Projet Image Compression

Détection de falsifications dans des images

Compte-rendu 2

Louis JEAN
Ayoub GOUSSEM
Master 1 IMAGINE
Université de Montpellier

10 mars 2024

Table des matières

1 Introduction

Ce compte-rendu détaille nos avancées dans le projet durant la semaine du 4 mars. Au programme, choix d'implémentation, mise en place d'une première méthode de détection et création de l'interface utilisateur. **N.B :** veuillez lire le README pour faire fonctionner le projet.

2 Choix d'implémentation

L'infrastructure de notre projet est fondée sur des choix d'implémentation stratégiques qui privilégient la performance, la maintenabilité et la portabilité du code. Nous avons opté pour l'utilisation de **Virtualenv**, **Pillow**, **Numpy** et **customtkinter**, chacun répondant à des besoins spécifiques de notre application.

2.1 Virtualenv

L'isolation des dépendances est cruciale pour assurer la reproductibilité de notre application sur différentes machines et environnements, notamment sur les ordinateurs de la faculté pour le jour de la soutenance. **Virtualenv** crée des environnements Python isolés, ce qui permet de gérer les dépendances de manière indépendante pour chaque projet. Cela évite les conflits entre les packages et garantit que tous les développeurs travaillent avec les mêmes versions des bibliothèques. Pas de "Ça marche sur ma machine !".

2.2 Pillow

Pour le traitement des images, nous avons choisi **Pillow**, un fork de la bibliothèque Python Imaging Library (PIL). **Pillow** offre des fonctionnalités étendues pour l'ouverture, la manipulation et la sauvegarde de nombreux formats d'images. Cette bibliothèque nous permet de nous concentrer sur des algorithmes de détection de falsification tout en maintenant un code clair et lisible.

2.3 Numpy

En utilisant **Numpy**, nous pouvons effectuer des calculs de manière rapide, ce qui est impératif compte tenu de la grande quantité de données contenues dans les images. De plus, il est possible de transformer une image **Pillow** en tableau **Numpy**, ce qui s'avère très pratique pour manipuler les pixels.

2.4 CustomTkinter

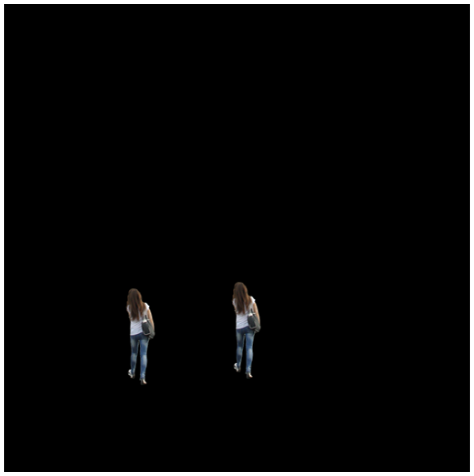
Pour l'interface utilisateur, **CustomTkinter** a été préféré à **Tkinter** traditionnel en raison de son design moderne et de ses composants personnalisables qui améliorent l'expérience utilisateur. Il est léger et conserve la facilité d'utilisation de **Tkinter**, ce qui le rend adapté pour un déploiement rapide et efficace de l'interface utilisateur de l'application.

3 Première méthode de détection

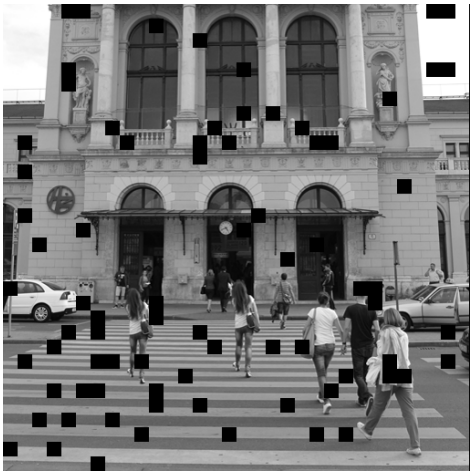
Nous avons mis en œuvre une méthode de détection basée sur l'analyse des caractéristiques des blocs de pixels au sein des images. L'approche adoptée consiste à diviser l'image en blocs et à calculer pour chacun la moyenne et l'écart-type. Les blocs sont de taille paramétrable. Les blocs sont ensuite comparés deux à deux afin d'identifier les paires présentant des similitudes suspectes, potentiellement indicatives d'une manipulation de type copy-move. Les critères de comparaison incluent des seuils adaptatifs qui tiennent compte de la variabilité globale de l'image, réduisant ainsi le risque de faux positifs dans les zones homogènes. Les blocs suspects sont marqués en noir et une image résultante est générée, mettant en évidence les régions potentiellement falsifiées.

Cette méthode, très naïve et peu élaborée, constitue seulement le premier pas de notre programme. Elle manque d'ailleurs beaucoup de rafinement, car elle ne parvient pas vraiment à détecter les falsifications. Nous comptons l'affiner en utilisant de nouvelles caractéristiques bien plus précises, comme des descripteurs de texture ou les coefficients de la transformée de Fourier. Nous envisageons aussi d'implémenter du clustering pour regrouper les blocs selon leurs caractéristiques, en utilisant un k-mean.

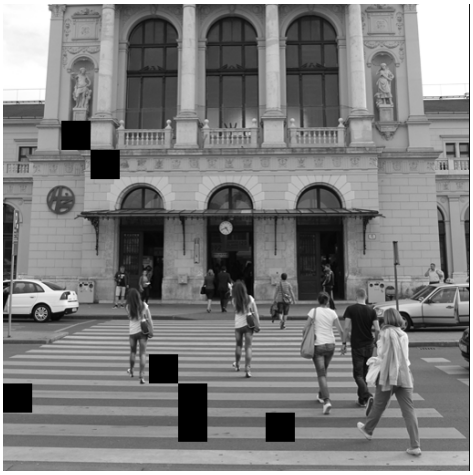
(a) Image falsifiée



(b) Falsification mise en évidence



(c) Image générée par notre programme pour des blocs de taille B = 16

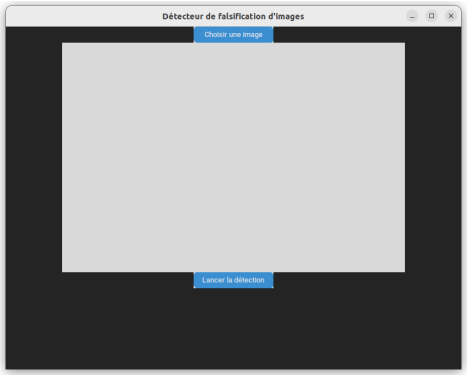


(d) Image générée par notre programme pour des blocs de taille B = 32

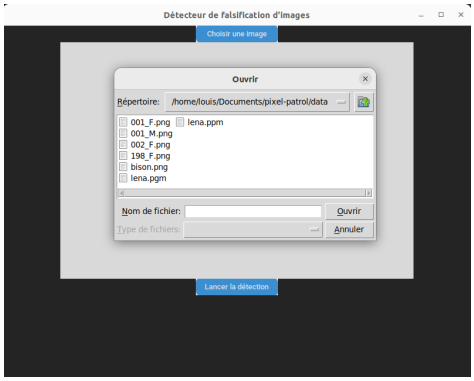
Figure 1: Illustration de notre application sur une image issue du dataset CoMoFoD

4 Interface utilisateur

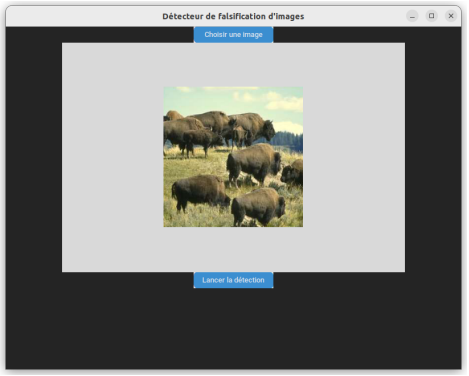
Parallèlement à la méthode de détection, une interface utilisateur a été conçue en utilisant la bibliothèque **CustomTkinter**. L'interface permet à l'utilisateur de sélectionner une image via un dialogue de fichiers, d'afficher l'image chargée, et de lancer le processus de détection. Une fois la détection effectuée, l'image résultante est affichée dans l'interface, permettant à l'utilisateur d'examiner visuellement les résultats de l'analyse.



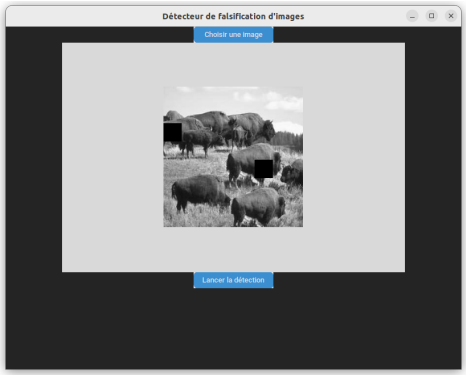
(a) Ouveture de l'application



(b) Choix de l'image



(c) Affichage de l'image choisie



(d) Affichage de l'image obtenue après détection

Figure 2: Illustration de notre interface graphique

Bien sûr l'interface graphique n'est pas du tout complète, et nous comptons rajouter de nombreuses fonctionnalités, permettant une analyse plus poussée des résultats (meilleur affichage des falsifications détectées, pourcentage de sûreté de falsification, possibilité de charger plusieurs images en même temps, ...).

5 Conclusion

Les avancées de cette semaine constituent des étapes cruciales dans notre projet de développement d'un outil de détection de falsification d'images. La première méthode de détection mise en place jette les bases du code, mais doit être grandement améliorée. L'interface utilisateur marque un bon début, nous allons pouvoir itérer dessus. Les prochaines étapes incluront des tests plus approfondis, l'affinement des méthodes de détection et l'enrichissement des fonctionnalités de l'interface utilisateur.