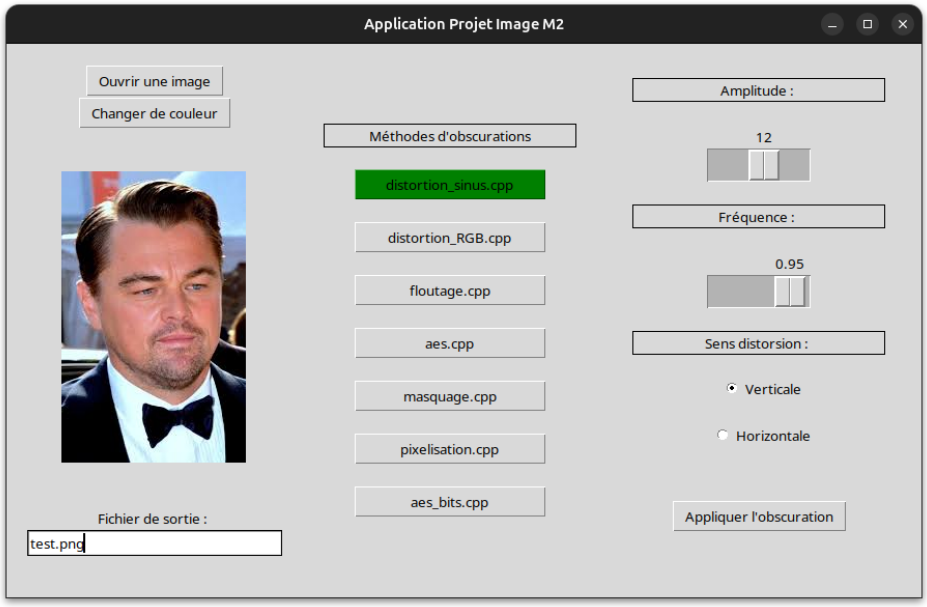


FIGURE 5 – Paramètres modifiables via l'application

— Enfin, pour valider les paramètres en produire l'image obscurcie, il suffit d'appuyer sur le bouton 'Appliquer l'obscuration'. L'image en sortie s'affichera alors sur l'interface, accompagnée des résultats de son évaluation par rapport à l'image initiale.

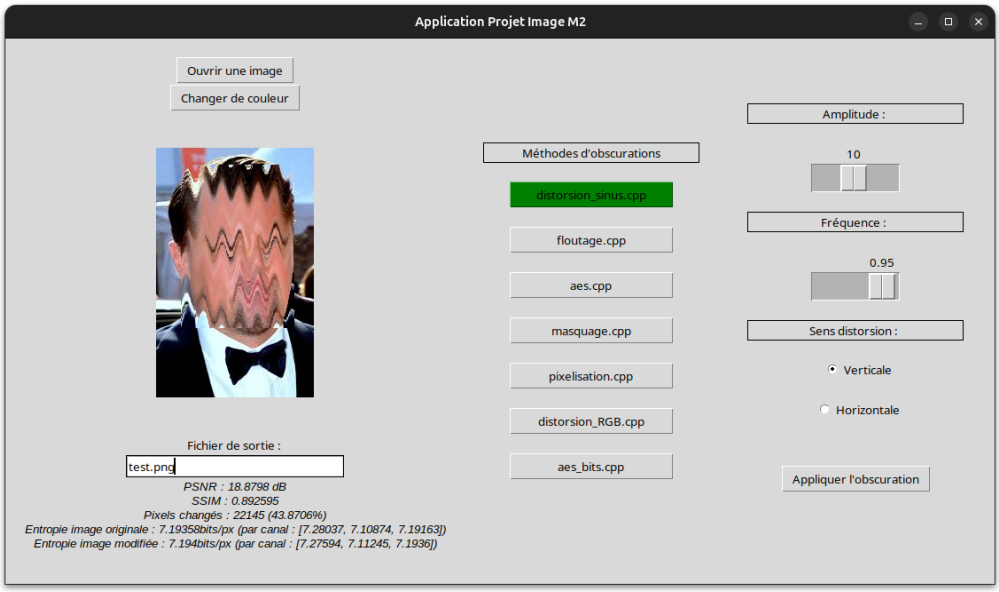


FIGURE 6 – Interface une fois l'image obscurcie produite