Détection de grille JPEG par compression simulée

Tina NIKOUKHAH, Miguel COLOM, Jean-Michel MOREL, Rafael GROMPONE VON GIOI

CMLA, CNRS, ENS Paris-Saclay, Université Paris-Saclay, 94235 Cachan cedex, France
{nikoukhah, colom, morel, grompone}@cmla.ens-cachan.fr

Résumé – La rétro-ingénierie est un outil efficace pour détecter les falsifications d'images. Nous proposons une méthode simple et fiable pour détecter la compression JPEG et la position de sa grille, et la comparons à l'état de l'art. La méthode peut être utilisée afin de déterminer si une image a été rognée, ce qui est souvent le premier indice de falsification.

Abstract – Reverse engineering is a useful tool for image falsification detection. We propose a simple and accurate method detecting JPEG compression and its grid origin, and compare it to the state of the art. The method can be used to determine if an image has been cropped, which is often the first indication of forgery.

1 Introduction

La falsification d'images est actuellement utilisée massivement sur le web et alimente continuellement les infox. Ce thème a pris un relief extraordinaire depuis que des outils de manipulations d'images digitales sont accessibles au grand public. Certains réseaux sociaux permettent même de retoucher les images et les vidéos directement en ligne. Ces plateformes (Facebook, Instagram, Snapchat, etc.) effacent toutes métadonnées pour motifs de confidentialité. Il est donc nécessaire pour les journalistes et les médias d'avoir accès à des outils de détection de falsifications qui ne requièrent aucune information préalable [1].

Heureusement, il reste tout de même de l'information concernant l'historique de l'image à l'intérieur de l'image elle-même. Notre approche se base sur le fait que toute opération subie par une image laisse des traces invisibles mais détectables. Ainsi, il est possible de les récupérer pour pouvoir détecter si parmi ces traces, certaines ont été altérées par une modification malveillante. Une des opérations les plus connues est la compression JPEG. Elle permet de sauvegarder une image sans avoir besoin d'une grande capacité de stockage. Nous proposons ici une méthode de détection de la grille de division de l'image en blocs JPEG 8×8. La méthode, plus simple et rapide que l'état de l'art, se base sur l'algorithme de compression même : les différentes possibilités de grilles sont testées et celle créant le fichier JPEG le plus léger est choisie.

Une brève description de l'algorithme de compression JPEG est présentée en Section 2, suivie par une revue de l'état de l'art en Section 3. Enfin, la méthode proposée est décrite en Section 4 et des résultats dont une application à la détection de recoupe d'images sont présentés en Section 5. Une discussion conclut cet article.

2 Compression JPEG

Dans le codage JPEG, après une conversion des canaux de couleurs, une image donnée est divisée en blocs de taille 8×8 pixels, chacun codé indépendamment. En raison de l'encodage indépendant, des discontinuités de pixels sont introduites à travers les limites des blocs de l'image décompressée.

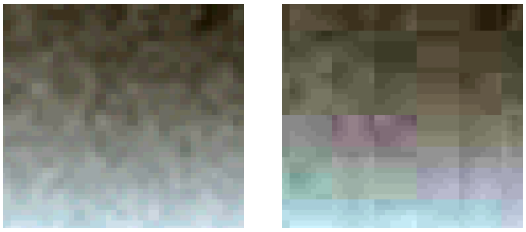


FIGURE 1 – Zooms sur une image non compressée et compressée à 80%. Le contraste des images a été rehaussé pour observer les artefacts JPEG.

La Figure 1 montre ce phénomène d'artefacts de blocs JPEG. L'image de gauche représente une partie d'une image non compressée et à droite, la même partie après compression JPEG de qualité 80%. Après un changement de contraste, on y distingue clairement des blocs de 8 pixels de côté. Le facteur de qualité JPEG correspond à son taux de compression qui est un paramètre de l'algorithme allant de 1 à 100. Plus ce taux est faible, plus le fichier résultant est léger, et plus l'image est dégradée. Lors de la compression, un compromis est fait entre poids et qualité.

L'opération subie par chacun des blocs 8×8 est appelée quantification. Elle se fait dans le domaine spectral, après application de la DCT (transformée en cosinus discrète). Un tableau de quantification (lié à la qualité de compression) fournit un facteur pour chaque composant DCT. C'est lors de cette étape de l'algorithme au cours de laquelle se produit la

plus grande perte d'information (et donc de qualité visuelle), mais c'est aussi celle qui permet de gagner le plus de place. En effet, certains coefficients DCT sont annulés lorsqu'ils ont une faible valeur par rapport au facteur de quantification (ce qui est le cas de la plupart des hautes fréquences car elles sont sujettes à de forte quantification). Chaque matrice de valeurs DCT quantifiées est balayée en zig-zag et les coefficients sont rangés sous la forme d'un vecteur dans lequel les premières composantes représentent les basses fréquences et les dernières les hautes fréquences. Puis, une compression sans perte par codage RLE (codage par plages) est appliquée afin d'exploiter la présence de longues séries de zéros en fin de vecteur. Finalement, ces données sont comprimées par un code de Huffman auxquelles est ajouté un en-tête afin de former le fichier sous le format JPEG.

3 État de l'art

Il existe plusieurs outils pour la détection des effets de la compression JPEG. Deux grandes familles en ressortent : les méthodes basées sur les histogrammes des coefficients DCT [2, 3] et les méthodes basées sur la détection d'un contraste plus fort à la limite des blocs [4, 5]. Cette dernière catégorie permet de détecter ce qui est sans doute l'un des schémas de trucage les plus couramment utilisés : le copier-coller d'images qui rompt l'alignement de la grille originale. Une autre manière de détourner une image est par son recadrage, qui permet de choisir tendancieusement une partie de la scène photographiée. Cette méthode fréquente en photo-journalisme peut modifier considérablement l'interprétation d'une scène. Pour détecter un recadrage, Li et al. [7] effectuent une extraction de la grille (BAG) et détectent le fait que son origine n'est plus $(0, 0)$. Récemment, les auteurs de [6] se sont basés sur la mesure d'artefact introduite par Fan et al. [4]. Dans un travail récent [8], nous avons appliqué le filtre proposé par Chan et al. [5] pour révéler ces artefacts de blocs avant d'appliquer une méthode statistique permettant d'augmenter la fiabilité de la détection.

4 Méthode proposée

Dans une image ayant subi une compression JPEG, les blocs 8×8 sont créés suivant un schéma régulier commençant au pixel en haut à gauche de l'image et donc coïncidant avec une grille d'origine $(0, 0)$.

Le but de la méthode est de retrouver l'étape de séparation en blocs 8×8 de l'algorithme JPEG. Cela revient à obtenir la position de la grille en donnant son origine (celle-ci peut varier si l'image est rognée). Si une grille est présente, parmi les $8 \times 8 = 64$ différentes possibilités d'origine, une seule est correcte. L'Algorithme 1 fournit un pseudo-code [1] que nous allons maintenant expliquer.

1. Code source et démo en ligne : https://github.com/tinankh/Simplest_GOD

Algorithme 1 : Détecteur d'origine de grille JPEG

entrée : image I
paramètre : $\kappa = 7$
sortie : origine de la grille (g_x, g_y)

pour $x \in \{0 \dots 7\}$ **faire**
  pour $y \in \{0 \dots 7\}$ **faire**
    $I_{x,y} \leftarrow$ recadrage de I suivant x, y
    $s_{x,y} \leftarrow$ taille-fichier$(\texttt{JPEG}(I_{x,y}, Q = 100))$
$\bar{s}_1, \dots, \bar{s}_{64} \leftarrow$ ordre-croissant$(s_{0,0}, \dots, s_{7,7})$
$\mu \leftarrow$ moyenne$(\bar{s}_{17}, \dots, \bar{s}_{64})$
$\sigma \leftarrow$ écart-type $(\bar{s}_{17}, \dots, \bar{s}_{64})$
si $\bar{s}_1 < \mu - \kappa \cdot \sigma$ **alors**
  $g_x, g_y \leftarrow$ argmin $(s_{x,y})$
sinon
  $g_x, g_y \leftarrow$ grille non trouvée

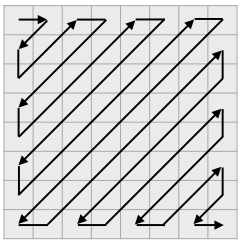
La Figure 2 illustre un exemple d'un des blocs de l'image après l'application de la DCT et un arrondi des coefficients vers l'entier le plus proche. On remarque que contrairement aux deux autres cas présentés à droite dans la Figure 2, on retrouve un nombre important de zéros dans la matrice. Ces zéros sont concentrés vers la fin du vecteur créé par zig-zag, comme le montre le schéma en haut à gauche de la Figure 2. D'après le principe même de la compression JPEG, le codage RLE rend les vecteurs avec des zéros successifs plus courts et donc cela prend moins de place lors du codage Huffman et donc dans le fichier final.

Notre proposition consiste à tester les 64 différentes origines de grilles possibles. Ainsi, étant donné une image de largeur W et de hauteur H, on crée par recadrage, 64 variantes de l'image de taille identique : de largeur $8\lfloor W/8 \rfloor - 8$ et de hauteur $8\lfloor H/8 \rfloor - 8$. Comparer la taille du fichier final après un arrondi à l'entier de leurs coefficients DCT et les ranger suivant le codage RLE, revient à effectuer une compression JPEG sans perte, c'est-à-dire avec une qualité de 100% [2]. L'utilisation de l'algorithme JPEG même nous permet une méthode simple qui bénéficie de codes de compression existants très efficaces.

Parmi les 64 variantes compressées sans perte, le fichier le plus léger correspond à celui qui a été découpé correctement : suivant l'artefact des blocs JPEG. Sur la Figure 3, la taille en octet des différentes variantes est affichée. Dans l'ordre, les trois graphes sont associés au cas où l'image d'entrée est : non compressée, compressée et enfin compressée-rognée de telle sorte que les 4 premières lignes et les 4 premières colonnes de pixels ont été retirées.

Par rapport au premier diagramme à barres, on remarque une certaine structure présente dans le deuxième diagramme. La barre rouge la plus courte correspond au fichier de plus petite taille et donc à la position exacte de la grille, ici $(0, 0)$. Les 7 barres d'après sont aussi plus courtes que les suivantes. Elles

2. Contrairement à ce qu'on pourrait penser, si l'image a subi une compression JPEG avec une qualité Q, la compresser avec ce même taux ne changerait pas la taille du fichier à la bonne grille mais réduirait celles des autres, rendant la discrimination moins efficace.



589	3	6	-3	0	2	-2	0
-16	6	0	4	-1	0	-2	-2
32	-1	4	-4	-1	1	-2	0
-4	-9	4	1	-1	-2	-4	0
-12	-15	4	3	-2	-3	-1	-1
-21	-11	7	3	-2	-4	-1	1
-23	5	8	2	1	-2	-1	0
-6	7	2	2	0	-1	0	1

non-compressée

586	-3	6	-1	0	1	-1	1
-15	5	-1	8	0	0	0	1
30	1	6	0	0	0	0	0
-7	-7	0	0	0	-1	0	0
-14	-18	1	0	0	0	0	0
-20	-14	0	0	0	0	0	0
-20	0	0	0	0	0	0	0
0	0	0	-1	0	0	0	0

bonne position (0,0)

641	-6	-15	1	-1	3	4	0
-34	-2	3	0	3	-1	-3	0
-15	1	4	2	-2	-3	2	-2
-7	3	4	-1	1	1	-1	2
-2	0	-4	-1	-2	4	0	-2
6	4	2	-3	1	-2	1	-1
4	2	-1	-2	0	2	0	-1
3	1	-1	0	0	2	0	0

mauvaise position (4,4)

FIGURE 2 – Matrices des coefficients DCT arrondis à l'entier le plus proche dans un cas non compressé, compressé avec la bonne position de grille et compressé-rogné.

correspondent aux positions testées ayant au moins une des deux coordonnées exacte (suivant x). Concernant les autres, une barre sur 8 est plus courte, cela correspond aux positions ayant l'autre coordonnée exacte (suivant y). En effet, sur le troisième graphe, on retrouve le minimum correspondant à la position correcte (ici $x = 4$ et $y = 4$) et les 15 autres plus petits correspondent aux coordonnées où $x = 4$ ou $y = 4$.

En résumé, sur une image ayant subi une compression JPEG, parmi les 64 variantes, il y a un fichier de petite taille (correspondant au x, y correct), 15 fichiers de taille moyenne (correspondant au x ou y correct, mais pas les deux), et 48 fichiers de grande taille. Nous utilisons la taille de ces derniers 48 fichiers pour définir un seuil de référence afin de valider ou non la détection selon la taille du ficher le plus petit.

Supposons les 64 tailles de fichiers ordonnées dans l'ordre croissant $\bar{s}_1, \ldots, \bar{s}_{64}$. Une détection est alors validée lorsque

$$\bar{s}_1 < \mu - \kappa \cdot \sigma,$$

où μ et σ sont, respectivement, la moyenne et l'écart-type des 48 plus grandes tailles de fichiers. Ici, nous proposons un seuil qui repose sur une valeur $\kappa = 7$ déterminée empiriquement de telle sorte à n'avoir aucune fausse détection dans des images de bruits. Ce résultat est illustré dans la section suivante.

5 Résultats

La première évaluation a été effectuée dans les cas où aucune détection ne devrait être obtenue. Cette expérience permet de fixer la valeur κ qui détermine le seuil de décision de la méthode. Le premier ensemble de données est composé de 200 images de bruit suivant une distribution gaussienne de tailles 500×500 et 1000×1000. Nous avons également utilisé les

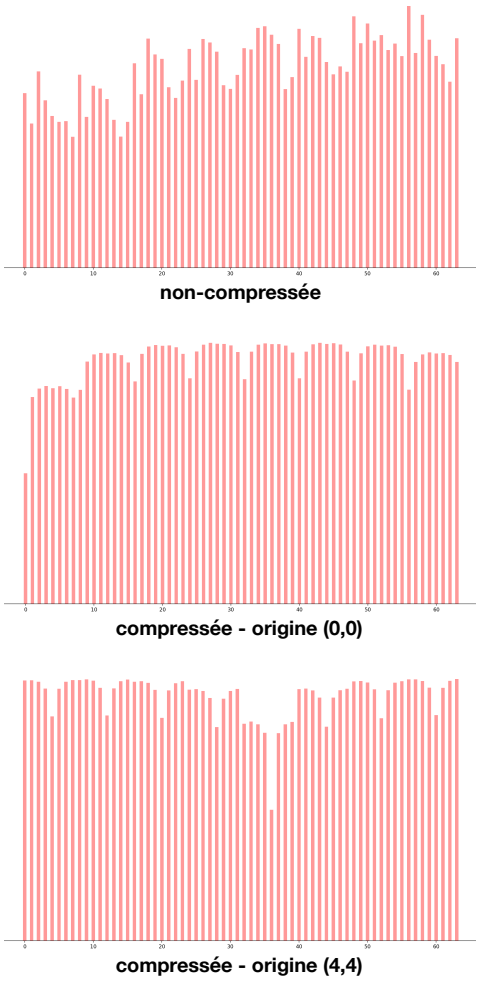


FIGURE 3 – Tailles des 64 fichiers pour l'image non compressée, compressée intacte et compressée recadrée. Chaque barre rouge représente la taille du fichier à une certaine position de la grille. Les diagrammes à barres ont été zoomés afin de mieux distinguer les différences.

collections d'images non compressées UCID [9] (886 images) et Kodak [10] (24 images). La Table 1 montre le résultat pour le seuil empirique obtenu de $\kappa = 7$. On obtient peu de fausses détections parmi la base d'images non compressées. La méthode rapide d'extraction de grille (BAG) [7] ne dispose pas de critère de décision. Dans tous les cas une position de grille est donnée et donc tout résultat est une fausse détection. En revanche, notre méthode précédente (GOD) [8] est basée sur la validation *a contrario* garantissant une non détection dans le bruit. Elle a donc un taux de faux positifs faible.

Pour illustrer la validité de l'approche proposée, nous avons effectué des détections sur 12 288 images générées à partir de la base de données d'images non compressées Kodak [10]. Les 24 images de la base ont été compressées à différents facteurs de qualité (50, 60, 70, 80, 90, 93, 95, 98 et 99), puis rognées suivant les 64 différentes positions pour tester toutes les positions de grille possibles. La Table 2 montre les résultats pour les trois méthodes comparées. La méthode BAG renvoie

		base d'images		
		bruit	UCID [9]	Kodak [10]
BAG [7]	% correct	—	—	—
	% faux	100	100	100
GOD [8]	% correct	—	—	—
	% faux	0	0.3	0
Méthode	% correct	—	—	—
proposée	% faux	0	0.04	0

TABLE 1 – Résultats de la méthode proposée comparée à BAG et GOD sur des images non compressées.

		facteur de compression JPEG				
		≤ 80	90	95	98	99
BAG [7]	% correct	97	95	85	31	0
	% faux	3	5	15	69	100
GOD [8]	% correct	100	91	70	55	41
	% faux	0	0.003	0.05	0.06	0.1
Méthode	% correct	100	100	100	50	0
proposée	% faux	0	0	0	0	0

TABLE 2 – Résultats de la méthode proposée comparée à BAG et GOD sur 12 288 images compressées et rognées générées à partir de la base de données Kodak [10].

de bons résultats pour des taux de compression jusqu'à 95%. Cependant, elle renvoie de fausses détections. En revanche, la méthode GOD cherche à contrôler ce pourcentage de fausses détections et parvient à obtenir de bons résultats pour de hautes qualités de compression. Enfin, la méthode présentée ici présente un score parfait jusqu'à un taux de compression de 95% et aucune fausse détection. Au-delà de 98%, il est difficile de distinguer une nette différence de taille de fichier. L'étape de recadrage implique qu'entre chaque version (image de même taille mais avec des origines différentes), quelques lignes et colonnes peuvent être différentes et donc on ne traite pas exactement le même contenu d'image. Ainsi, cette différence prend le dessus pour les hautes qualités de compression.

Malgré sa simplicité, la méthode proposée produit des résultats comparables à l'état de l'art et meilleur pour la majorité des taux de compression. Une application à un cas de manipulation d'information est illustrée dans la Figure 4 : la photographie de droite a été rognée afin d'obtenir l'image de gauche qui ressemble à l'ombre d'un requin. Pour cette dernière, notre méthode détecte une grille d'origine $(4, 7)$ et donc décide que l'image a été rognée. En effet, l'origine de la grille JPEG est toujours $(0, 0)$ sauf si l'image a subi un recoupage. On détecte bel et bien une grille JPEG de position $(0, 0)$ dans l'image globale.

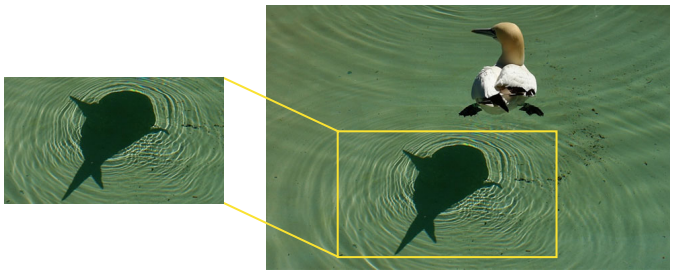


FIGURE 4 – Image sortie de son contexte.

6 Conclusion

Une méthode de détection d'origine de grille JPEG a été proposée. Elle est basée sur les tailles des fichiers JPEG obtenus après compression sans perte des 64 variantes recadrées de l'image. Aucune information concernant le format de l'image initiale n'est nécessaire. L'algorithme proposé permet à lui seul de détecter la présence d'une compression JPEG et d'en donner l'origine de sa grille. Ainsi, la solution proposée peut être utilisée comme un algorithme autonome pour détecter les opérations sur les recoupes d'images, ou elle peut être insérée dans une chaîne de traitement avancé typique pour une détection complexe et locale d'altération. Pour cela, il serait intéressant d'accélérer la méthode car la complexité actuelle revient à celle de 64 compressions JPEG.

Remerciements. Les auteurs tiennent à remercier le soutien financier du projet de recherche ANR-16-DEFA-0004 Signatures d'images du challenge ANR/DGA DEFALS.

Références

[1] H. Farid. *Photo forensics*. MIT Press. 2016.

[2] S. Ye, Q. Sun et E. C. Chang. *Detecting digital image forgeries by measuring inconsistencies of blocking artifact*. IEEE ICME, pp. 12–15, 2007.

[3] T. Bianchi, A. De Rosa et A. Piva. *Improved DCT coefficient analysis for forgery localization in JPEG images*. ICASSP, pp. 2444–2447, 2011.

[4] Z. Fan et R. L. de Queiroz. *Identification of Bitmap Compression History : JPEG Detection and Quantizer Estimation*. IEEE TIP, 12(2), 230–235. 2003.

[5] Y. L. Chen et C. T. Hsu. *Image Tampering Detection by Blocking Periodicity Analysis in JPEG Compressed Images*. IEEE MMSP, pp. 803-808, 2008.

[6] C. Iakovidou, M. Zampoglou, S. Papadopoulos et Y. Kompatsiaris. *Content-aware detection of JPEG grid inconsistencies for intuitive image forensics*. J. Vis. Commun. Image Represent., 54, p. 155–170, 2018.

[7] W. Li,Y. Yuan, et N. Yu. *Passive detection of doctored JPEG image via block artifact grid extraction*. Signal Processing, 89(9), 1821-1829, 2009.

[8] T. Nikoukhah, R. Grompone von Gioi, M. Colom et J-M. Morel. *Automatic JPEG Grid Detection with Controlled False Alarms, and Its Image Forensic Applications*. IEEE-MIPR, pp. 378–383, 2018.

[9] G. Schaefer, M. Stich. *UCID : An uncompressed color image database*. Storage and Retrieval Methods and Applications for Multimedia, vol. 5307, pp. 472-481, 2004.

[10] Kodak. *Kodak Lossless True Color Image Suite*. 1999.