Traitement d'images en Python

Détection de la censure en mosaïque

BATANY William

BENNACER Ibrahim

CHALANDON Pierre

CHIM Benoit

SOMMAIRE

I. INTRODUCTION……P3

II. CENSURE D'UNE IMAGE……P4

a) *L'image Zoulette.png*
b) *L'image d'un paysage.png*
c) *L'image des Fruits & Légumes.png*

III. MISE EN PLACE DE MASKS……P10

IV. LA RECHERCHE……P11

V. AMELIORATIONS POSSIBLES……P14

VI. CONCLUSION……P14

I. INTRODUCTION

Notre atelier portant sur le traitement d'images sous le langage de programmation Python, était scindé en deux parties :

- *Traitement d'images basiques*

L'objectif de cette première partie était d'acquérir les bases nécessaires en Python pour la suite de notre projet, ainsi que d'apprendre les bases de l'analyse et manipulation d'images, comme par exemple :

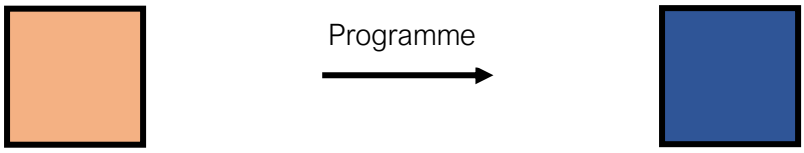
- Représentation d'une image.
- Calcul d'un histogramme.
- Transformations simples (flip, rotations…).

Ces notions de programmation, acquises sur Schooding, nous permettrons alors, par groupe de 4, de concevoir un projet basé sur l'étude de la mosaïque en Python.

- *Traitement d'images avancé* (~5 mois)

Ce projet, se déroulant sur le restant du semestre, a pour objectif de concevoir un programme capable de détecter la présence de censure mosaïque sur une image. Pour cela, nous sommes partis d'un programme fourni par notre professeur, Mr Raynal, permettant de générée, de façon aléatoire, une censure mosaïque sur une image donnée.

La censure, étudiée ici, est une censure dite en « *Mosaïque* ». Cela signifie qu'elle est représentée par un carré de pixel d'une couleur unie.



Censure dans l'image de base *Représentation de la censure détectée*

On cherche donc à détecter les carrés de censure présents dans l'image. Puis de les retourner d'une nouvelle couleur, sur une nouvelle image.

II. CENSURE D'UNE IMAGE

a) L'image Zoulette.png

Tout d'abord, nous avons décidé de travailler sur l'image suivante, qui nous semblait intéressante, du fait de la multitude de couleurs sur les graffitis en arrière-plan.

Une fois la censure appliquée, il nous fallait retourner la censure présente.

Durant les semaines suivantes, l'élaboration de notre code s'est peaufinée, pour obtenir une première itération de notre projet.

Au départ, nous avions imaginé un code essayant de répondre à notre problématique : Son processus était de convertir l'image en nuances de gris, puis de transformer les pixels censurés en pixels rouges. Cependant, nous ne l'avons jamais utilisé car ce dernier rendait la compilation infinie.

Nous nous sommes donc orientés, avec l'aide de notre professeur, vers une autre méthode.

Cette deuxième méthode nous permet de reconnaître les pixels de « presque même couleur » qui sont côte à côte. Si le programme détecte un nombre important de pixels similaire proche les uns des autres, alors il les retourne en bleu. Nous avons ici un premier prototype fonctionnelle répondant à notre problématique de détection de censure.

Résultat obtenu :

On constate que toutes les zones censurées ont bien été détecter cependant notre programme a pris en compte d'autres pixels, non censurés, surement dû au fait que l'image soit trop uniforme pour certaines zones.

Pensant que cette image était idéale pour avancer dans notre projet, il s'avère qu'elle était plutôt un frein à notre avancée. Malgré cela, elle nous a permis de bien comprendre les différents enjeux et difficultés auxquelles nous allons être confrontées par la suite. En effet, selon l'image choisie, nous aurions des problèmes de censure différents.

Par conséquent, nous avons décidé de changer de photo, en prenant une image avec des couleurs hétérogènes sur différents plans.

b) *L'image d'un paysage.png*

Pour perfectionner notre programme, nous avons décidé de lui ajouter une deuxième méthode pour lui permettre d'être plus précis dans la détection des carrées censuré.

L'objectif était ici de faire en sorte qu'il repère les pixels côte à côte, et qui sont presque de même couleur (comme précédemment), mais en plus, ajouté une autre fonctionnalité : une détection à l'intérieur même de la détection précédente. En effet, si dans une zone de pixels détecté plus de 91% des pixels sont bleus (il faut un nombre important de pixels bleus similaires et côte à côte), alors toute la zone sera remplie en bleu. Dans le cas contraire la zone ne sera pas considérée comme censurée.

Résultat obtenu :

On obtient un résultat beaucoup plus précis, avec une diminution de la marge d'erreur.

Cependant, on voit que les zones ne sont pas encore parfaitement délimitées et qu'une certaine partie autour de la zone de censure n'est pas encore détectée par notre programme.

Néanmoins, ce programme présente plusieurs problèmes, notamment son temps d'exécution de plusieurs minutes. De plus, au vu des nombreux plantages de l'ordinateur durant les diverses compilations, il semblait essentiel de se penchés sur un moyen de ne plus avoir ce problème.

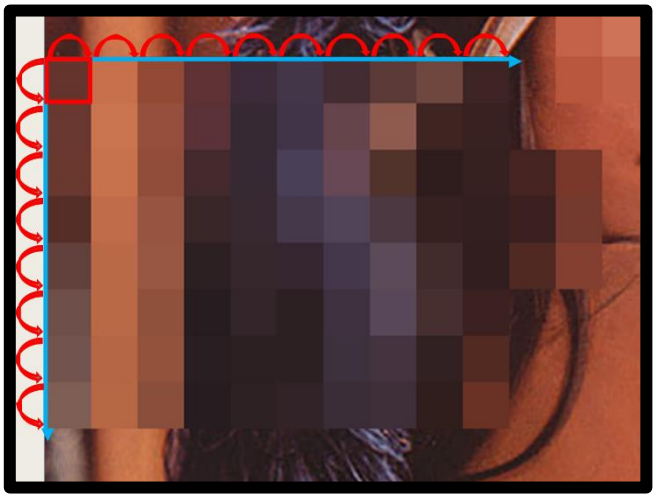
c) *L'image des Fruits & Légumes.png*

Dans un premier temps, il nous paraissait logique de tester notre programme avec une nouvelle image afin de savoir si on obtient une nouvelle fois un résultat tout aussi convainquant. Pour cela, nous avons choisi l'image suivante :

Cette image a pour particularité de ressembler à l'image précédente, de pars ses couleurs hétérogènes, mais de taille bien plus inferieur. On pouvait supposer que plus l'image est petite, moins le programme mettrait de temps à parcourir l'image.

Ce qui était, en partie, le cas. Le temps de compilation était légèrement moins long, mais pas suffisamment pour que nous puissions travailler de manière fluide. Il a donc fallu modifier le programme.

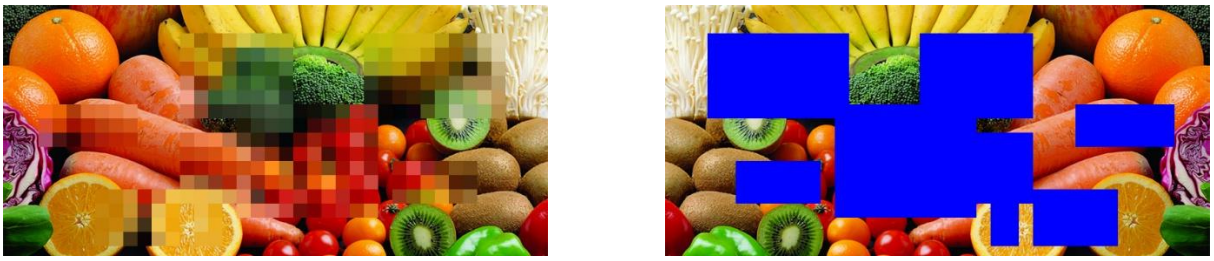
Afin de gagner en temps de calcul, notre solution était de demander à notre programme de parcourir les pixels de l'image avec une distance euclidienne. Pour obtenir une détection optimale pour un temps de calcul faible, il faut parcourir l'image censuré non pas pixel par pixel mais side par side (avec side représentant la largeur d'un carré de censure). En faisant cela, le programme ne parcourt plus l'intégralité de l'image et détecte le coin supérieur gauche (point de départ) de chaque potentiel carré de censure.



Le programme ne va plus parcourir l'image pixels par pixels (comme indiqué par les flèches bleues), mais side par side (comme indiqué par les flèches rouges).

Il va parcourir toute l'image plus rapidement.

Cette modification a porté ses fruits, car dorénavant, la détection de censure est quasiment instantanée. On a donc eu un **gain de temps considérable** par rapport à notre précédente version.



Détection après censure

De plus on remarque, que les carrés sont **parfaitement bien délimités**, et que la censure correspond bien uniquement à la zone censurée. Nous avons donc réussi à régler les problèmes antérieurs.

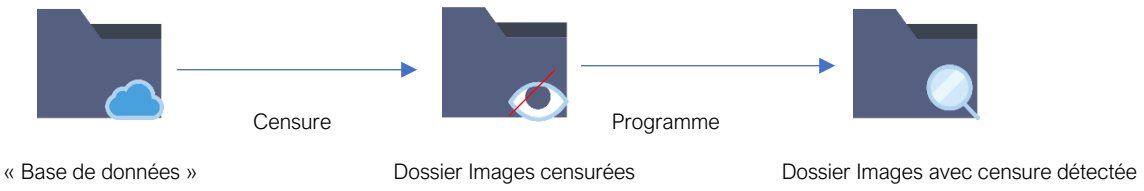
Mais après ?

Une fois que notre programme avait fait ses prévues, il fallait tester ce dernier sur plusieurs images totalement différentes. Pour cela nous avons créé un espace de travail plus lisible pour nous avec la présence de **répertoire**, sous forme de « base de données ». Le programme va désormais chercher les censures sur l'ensembles des images du répertoire « images censurée » puis les envoyées, après détection, dans un répertoire « images détectée ».

A ce moment-là, nous avons désormais mis en place 3 dossiers :

- Dossier des images à censurer.
- Dossier des images censurées par le programme « makemosaïque »
- Dossier des images avec la censure détectée.

De ce fait, avec ces 3 dossiers, il est bien plus simple de se retrouver dans nos différentes images étudiées. L'objectif de pouvoir compiler plusieurs images à la fois est donc atteint.

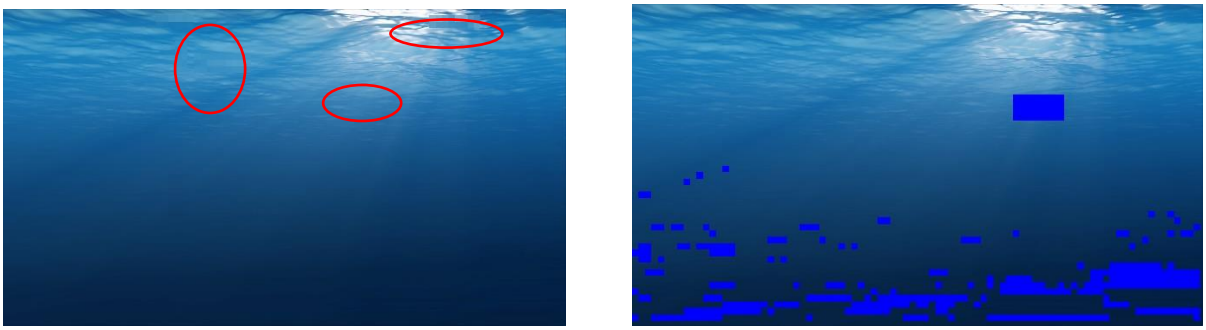


Une fois les répertoires créés, nous avons mis plusieurs images de différentes tailles/couleurs/résolutions.

Après plusieurs tests, on distingue alors 2 cas :

- Pour les petites images, le programme fonctionne.
- Pour les grandes images de couleur uniforme, le programme est moins efficace, il détecte de la censure supplémentaire.

Exemple avec une image d'un océan, où les couleurs sont quasiment pareilles.

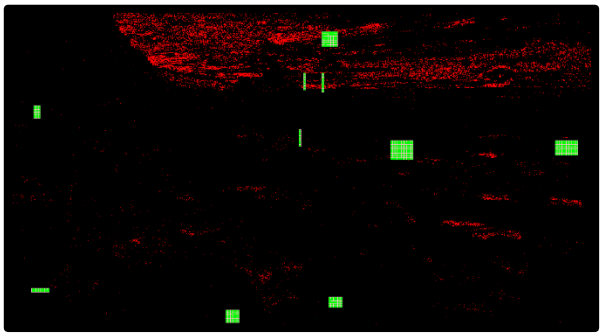


Différence au niveau du bas de l'image, où de la censure en trop est détectée.

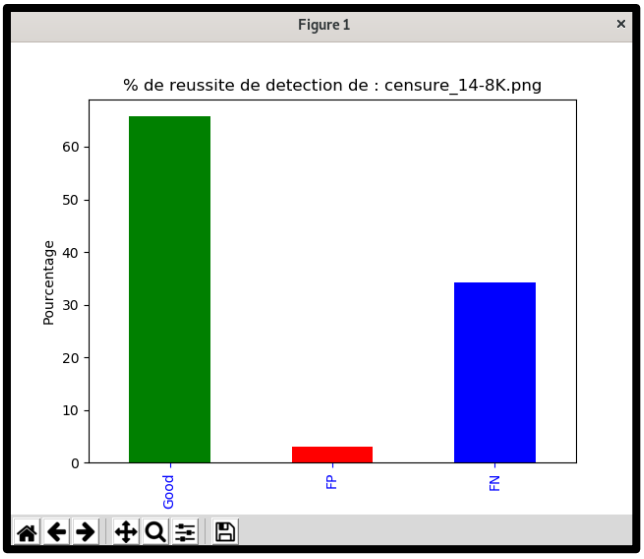
III. MISE EN PLACE DE MASKS

Pour résoudre nos différents problèmes, nous avons décidé de mettre en place des « masks ». Ce dernier permet d'avoir une comparaison en temps réel et exact entre nos résultats obtenu et ce que nous devions obtenir.

Pour cela, durant la censure des images, la programme va désormais générer un dossier contenant les coordonnées exactes des carrées de censure pour les comparer avec les carrée de censure que notre programme a lui détecté. A l'aide de ces coordonnées, le programme va créer une image mettant en évidence les réussites et les échecs de notre programme.



Pour éviter d'avoir a effectué une comparaison approximative a l'œil nu, nous avons décidé de présenter les résultats sous forme d'histogramme :



Histogramme représentant le pourcentage des bons pixels/faux positifs/faux négatifs.

Ce support est un excellent moyen de réfléchir aux différentes améliorations à apporter à notre programme.

A ce moment-là de notre projet, il nous était désormais plus facile de s'interroger sur les différentes causes des problèmes que nous rencontrons notamment sur les images à haute résolution. La présence d'histogramme permet de comprendre rapidement ce qu'il faut remodifier dans notre programme.

IV. LA RECHERCHE

Nous avons commencé la Recherche pour perfectionner notre programme. Par binômes, nous avons travaillé chacun sur une problématique différente :

1. Quel rôle joue la résolution de l'image ?
2. La taille du side (radius) joue-t-il un rôle majeur sur l'efficacité de la détection de censure ?

➢ Première problématique : La résolution

Pour cela, nous avons essayé notre programme sur une même image, mais avec une résolution différente à chaque fois.

Largeur	Hauteur	Standard
256	144	144p
426	240	240p
640	360	360p
848	480	480p
1280	720	HD
1366	768	WXGA
1600	900	HD+
1920	1080	FULL HD
2560	1440	QHD
3200	1800	QHD+
3840	2160	UHD1
4096	2304	4K
5120	2880	5K
7680	4320	8K

Après tests, notre programme fonctionne parfaitement. On ne constate pas de faux positifs ou faux négatifs. Néanmoins, dès l'image 4K, nous rencontrons un problème : en effet, des erreurs mineures sont présentes. Cela peut s'expliquer par le nombre très important de pixels sur l'image. Nous avons réussi à résoudre ce problème. Cependant, sur l'image 8K, le programme est plus long à exécuter et il reste quelques faux positifs.

Image 8K

Cette marge d'erreur reste assez négligeable, car la totalité de la censure est retrouvée, seuls quelques carrés sont en trop dans le résultat final. De plus, la grande résolution rend le pourcentage de faux positifs vraiment négligeable (de l'ordre inférieur à 1%)

Par conséquent, on sait que la résolution de l'image est un facteur assez important pour retrouver une censure parfaite, mais ce n'est pas la seule explication. Pour cela, rechercher des informations sur la deuxième problématique est primordial pour la réussite de notre projet.

- Deuxième problématique : La taille du side

Qu'est-ce qu'un side ?

C'est la représentation de la longueur du carré de censure affiché dans la nouvelle image.

La recherche de sa formule était une grosse interrogation. Essayer de calculer sa taille et son radius et le reformuler sous forme de langage de programmation et non mathématique a été un véritable défi.

Pour calculer la taille du side, on demande à notre programme de parcourir chaque pixel de toute l'image. A chaque pixel parcouru, il vérifie combien sont similaires côte à côte. Nous affichons ensuite l'histogramme correspondant à cela et nous observons un phénomène intéressent : il y a une chute dans l'histogramme au niveau du side de notre image. De ce fait, on arrive à détecter où il se situe exactement, puis on dérive une première fois, ainsi qu'une deuxième fois (le nombre de pixels similaires côte à côte), pour obtenir les graphes permettant de récupérer la taille du side. En effet, les dérivés première et seconde traduisent la chute dans l'histogramme par des pics. Ceux-ci sont alors facilement détectable.

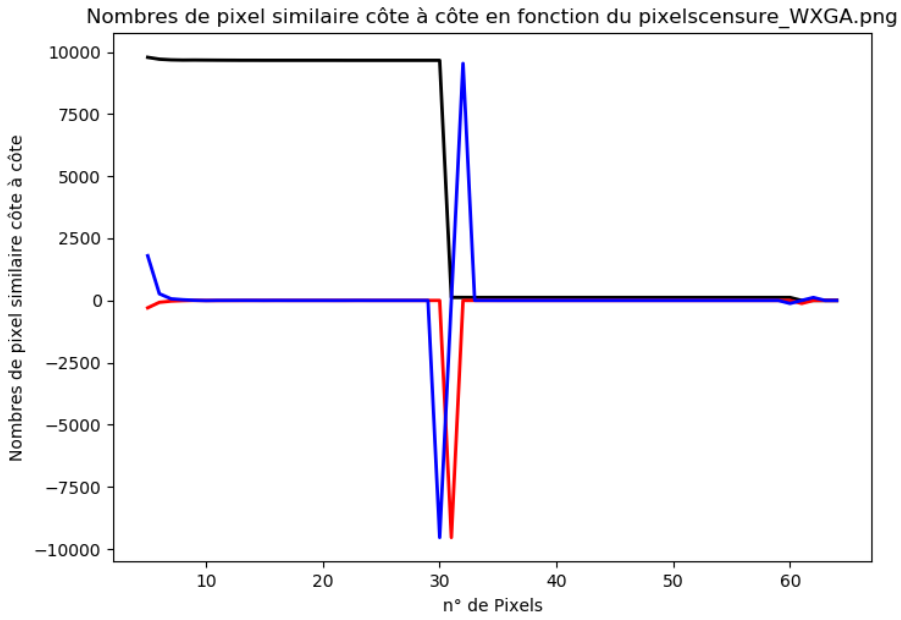
Une fois réussi, on réutilise cette formule, en les testant sur les 3 premières images de notre étude, avec une censure possédant 3 valeurs de side différentes. (Taille 15, 20 et 30)

Les tests étant concluants, nous avons continué notre observation avec un essai pour une série de même image mais de résolution différente.

On remarque qu'avec des sides identiques, entre 0 et 8K la détection n'a pas de défauts, mais qu'au-delà de ce palier, des petites erreurs apparaissent.

En effet, on ne retrouve pas la même taille de side partout (à certains tests, on se retrouvait avec des sides de taille 10-12, ce qui ne correspondait absolument pas)

La « zone de fracture » qui représente la bonne taille du side à récupérer était parfois difficile à détecter car selon les images, il y avait plusieurs zones de fractures. Heureusement, la zone de fracture représentant le side est la plus importante ce qui nous permet de la détecter à tous les coups avec un système de max. Ce dernier a été une très grosse période de recherche durant notre projet, notamment avec Schooding et Internet, dans le but d'en avoir un le plus précis et le plus efficace possible.



V. AMELIORATIONS POSSIBLES

Afin d'améliorer notre projet, nous aurions aimé travailler sur des images uniformes ou quasi-uniformes et des images à haute résolution : ces images présentes un grand nombre de pixels côte à côte trop similaires ce qui rend la détection de censure difficile avec notre programme actuel.

Enfin, nous aurions aimé tester d'autres méthodes de détection de censure, notamment détecter une succession de carrés de censure côte à côté, en plus de nos autres méthodes afin d'affiner nos détections.

VI. CONCLUSION

A la fin de notre projet, nos objectifs ont été atteints et nous sommes satisfaits de nos résultats obtenus, mais conscients qu'ils peuvent être améliorés. Cependant, pour un projet sur un langage de programmation qui nous était inconnu il y a 5 mois, nous avons su avancer en groupe et travailler efficacement ensemble, en se répartissant bien les taches, en fonction des forces et des faiblesses de chacun. Cette excellente ambiance de travail de groupe à été bénéfique, pour notre premier projet basé sur le long terme.

Nous sommes aussi contents d'avoir choisi cet atelier, nous permettant de découvrir une nouvelle facette du monde de l'informatique, que nous ne pouvions pas imaginer. Cela nous a permis d'allier l'utile à l'agréable, car même si travailler sur la détection de censure nous a demandé rigueur et sérieux, cela a été une agréable expérience.

Le fait d'être confrontés une multitude de problèmes différents les uns des autres, nous a apporté une véritable réflexion, et des situations à savoir contrôler, qui nous serons surement utile pour notre métier d'ingénieur.

Nous aurions aimé avoir plus de temps, pour nous concentrer sur un nouveau projet pouvant être associé au notre, qui est la reconstitution de censure.