Projet Image - Compte-rendu 7

Sécurité Visuelle - Obscuration d'image

Loïc Kerbaul - Valentin Noyé

2 décembre 2024

1 Introduction

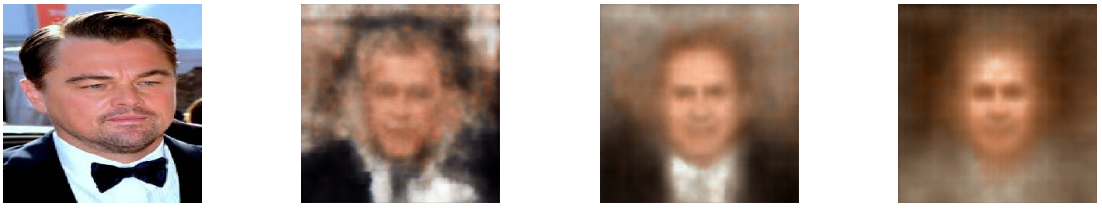
Dans ce compte-rendu, nous essayons d'améliorer nos résultats en utilisant un auto-encodeur variationnel et en utilisant le dataset Tiny Imagenet. Nous préparons également le terrain pour l'offuscation de séquences d'images pour la vidéo de présentation ainsi que pour la présentation de l'interface.

2 Auto-encodeur variationnel

Nous avons essayé d'implémenter un auto-encodeur variationnel (VAE) en se basant sur des ressources en ligne telles que la documentation de Keras qui offrait un tutoriel sur son implémentation. Car ce dernier donnait de très bons résultats avec MNIST, il était à supposer que nous puissions avoir nous aussi des résultats satisfaisants, en l'adaptant à notre dataset LFW d'images de taille 256×256.

En utilisant une architecture de réseau de neurones similaires à la précédente employée avec notre auto-encodeur simple, nous obtenons des résultats présentés dans la figure 1 lorsque nous effectuons une réduction de la dimensionalité des données.

Car nous avons peu de caractéristiques (256), nous nous attendions à obtenir des résultats distordus, mais en l'occurrence nous ne retrouvons en aucun cas notre acteur Leonardo DiCaprio ni le mandrill (quoique le modèle étant entraîné sur des visages), mais un résultat moins bruité que précédemment. Les résultats nous laissent donc perplèxes.



(a) DiCaprio original (b) DiCaprio - AE (c) DiCaprio - VAE (d) Mandrill - VAE

FIGURE 1 – Obscuration par réduction de la dimensionalité à 256 caractéristiques

3 Classifications avec Tiny ImageNet

3.1 Mise en place

Nous employons ici une fraction du dataset Tiny ImageNet afin d'évaluer nos méthodes d'obscuration, contenant 200 classes de 500 images chacune, soit un total de 100 000 images

de taille 64×64. L'idée derrière une telle utilisation est de pouvoir travailler à présent sur des images deux fois plus grandes. Le modèle de réseaux de neurones a donc été adapté pour faire 64, 128, 256 puis 512 convolutions suivies d'une normalisation du batch puis d'un pooling 2×2. À la fin, on aplatit les caractéristiques puis on fait un fully-connected vers 1024 caractéristiques avec un dropout de 0,5 pour éviter l'overfitting. On retrouve en dernière couche un fully-connected vers N classes, selon notre choix. Ceci est le modèle avec la plus grande précision que nous avons pu construire pour ce dataset.

Nous sélectionnons à présent un nombre limité de classes, tel que 10. Nous remarquons immédiatement que la précision de 45,53% du modèle est inférieure à celle de CIFAR-10 comprise entre 70% et 80%, ce que nous attribuons en majorité à quelques facteurs :

— Les classes contiennent un très petit nombre d'images d'entraînement de 400 à 450 images, face aux 6000 de CIFAR-10. En effet, nous gardons de côté 50 à 100 images pour les tests. Nous nous sommes également adonné à augmenter artificiellement la taille du dataset à 600, 750 et 1000 images par classe, mais cela semblait seulement introduire un sur-apprentissage.

— Les images de Tiny ImageNet contiennent beaucoup d'artéfacts JPEG ou sont de moindre qualité.

— Beaucoup plus de détails sont introduits dans des images 64×64, et également beaucoup de variation. Par exemple, la classe «boulier »contenait parfois des personnes ou autres objets à côté, ce qui pouvait poser problème lors de la classification.

3.2 Résultats

Nous avons effectué les différentes classifications sur l'ensemble des méthodes d'offuscation. Nous obtenons une précision de 24,80% sur des données obscurcices sans contexte et sans entraînement, et une surprenante précision de 22,10% sur les données avec contexte, à savoir que nous obscurons ici une zone minimale de 32×32. Après entraînement du modèle avec données offusquées, on obtient 25,12% sans contexte et 47,95% avec contexte, ce qui est beaucoup plus rassurant dans le dernier cas, mais étonnant car cette précision dépasse la précision du modèle sur le dataset inchangé.

Ensuite, en entraînant notre modèle à reconnaître si une image a été obscurée ou non, on obtient cette fois-ci une très bonne précision de 88,70% qui démontre que l'efficacité du modèle à reconnaître l'obscuration n'est influé que très légèrement par le dataset. On obtient également une précision de 81,55% par introduction de contexte.

Finalement, pour la reconnaissance du type d'obscuration, on obtient une précision de 54,76% sans contexte à l'obscuration et 51,65% avec.

Il est clair que les résultats que nous obtenons sur ce dataset sont beaucoup moins précis que sur CIFAR-10, ce qui peut suggérer que cela provient du dataset, voire même du nouveau modèle, ce qui n'est pas certain.

4 Obscuration de vidéo

Cette semaine, nous avons adapté nos techniques d'obscuration afin de pouvoir les utiliser sur des vidéos. Nous utilisons la librairie OpenCV avec Python pour récupérer les frames d'une vidéo source, afin de les obscurcir puis de reconstruire la vidéo cible avec ces nouvelles frames. Pour chaque frame, l'obscuration est appliquer sur les zones où l'on détecte des visages. Le tracking des visages est rendu possible grâce à OpenCV (qui utilise la méthode des cascades de Haar).

5 Ajout dans l'interface

Nous avons ajouté dans l'application la possibilité de pouvoir traiter des vidéos, avec les mêmes méthodes d'obscurations que celles destinées aux images. Pour cela, il suffit de sélectionner une vidéo (au format `mp4`) au lieu d'une image. Si la vidéo est bien ouverte, la première frame s'affichera dans l'interface (voir sur la Figure 2). Comme pour les images, il faut ensuite choisir une méthode d'obscuration, définir les paramètres, donner un nom pour la vidéo en sortie, et enfin lancer le processus.

La vidéo produite sera enregistrée dans le répertoire *Code/results*, avec le nom qui aura été rentré dans le champ de saisie. Pour tester l'obscuration de vidéo, nous avons fourni un fichier vidéo (au format `mp4`) en exemple, que vous trouverez dans le répertoire *Code/videos*. Cette vidéo ayant une durée de plus d'une minute, l'obscuration de chacune des frames peut prendre un certain temps. Il est cependant possible d'arrêter le processus à tout moment, en faisant `Ctrl+C` dans le terminal ayant servi à lancer notre application. La vidéo en sortie ne contiendra alors que les frames qui auront pu être obscurcies.

Les librairies Python suivantes sont nécessaires à la bonne exécution du code : **opencv-python** et **opencv-contrib-python**.

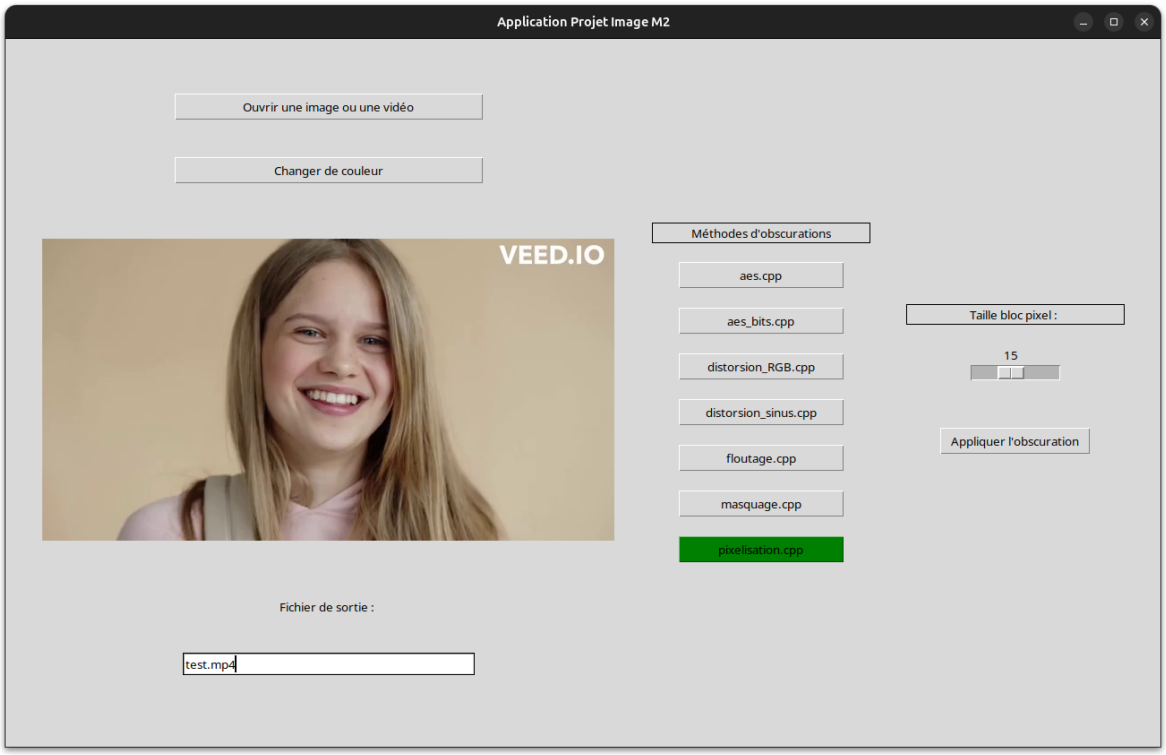


FIGURE 2 – Obscuration d'une vidéo grâce à notre interface