Projet image : Détection de zones falsifiées par copier-déplacer dans des images, compte rendu 7

Marie Bocquelet, Arthur Villarroya-Palau, Daniel Blanchard

Master 1 IMAGINE Semestre 2, 9 avril 2023

Table des matières

1 Introduction :

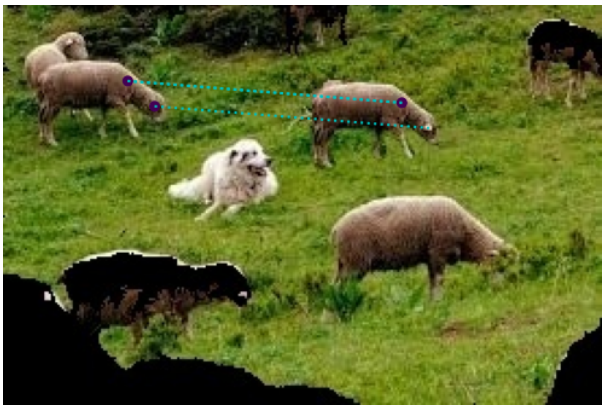
Ne restant plus que trois semaines de projet, les rapports de fin de semaine seront plus conséquents que précédemment. Durant les dernières semaines, nous avons pu considérablement avancé. En effet, tout d'abord l'application nécessaire à la démonstration finale est dans sa phase de finalisation, ce qui est parfait vis à vis du temps dont nous disposons pour préparer la démonstration finale. D'autres part, bien qu'il existe de nombreuses méthodes [5], trois méthodes on été testées pour pouvoir identifier des zones de copy-move : la première se base sur l'algorithme SIFT associée à un programme de clustering, la seconde se base sur la transformée en cosinus discrète et le principe de lexicographie, enfin la dernière se nomme LPB (Local Binary Pattern) qui est une méthode basée sur les blocks. Enfin, des recherche sur le CNN ont été faites, mais pas très concluantes pour l'instant. Au cours de ce rapport, nous allons passer en revue l'ensemble des méthodes testées, en données les avantages et les inconvénients, et mettre quelques illustrations.

2 Les méthodes utilisées :

2.1 Sift Clustering :

L'algorithme SIFT est un algorithme permettant de détecter les points d'intérêt dans une image et d'extraire des descripteurs de certaines zones des l'image : les zones les plus intéressantes. Il y a plusieurs moyens d'utiliser SIFT afin de déterminer les zones copiées-déplacées dans une image, mais la plupart se résument en deux mots : SIFT et cluster. Là où le plus gros changement s'effectue est dans la méthode de clustering. En effet, il existe une multitude de manières de faire : clustering en utilisant l'algorithme FCM [2], qui est très similaire à l'algorithme K-means basé sur les centroïdes des clusters et sur la convergence de l'algorithme, ou encore le clustering par échelle superposées ou basé sur les couleurs des keypoints [3]. Ci-dessous, vous pouvez voir l'obtention des points d'intérêt dans une image, ainsi que le résultat de la détection.

(a) Détection des points d'intérêt sur une image



(b) Détection de la zone falsifiée dans l'image

2.2 DCT + Lexicographical order :

Cette méthode à pour but de modéliser les valeurs des pixels dans le domaine fréquentiel grâce à la transformée en cosinus discrète [6]. L'image est divisée en plusieurs blocks de même taille, et pour chaque block on calcule un coefficient DCT. Une fois ceux-ci obtenus, nous obtenons une matrice les répertoriant. Il faut alors appliquer un tri lexicographique à cette matrice puis rechercher les blocks similaires par comparaison de coefficient. Dans le cas ou des blocks seraient similaires selon un certain seuil, alors on colorera cette partie en blanc et le reste sera mis en noir. Ceci nous permet de bien voir seulement les zones copiées-déplacées [1].

Cette méthode est cependant restée en suspend, puisque nous n'arrivions pas la comprendre parfaitement, ce qui fait que nous n'avions pas de résultats concluants et surtout, nous perdions un peu de temps à essayer tant bien que mal de comprendre, mais sans succès. Mais si nous avions réussis voilà le genre de choses que nous aurions pu obtenir :

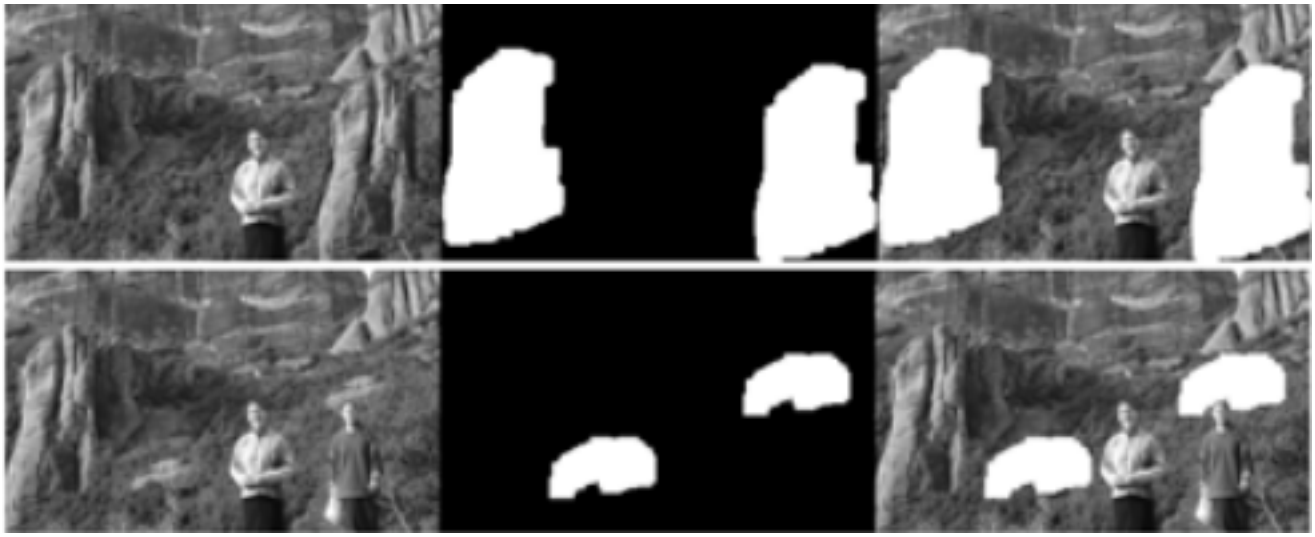


FIGURE 2 – Détection de copier-déplacer avec DCT

A gauche nous avons l'image falisifée, au centre la détection des endroits falsifiés, et à droite la vérification que ce sont les bons endroits.

2.3 Local Binary Pattern :

Cette dernière méthode est en cours d'implémentation au moment où ce rapport sera publié. Mais pour en résumer le principe, nous avons plusieurs étapes. Sachant que c'est une méthode basée sur les blocks, notre image va être divisée en plusieurs blocks de même taille, idéalement carrés. Une fois l'image divisée, nous allons appliquer sur chaque block la méthode du LBP [**lpb**]. Cela consiste à seuiller le 8-voisinage de chaque pixel, et à considérer le résultat comme un nombre binaire. La matrice 3x3 obtenue uniquement composée de 0 et 1 va être multipliée par les poids LBP et l'ensemble des éléments vont être sommées pour obtenir la valeur LBP du pixel courant [4]. Vous pouvez voir une illustration pour accompagner l'explication ci-dessus :

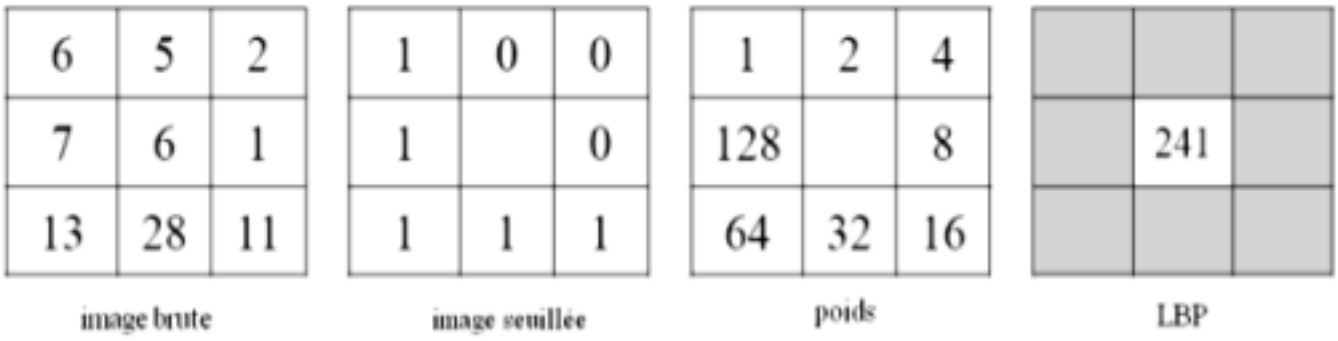
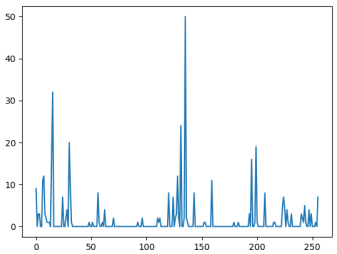


FIGURE 3 – Fonctionnement de l'opérateur LBP

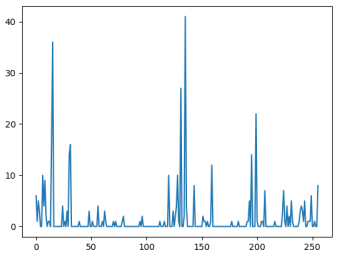
Il est important de souligné que le LBP est plus précisément un opérateur de texture.

Une fois cette étape réalisée, nous devons calculer les histogrammes de chacun des blocks. Pour ceci, c'est tout simplement la méthode basique que l'on applique. Ci-dessous vous pouvez voir les histogramme des composantes rouge, verte et bleue après LBP obtenus par nous-même sur le block 0 de l'image de poissons falfisiée :

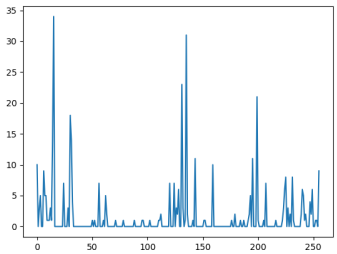
FIGURE 4 – Image falsifiée



(a) Histogramme du block 0 composante bleue



(b) Histogramme du block 0 composante verte



(c) Histogramme du block 0 composante rouge

Il est aussi important de préciser que nous utilisons les trois composantes couleur de l'image en notre faveur, cela permet de faire un filtrage beaucoup plus précis concernant les candidats à une falsification par copier-déplacer.

L'étape suivante est de calculer les distances entre chaque block. Nous avons choisis d'utiliser la distance cosinus. En effet, nous considérons chaque histogramme comme un vecteur représentant le block. Ainsi le cosinus entre deux vecteur nous permet de connaître de "pourcentage" de similitude entre deux d'entre eux. En effet, le cosinus de l'angle entre deux vecteurs se trouve dans l'intervalle [-1, 1]. Un cosinus de -1 indique des vecteurs opposés, un cosinus de 0 indique des vecteurs indépendants, et enfin un cosinus de 1 indique des vecteurs colinéaires. Bien sûr il existe aussi toutes les valeurs différentes de -1, 0 et 1, et il est logique de se dire que plus on se rapproche d'une de ces trois valeurs, plus les vecteurs tendent à être opposés, ou indépendant, ou coliéaires. L'avantage c'est qu'il existe une fonction en Python du package **sklearn.metrics.pairwise** intitulée **cosine-distances** qui nous permet facilement d'obtenir les distances entre chaque block.

Les distances entre les blocks vont nous permettre de faire ce qu'on appelle une primary candidate selection. C'est à dire que chaque nouvelle distance calculée va être comparée à la distance précédente et si celle si est inférieure, alors la paire de blocks correspondante passe en tête de liste. Une fois ce filtrage effectué, nous prenons seulement le premier quart de la liste qui correspond au blocks les plus probables d'avoir subit un copier-déplacer. D'autres filtres vont être appliqués afin d'être encore plus précis. Par exemple la distance spatiale entre les blocks va être calculée, éliminant ainsi les blocks qui sont trop proches dans l'image. Ci-dessous vous pouvez voir la forme des résultats que nous attendons ainsi qu'un schéma résumant la méthode :

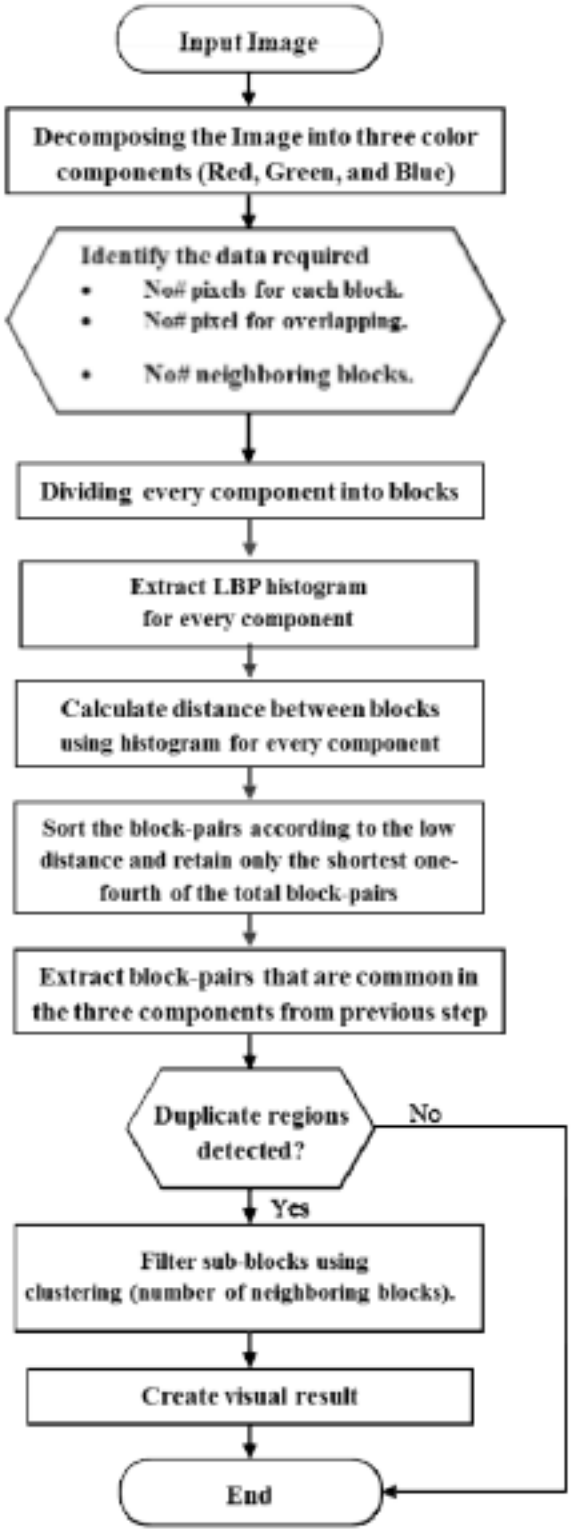


FIGURE 6 – Schéma résumant les étapes de la méthode

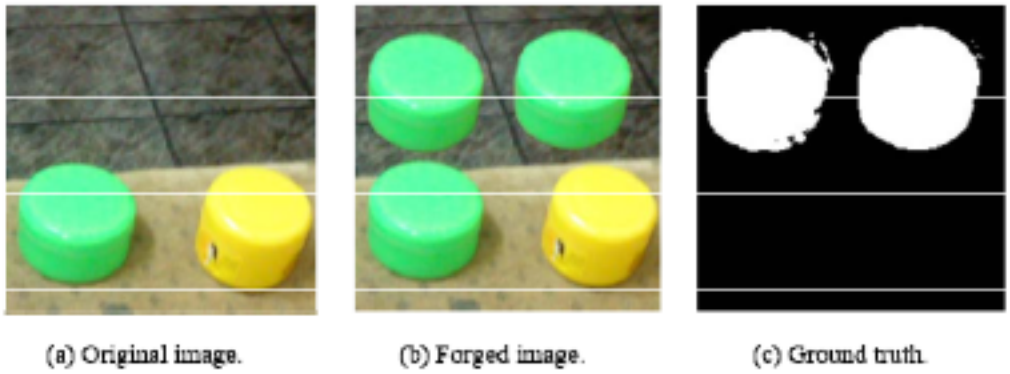


FIGURE 7 – Exemple de résultats que nous souhaitons produire

3 L'application :

Pour l'application, c'est à dire l'interface qui va nous permettre de produire notre démonstration, nous avons choisit d'utiliser la bibliothèque **tkinter** pour la générer. Cette bibliothèque est très simple d'utilisation. Pour l'instant 3 boutons y ont été ajoutés, un qui permet de browse dans les dossiers de l'ordinateur et de sélectionner une image. Un autre qui applique la méthode du sift clutering au moment du click sur l'image sélectionnée au préalable. Et un dernier pour l'instant qui est censé permettre d'appliquer l'opérateur LBP sur l'image, mais qui ne le fait pas encore puisque l'algorithme est en cours de finalisation. Vous pouvez voir ci-dessous le résultat de l'application à ce jour :

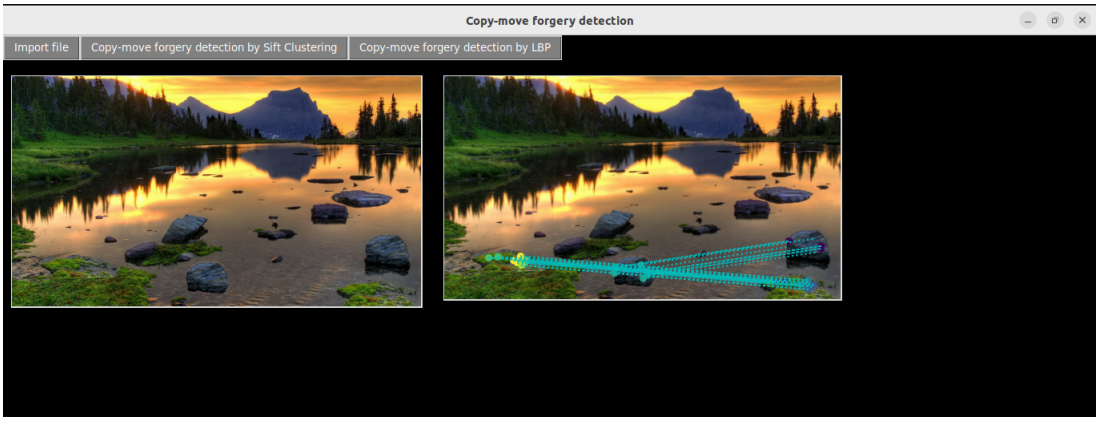


FIGURE 8 – Visualisation de l'application

D'autres fonctionnalités sont bien évidemment au programme, telles que afficher les propriétés des images, pouvoir importer plusieurs images et appliquer les algorithmes sur l'ensemble d'entre elles, et si le temps nous le permet, de calculer les VP, VN, FP, FN ainsi que de mettre en place un algorithme de deep learning.

Bibliographie

[1] S.K Vasistha Ashima Gupta Nisheeth Saxena. "Detecting Copy move Forgery using DCT". In : t. 3. 5. 2013, p. 1-5. doi : https://www.researchgate.net/publication/236632995_Detecting_Copy_move_Forgery_using_DCT.

[2] Gouda I. Salama Hesham A. Alberry Abdelfatah A. Hegazy. "A fast SIFT based method for copy move forgery detection". In : t. 3. 2. 2018, p. 159-165. doi : https://www.sciencedirect.com/science/article/pii/S2314728818300114?via%3Dihub.

[3] YINGDA LYU HAIPENG CHEN XIWEN YANG 2. "Copy-Move Forgery Detection Based on Keypoint Clustering and Similar Neighborhood Search Algorithm". In : t. 8. 2020, p. 36863-36875. doi : 10.1109/ACCESS.2020.2974804.

[4] Samira SADI Aghiles GOUNANE. "Contribution de l'approche LBP à la classification des images multispectrales. Application aux données TM de la région du Hoggar." In : 3juillet 2014.

[5] Dr. Vinay Chopra. "A Review : Block-Based Copy-Move Forgery Detection Methods". In : t. 5. October 2016, p. 50-53.

[6] Moise Fah Kue Nathalie Diane Wandji Sun Xingming. "Detection of copy-move forgery in digital images based on DCT". In : t. 10. 2. 1 March 2013, p. 1-5. doi : https://doi.org/10.48550/arXiv.1308.5661.