

Détection de copier-coller

Coni Soret

coni.soret@ensae.fr

Abstract

Ce rapport présente une méthode pour détecter des opérations de copier-coller dans des images, basée sur l'algorithme *Patchmatch*. Plusieurs critères seront présentés pour déterminer si un pixel peut être considéré comme issu d'un copier-coller. Ces critères seront finalement évalués sur un ensemble d'images présentant des opérations de copier-coller.

Introduction

Les falsifications d'images par copier-coller consistent en la copie d'une ou plusieurs parties d'une image et au collage à un autre endroit de cette même image. Le but de ces opérations est notamment de dupliquer ou de cacher un objet d'intérêt [2, 3].

Pour détecter ces copier-coller, l'idée naïve est de comparer chaque *patch* de l'image avec tous les autres. Cette méthode est très lente mais aussi inefficace car elle ne prend pas en compte le contexte du patch. En effet, ce patch peut appartenir à une zone uniforme (ciel) ou avoir été légèrement modifié, ce qui fausserait les résultats dans les deux cas.

Pour pallier à la lenteur d'une recherche gloutonne, nous utiliserons l'algorithme *Patchmatch* [1], permettant de trouver pour chaque pixel, celui qui lui correspond en terme de similarité de patch. Puis à partir de ces correspondances entre pixels d'une image, nous pourrons définir des critères permettant de déterminer la présence de copier-coller.

1. Patchmatch

Patchmatch est un algorithme randomisé permettant de rapidement construire une grille de correspondance entre patch, autrement dit, trouver le plus proche voisin de chaque patch.

Cet algorithme est composé de 3 étapes différentes qui sont répétées successivement pendant quelques itérations. En général, *Patchmatch* converge au bout d'un très faible nombre d'itérations, environ 5.

L'algorithme va d'abord initialiser aléatoirement une grille de déplacements, représentant les vecteurs de déplacement du patch à son plus proche voisin, puis successivement effectuer la propagation, la recherche aléatoire et la

symétrisation. Formellement, on cherche f une fonction de déplacement qui vérifie pour une certaine distance D :

$$f(z) = \underset{d \neq 0}{\operatorname{argmin}} D(P(z), P(z + d))$$

où $P(z)$ est le patch aux coordonnées z .

Propagation

Dans cette étape, pour chaque patch de l'image, l'algorithme compare le plus proche voisin actuel avec ceux définis par les déplacements de ses pixels voisins.

$$f(z) = \underset{d \in \{f(z), f(z^u), f(z^l)\}}{\operatorname{argmin}} D(P(z), P(z + d))$$

où z^u et z^l correspondent au pixel au dessus et à gauche respectivement. L'idée est que les déplacements optimaux des plus proches voisins sont les mêmes sur une zone, et qu'avec cette propagation, on va rapidement remplir la zone à partir d'un seul déplacement optimal trouvé.

De plus on alternera les propagations, en allant du coin en haut à gauche (en prenant z^{upper} et z^{left}) puis dans l'autre sens (en prenant z^{lower} et z^{right}).

Recherche aléatoire

Cette étape consiste à comparer l'actuel déplacement optimal avec des déplacements aux alentours. Pour cela on défini un set de déplacements à comparer de la manière suivante :

$$\mathcal{D} = \{d + 2^i r_i ; i = 0 \dots L, r_i \sim R\}$$

où R est une variable aléatoire uniforme sur $\{-1, 0, 1\} \times \{-1, 0, 1\} \setminus (0, 0)$. L'idée est de chercher aléatoirement un patch aux alentours du plus proche voisin qui serait encore plus proche.

Symétrisation

Cette étape va simplement permettre de tester si un patch n'est pas le plus proche voisin de son plus proche voisin :

$$f(z + f(z)) = \underset{d \in \{-f(z), f(z + f(z))\}}{\operatorname{argmin}} D(P(z), P(z + d))$$

Moments de Zernike

Pour calculer la distance D d'un patch à un autre, nous pourrions simplement calculer la distance euclidienne entre deux patchs. Mais, afin d'être robuste aux rotations dans les copier-coller, nous utilisons les polynômes de Zernike. En effet, les moments de Zernike ont la propriété d'être invariant par rotation [4].

Ainsi, dans la suite, pour comparer deux pixels, nous utiliserons les normes 1 entre les moments de Zernike des deux patchs autour de ces pixels, moyennées sur les trois canaux RGB.

2. Détection

Grâce à la carte de déplacements définis par *Patchmatch*, nous pouvons désormais détecter les copier-coller. Pour cela, il existe plusieurs méthodes, et je vais présenter celles que j'ai construites, en les distinguant en deux catégories, en commençant par les méthodes par différence.

Pour les différents tests, nous utiliserons des images contenant un copier-coller comme présenté en figure 1.



Figure 1. Une image et sa falsification

2.1. Méthodes par différence

Différence seule

La première idée est d'utiliser les plus proches voisins définis par *Patchmatch*, afin d'étudier si la différence entre deux pixels est inférieure à un seuil défini au préalable.

On observe sur la figure 2 que la falsification est bien



Figure 2. Copier-coller détecté par la méthode *diff*

détectée, mais que de nombreux autres pixels sont détectés. Les faux positifs sont principalement situés dans des zones plates (le ciel, la mer). On comprend que même s'il ne s'agit pas de copier-coller, il est aisé de trouver un patch très similaire à une zone plate.

Différence et écart-type

Pour pallier à ce défaut, nous utilisons une mesure de l'écart-type du patch considéré, définie comme la moyenne des écart-types sur chaque channel. On introduit un second seuil pour l'écart-type, définissant un second critère pour la décision du copier-coller.

On observe sur la figure 3 que de nombreux faux positifs ont été supprimés, mais il reste toujours du bruit. De plus, on remarque aussi que l'on a perdu une petite partie de la falsification, notamment car il peut exister des zones plates au sein même du copier-coller.



Figure 3. Copier-coller détecté par la méthode *diff-std*

Difference avec propagation

Il semble clair que la méthode décrite précédemment a ses avantages, mais il faut la compléter. Pour cela, on peut utiliser les déplacements trouvés par *Patchmatch* que l'on a très peu exploités. En effet, il est clair au vu de la figure 4, qu'une information supplémentaire est disponible quant à la constance des déplacements sur la région de copier-coller.

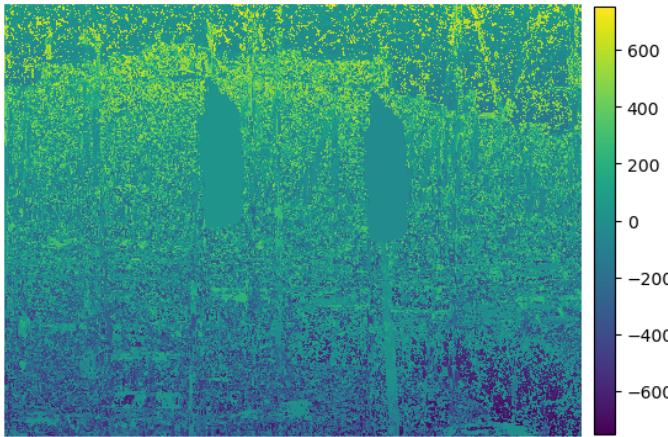


Figure 4. Grille de déplacements optimaux pour la coordonnée x

On définit alors une nouvelle méthode qui va tout d'abord trouver des pixels "certains" de falsification, puis les propager aux pixels voisins à condition qu'ils aient le même vecteur de déplacement. Pour trouver ces pixels certains, on impose des seuils très contraignants de différence et d'écart-type.

On remarque sur la figure 5 qu'il reste de nombreux faux négatifs sur les bords du copier-coller. Cela vient à la fois

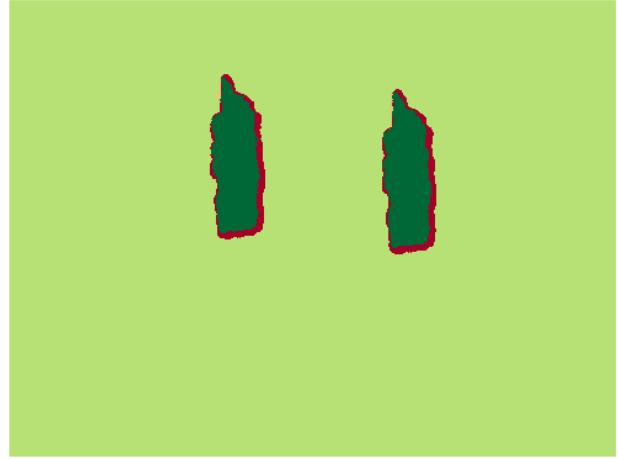


Figure 5. Vérité de la labellisation. vert clair : vrai négatif ; vert foncé : vrai positif ; rouge : faux négatif ; orange : faux positif.

du copier-coller en lui même, qui a été adouci sur les bords pour une meilleure intégration, mais aussi de la méthode par patch. Pour régler cela, on fait une dilatation à la fin pour propager les labels positifs.

La figure 6 semble montrer des résultats satisfaisants.



Figure 6. Copier-coller détecté par la méthode *prop-dilated*

2.2. Méthodes par histogrammes

Grâce aux méthodes précédentes, on a pu se rendre compte de l'efficacité des cartes de déplacement. On peut alors essayer de construire une méthode qui utilise en premier lieu ces déplacements et non pas les patchs.

Histogramme

Pour utiliser cette idée, on se sert de l'histogramme 2D des déplacements optimaux.

On remarque sur la figure 7 deux points particulièrement

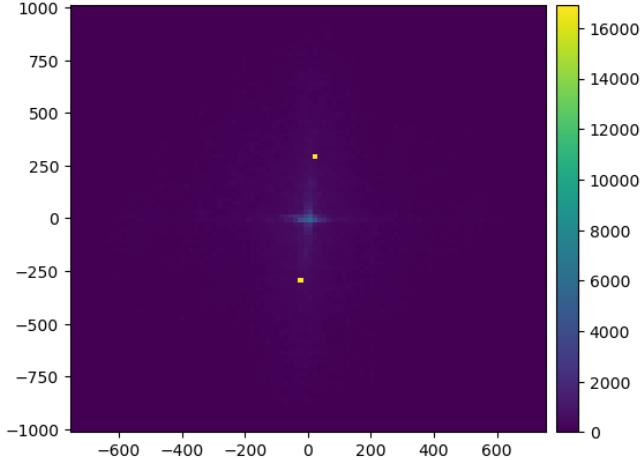


Figure 7. Histogramme des vecteurs de déplacement

denses qui correspondent exactement aux déplacements du copier-coller. Il nous suffit maintenant de détecter automatiquement ces points, puis labeliser positivement tous les pixels ayant ces vecteurs de déplacement.

Dans un premier temps, on compare chaque valeur de l'histogramme avec la valeur moyenne, et si on observe une valeur n fois supérieur à la valeur moyenne, alors on considère que c'est un vecteur issu d'un copier-coller.

Le résultat présenté à la figure 8 est assez bon, mais on



Figure 8. Copier-coller détecté par la méthode *histo*

remarque du bruit. Si on observe bien l'histogramme, on remarque des zones denses autour du 0, qui provoquent ces faux positifs. Ces densités autour du 0 sont logiques car les patchs situés à courte distance ont plus de chance d'être similaires, mais aussi car tous les déplacements ne sont pas possibles sur chaque pixel, par exemple pour un pixel situé

sur le bord.

Histogramme et NFA

Cette méthode est motivée par deux raisons. La première étant la volonté de se dispenser des seuils et la seconde étant de corriger les défauts de la méthode précédente.

Pour cela on introduit le *Nombre de Fausses Alarmes* (NFA), qui représente le nombre moyen de copier-coller présents dans un ensemble d'images. Dans notre cas, il est égal à 1 car toutes les images ont un copier-coller.

En utilisant à nouveau un histogramme, on va chercher à déterminer la probabilité d'observer au moins une telle valeur. L'idée est que pour les points très denses (ceux des vecteurs de déplacement), la probabilité sera très faible, et on pourra considérer qu'il y a une anomalie en ces points. Pour précisément estimer la probabilité d'une quantité pour un vecteur de déplacement, il faudrait regarder chaque pixel ayant ce vecteur de déplacement, le nombre de possibilités qu'il avait. Ceci est très certainement complexe, alors on peut se ramener à un problème plus simple, qui donne les mêmes résultats en espérance.

Prenons le cas d'une image de dimension 2×2 , son histogramme des vecteurs de déplacement sera alors de dimension 3×3 .

(-1,1)	(0,1)	(1,1)
(-1,0)	(0,0)	(1,0)
(-1,-1)	(0,-1)	(1,-1)

La simplification consiste à considérer que chaque point peut avoir un vecteur de déplacement quelconque, mais ceci n'est pas équiprobable et le schéma ci-dessus représente ces probabilités. Par exemple, la case $P(d = (0, 0)) = \frac{4}{16}$ alors que $P(d = (1, 0)) = \frac{2}{16}$. Ainsi, on obtient des lois binomiales pour chaque coordonnées i, j , avec n le nombre de pixels dans l'image et $p_{i,j} = \frac{(w-|i|)(h-|j|)}{(wh)^2}$ où w, h sont les dimensions de l'image. Finalement, on définit une anomalie si $F_{i,j}(x_{i,j}) \geq 1 - \frac{1}{2wh-1}$ où $F_{i,j}$ est la fonction de répartition d'une loi binomiale de paramètre $(wh, p_{i,j})$ et $x_{i,j}$ le nombre de pixels ayant le vecteur de déplacement i, j .

Les résultats présentés à la figure 9 sont bons, mais il reste du bruit.



Figure 9. Copier-coller détecté par la méthode *nfa*

Histogramme, NFA et noyau

Pour éliminer le bruit restant, on effectue une convolution avec un noyau de 1, puis on seuille le résultat pour éliminer les pixels seuls ou en petit groupe. De plus, comme avec la méthode par propagation, on dilate les pixels détectés. Les



Figure 10. Copier-coller détecté par la méthode *nfa-kernel*

résultats de la figure 10 semblent très bons.

3. Résultats

Pour évaluer les méthodes, on disposait de 80 images falsifiées ainsi que leurs labels. On a séparé les données en 2 groupes, l'un pour améliorer les méthodes et notamment définir les seuils. Puis avec les 40 autres, on a calculé la F-Measure $FM = \frac{2TP}{2TP+FN+FP}$ pour obtenir des résultats non surestimés.

Méthode	Moyenne	Médiane	Écart-type
diff	0.425	0.317	0.308
diff-std	0.547	0.548	0.248
prop	0.888	0.893	0.0254
prop-dilated	0.935	0.948	0.0350
histo	0.852	0.863	0.0554
nfa	0.814	0.842	0.0978
nfa-kernel	0.929	0.937	0.0345

Conclusion

À l'aide de l'algorithme *Patchmatch*, on a pu créer deux méthodes de détection de copier-coller performantes et rapides. Ces deux méthodes, bien qu'elles n'utilisent pas les même critères en premier lieu, donnent des résultats très proches.

L'intérêt de la méthode NFA est la signification concrète de son seuil, qui est justement le nombre de copier-coller moyen que l'on veut détecter par image. Alors que pour la méthode par propagation, le choix des seuils a été guidé par les résultats sur les images falsifiées.

Références

- [1] C. Barnes, E. Shechtman, A. Finkelstein, and D.B. Goldman. Patchmatch: a randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics*, 2009.
- [2] Davide Cozzolino, Giovanni Poggi, and Luisa Verdoliva. Copy-move forgery detection based on patchmatch. *IEEE International Conference on Image Processing*, 2014.
- [3] Davide Cozzolino, Giovanni Poggi, and Luisa Verdoliva. Efficient dense-field copy-move forgery detection. *IEEE Transactions on Information Forensics and Security*, 2015.
- [4] S.-J. Ryu, M. Kirchner, M.-J. Lee, and H.-K. Lee. “rotation invariant localization of duplicated image regions based on zernike moments. *IEEE Transactions on Information Forensics and Security*, 2013.

Annexe

Image falsifiée	Labels	prop-dilated	nfa-kernel
		