Méthodes de détection de falsifications d'images numériques

Simon Hamery, Tristan Cabantous,
Nicolas Cellier

Rapport de TER

Master Informatique

Encadrants : William Puech, Olivier Strauss, Vincent Itier

Équipe Sosiji
Département informatique
Université de Montpellier
France

mars/avril 2018

Méthodes de détection de falsifications d'images numériques

Groupe Sosiji

Proposé dans le cadre du TER de master informatique
Master 1 2017/2018

Résumé

Le projet *Fake ? Yes or No ?* consistait en l'élaboration d'un programme informatique permettant de déterminer si une image a été falsifiée ou non. Notre travail a été de s'intéresser aux techniques déjà existantes permettant de traiter ce problème, et d'en proposer des nouvelles combinant les précédentes, voir entièrement nouvelles. Nous nous sommes particulièrement intéressés à l'étude du bruit d'une image pour déterminer si une image a été trafiquée, et identifier la zone falsifiée dans l'image.
Mots-clés : *Traitement d'image, Détection d'images falsifiées, Bruit d'une image*

Abstract

The *Fake ? Yes or No ?* project consisted of the development of a computer program in order to determine if an image was falsified or not. Our work has been to take an interest in the already existing methods with the intention of treating this problem, and to propose new methods combining the previous ones, or even new ones. We were particularly interested in studying the noise of an image to determine if an image was tampered or not, and identify the falsified area in the image.
Keywords : *Image processing, Falsified image detection, Image noise*

Déclaration

Le travail de ce rapport est basé sur des recherches menées par l'équipe Sosiji, groupe d'étudiants du département informatique de la faculté de Montpellier. Aucune partie de ce rapport n'a été publiée ailleurs pour n'importe quel autre diplôme ou qualification. Il s'agit entièrement de notre propre travail excepté pour les parties où des références ont été placées dans le texte.

Copyright © 2018 by HAMERY, CABANTOUS, CELLIER.
"Les droits de copyright sur ce rapport doivent rester à leurs auteurs. Aucune citation en provenance de l'ensemble du rapport ne peut être publiée sans l'accord et le consentement de ses auteurs et l'information qui lui en est extraite doit être reconnue.".

Remerciements

Nous adressons nos remerciements tout particulièrement à Vincent Itier, Olivier Strauss et William Puech de nous avoir encadrés et soutenus lors de ce projet, pour leur patience et leur aide. Nous remercions également toutes les personnes qui nous ont conseillés et aidés tout le long de ce projet.

Nous vous souhaitons une agréable lecture de ce rapport.

L'équipe Sosiji

Table des matières

Table des figures

Table des algorithmes

Chapitre 1

Introduction

1.1 Présentation du contexte TER

Le TER est un module du second semestre de Master 1 Informatique de l'Université de Montpellier. Ce module est destiné à faire travailler le groupe sur plusieurs plans : recherche scientifique, rédaction du rapport scientifique que vous êtes en train de lire et travail en équipe. Ce rapport traitera donc du sujet "FAKE ? YES OR NOT" proposé par Olivier STRAUSS, suite à notre demande.

1.2 Présentation de l'équipe

L'équipe Sosiji est composée des membres suivants : **Nicolas CELLIER** - M1 IMAGINA, **Tristan CABANTOUS** - M1 IMAGINA et **Simon HAMERY** - M1 IMAGINA. L'équipe encadrante est composée des chercheurs suivants : **Olivier STRAUSS**, **Vincent ITIER** et **William PUECH**.

1.3 Choix du sujet

Nous avons suivi le cours de *Traitement du signal* au premier semestre. Cette matière nous a donné envie de continuer à travailler sur le traitement des signaux, et plus particulièrement des images. De plus la matière *Traitement des images* du semestre 2 d'IMAGINA nous permet d'avoir une base solide pour appréhender le sujet. Participer à un sujet de recherche moderne et d'actualité nous a d'autant plus motivé à choisir ce sujet.

1.4 Présentation du sujet

Les fake news deviennent notre environnement quotidien. Les réseaux sociaux en sont pleins. Souvent ces fake news sont associées à des fake pics, c'est-à-dire des photos qui ont été trafiquées pour en modifier le contenu informatif. L'exemple le plus connu est la disparition et la réapparition de dirigeants russes sur des photos officielles.

FIGURE 1.1 – Exemple de photo retouchée (originale à gauche, retouchée à droite)

L'objectif principal du projet a été de faire en premier lieu un état de l'art des différentes méthodes de détection de falsification d'images et dans un second temps d'y apporter notre contribution en essayant d'améliorer les méthodes existantes, et possiblement de réaliser de nouvelles méthodes.

Chapitre 2

État de l'art

Nous avons tout d'abord cherché à comprendre le problème posé et à prendre connaissance des différents travaux déjà réalisés sur la détection de falsification d'images. L'état de l'art nous a permis de mieux entreprendre notre développement par la suite.

2.1 Historique

2.1.1 Mise en scène falsifiée

Avant toute modification digitale, les premières falsifications furent effectuées directement dans la mise en scène photographique dès le milieu du 19e siècle. Hippolyte Bayard, pionnier de l'histoire de la photographie, proposa la première mise en scène faisant croire au public à son suicide.

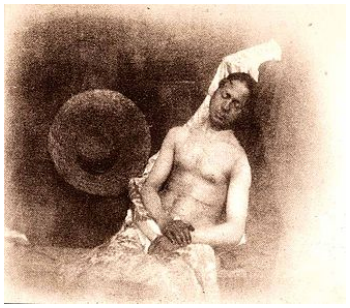
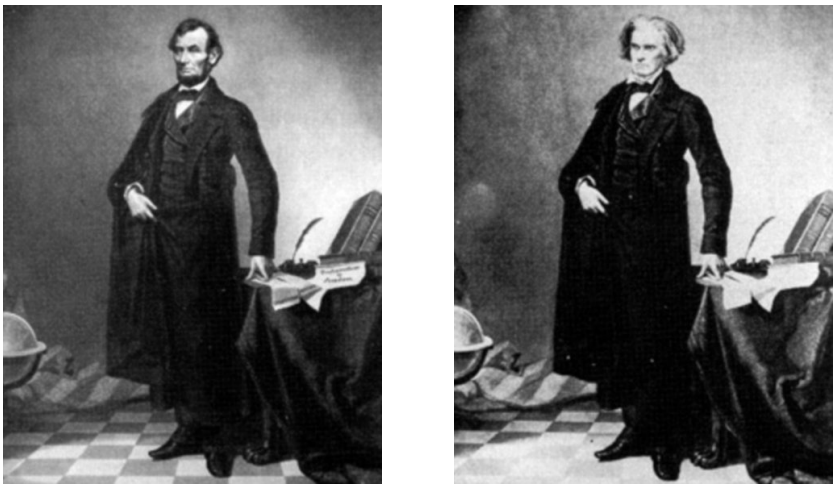


FIGURE 2.1 – Première mise en scène photographique : La « noyade » de Bayard, réalisée en 1840

2.1.2 Image falsifiée numériquement

Avec l'apparition des logiciels de retouches d'images comme *Photoshop* à la fin du XX^{e} siècle, le nombre d'images retouchées circulant sur internet ne cesse de croître. Ces logiciels permettent de manipuler des images à moindre coût. Des images historiques falsifiées bien connues telles que la photo des dirigeants Russes marquent encore les esprits. Ci-dessous une photos du président Abraham Lincoln dont la tête a été apposée par "copier-déplacer", méthode de falsification que nous expliciterons dans la partie suivante.



(a) (b)

FIGURE 2.2 – President Abraham Lincoln (gauche) composition de la tête de Lincoln et du corps du politicien John Calhoun (image de droite)

2.2 Les différents types de falsification

Dans ce rapport nous utiliserons le terme "falsification" pour tout type d'image modifiée. Dans la littérature on peut aussi retrouver le terme **forgerie**. La forgerie implique une création de toutes pièces par un faussaire pour produire un élément authentique mais peut aussi être issue d'une ou plusieurs parties authentiques existantes laissant croire que l'élément modifié est l'original. On distingue deux types de falsifications : les falsifications **endogènes**, aussi appelées **copier-déplacer** ("copy-move") et les falsifications **exogènes** appelées **copier-créer** ("copy-create").

2.2.1 Falsifications endogènes

Pour ce type de falsification, la personne copie une zone d'une image et la colle à un autre endroit de cette même image, il peut ainsi dupliquer un objet, ou bien cacher une partie de l'image.

FIGURE 2.3 – Exemple de falsification endogène ("copier-déplacer").

2.2.2 Falsifications exogènes

Aussi appelée “composition”, pour ce type de falsification, la personne prend une ou plusieurs images et copie-colle ces images sur différentes zones pour créer une nouvelle image falsifiée. C'est une sorte de fusion de plusieurs images.

FIGURE 2.4 – Exemple de falsification exogène ("copier-créer") : la troisième image est la fusion d'une zone de la première image sur la deuxième image

2.3 Méthodes de détection de falsification existantes étudiées

Certaines techniques de traitement d'image et de détection sont plus ou moins adaptées selon les types de falsification. Ces différentes techniques vont être explicitées dans les sections qui suivent.

Dans l'état de l'art actuel, les méthodes existantes de détection de falsifications exogènes et endogènes sont traitées par la mise en correspondance entre la partie copiée ou déplacée et l'autre partie de l'image non modifiée. Pour se faire, un algorithme de mise en correspondance de points appelé **SIFT** (Scale-invariant feature transform) est souvent utilisé pour diverses applications.

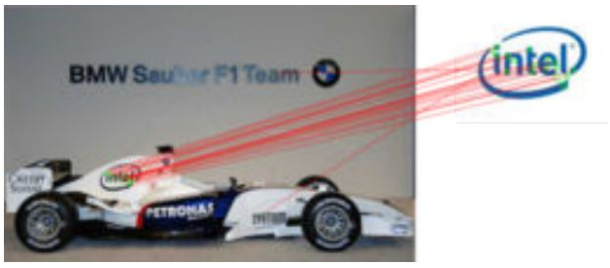


FIGURE 2.5 – Exemple de correspondance de points avec l'algorithme SIFT

L'algorithme SIFT ou transformation de caractéristiques visuelles invariante à l'échelle en français a fait ses preuves dans plusieurs domaines comme par exemple

l'assemblage de photos, la modélisation 3D, la recherche d'image par le contenu et le tracking video. Cet algorithme est utilisé pour identifier des points ou des zones similaires entre deux images sans prendre en compte l'échelle de l'image traitée, comme son nom l'indique.

2.3.1 Détection de falsification avec les contours

Il s'agit de localiser des zones d'intensités lumineuses discontinues sur une image pour déterminer des contours [1]. A l'aide de ces contours, on peut alors déterminer la zone falsifiée.

Il existe 2 grandes familles de détection de contours :

— La recherche des extremums de la dérivée première,

— et l'annulation de la dérivée seconde.

Il existe par exemple le **Filtre de Prewitt** qui utilise la dérivée première, et qui permet de localiser les contours d'une image numérique (échantillonnée).Ce filtre est composé de deux masques sous forme de matrices, l'une va calculer la dérivée en X, et l'autre la dérivée en Y d'une image.

Soit **A** notre image source, pour obtenir G_x l'image dérivée de A en x et G_y l'image dérivée de A en y on va convoluer les matrices suivantes avec A :

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} * A \qquad G_y = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} * A$$

Nous pouvons alors étudier les extremums de ces images, pour définir les contours d'une partie falsifiée dans une image. Cependant, la détection de contours pour déterminer une zone splicée fonctionne dans des cas très naïfs et irréalistes, d'autres méthodes sont souvent privilégiées.

1. Contour : On observe un contour lorsque l'intensité lumineuse des pixels passe d'une grande à une petite valeur (ou inversement).

2.3.2 Détection de falsification par l'analyse spectrale

L'analyse spectrale utilise la DFT [2] (Transformée de Fourier Discrète) pour mettre en valeur des variations des niveaux de luminosité et d'intensité et remarquer un pattern périodique et/ou des maximums locaux. Ces zones suspectes permettent la détection de ré-échantillonnage. La DFT souligne ainsi les changements sur des zones coupées ou mises à l'échelle.
Farid et Popescu [7] ont proposé une technique efficace qui détecte les falsifications endogènes. A l'aide d'une carte de probabilité calculée à partir d'une corrélation entre les voisins des pixels et différents échantillons périodiques issus de la transformée de Fourier, les basses fréquences de bruit sont ainsi écartées. Ces basses fréquences pouvaient engendrer des faux positifs [3].
L'analyse spectrale est sensiblement plus efficace sur des images non compressées ou compressées sans pertes.

2.3.3 Détection de falsification via les incohérences de luminance

La luminance d'une image est une grandeur correspondant à la sensation visuelle de luminosité d'une surface. Comme l'article de G. Peterson [4] le souligne, lorsque deux images sont prises de différentes caméras avec des éclairages différents, une divergence de luminance peut être mise en évidence lorsqu'une falsification endogène est appliquée. Cette analyse met l'accent sur les zones de distances similaires et qui présentent des niveaux de luminance qui diffèrent.

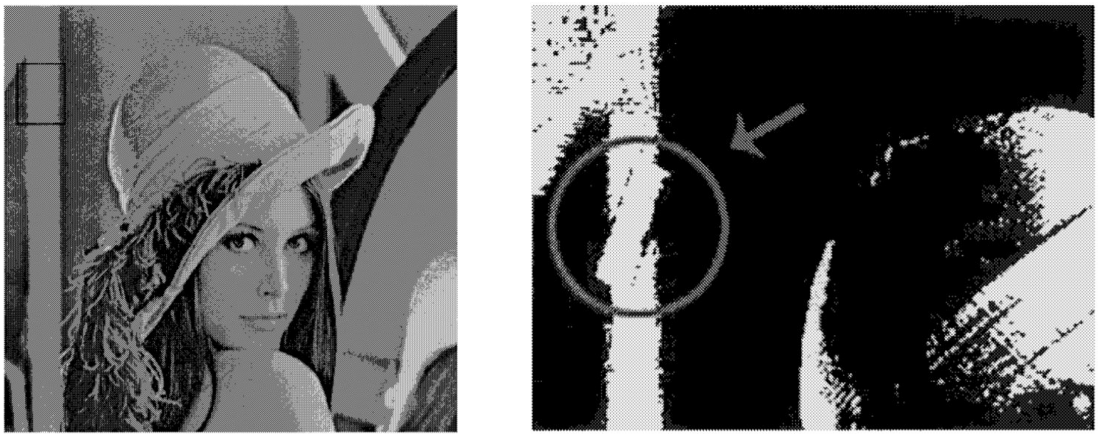
Pour détecter une falsification par des incohérences de luminance, il est possible de convertir l'image en niveaux de gris, puis de la binariser en noir et blanc à l'aide d'un seuil (entre 0 et 1). Si le pixel étudié a une valeur qui dépasse le seuil, sa valeur

2. Transformée de Fourier Discrète : Permet de représenter une signal dans le domaine fréquentiel. Pour les images, il s'agit de fréquences spatiales.

3. Faux-positifs : Dans notre cas il s'agit de zones considérées comme falsifiées par l'algorithme alors qu'elles ne le sont pas en réalité

est mise à la valeur maximale possible en niveaux de gris (255), ainsi il sera blanc. Sinon sa valeur sera mise au minimum(0) et dans ce cas là, il sera noir.

Une méthode remarquable est la méthode [7] qui permet de trouver le seuil de niveau de gris en minimisant la variance [4] entre les pixels blancs et les noirs. L'algorithme de cette méthode admet que l'image à binariser ne contient que deux classes de pixels, (le premier et l'arrière-plan) et en déduit par calculs un seuil optimal qui sépare ces deux classes afin que leur variance intra-classe soit minimale. On peut alors détecter des zones avec un niveau de luminance anormal par rapport à la moyenne.



(a) (b)

FIGURE 2.6 – Exemple de détection falsification par incohérence de luminance

Sur la figure 2.6 (b) le motif de luminance entouré d'un cercle est visiblement anormal. Mais de nos jours, les ajustements automatiques d'éclairages rendus accessibles par les logiciels de retouche d'images ne permettent pas de distinguer facilement les falsifications d'images via la caractéristique de la luminance uniquement : comme pour la totalité des méthodes étudiées, il est nécessaire de la combiner ou d'utiliser d'autres méthodes plus efficaces.

4. variance : mesure qui caractérise la dispersion d'un échantillon ou d'une distribution.

2.3.4 Détection de falsification via le modèle TSV (Teinte-Saturation-Valeur)

Pour détecter des falsifications, on peut aussi évaluer les couleurs dans un autre système de gestion de couleurs informatique. Le modèle **TSV**, teinte-saturation-valeur est basé sur une approche différente du modèle classique RVB (rouge vert bleu). La **teinte** est la valeur de l'angle sur le cercle chromatique (par exemple 0° correspond au rouge, 60 ° au jaune etc.). La **saturation** est "l'intensité" de la couleur. On peut la considérer comme un niveau de "pureté", elle varie entre 0 et 100% et plus elle est faible, plus l'image sera grisée et d'apparence "fade". Enfin, la **valeur** est l'équivalent de la "brillance" de la couleur, elle varie elle aussi entre 0 et 100% : plus elle est faible, plus la couleur est sombre (le zéro correspond au noir).

Comme pour la luminance, une zone d'une image falsifiée peut être identifiée grâce à son modèle TSV qui diffère de l'image originale. On détecte les zones avec des valeurs anormales de couleur et de luminosité.

(a) (b)

FIGURE 2.7 – Exemple de détection falsification avec le système de gestion de couleurs TSV

Sur la figure ci-dessus, on distingue bien un motif inégal et des couleurs anormales sur l'image (b) résultat du test par TSV. Preuve flagrante d'une altération de

l'image originale.

2.3.5 Compression JPEG

Format JPEG

Il s'agit d'une compression avec pertes durant laquelle l'image est divisée en bloc de 8 x 8 pixels et une conversion est effectuée depuis le modèle colorimétrique de l'image d'origine (usuellement le RVB) vers un autre modèle tel que le YCbCr (Y est la luminance et Cr et Cb sont la chrominance). Cette conversion exploite le fait que l'oeil humain est peu sensible à la propriété de chrominance. Un gain de place est donc possible en effectuant cette conversion.

Détection de falsification via la décompression JPEG

La détection de falsifications explicitée par G.Peterson [4] utilise les facteurs de qualité de compression JPEG (présentés par Fan et Queiroz [9]) qui peuvent varier selon les images et qui sont identifiables lors d'une falsification exogène tout en respectant les blocs de 8 x 8 pixels. Ainsi une zone d'une image A copiée dans une image B peut avoir un facteur de qualité de compression JPEG différent du reste de l'image B et permet ainsi d'identifier facilement qu'il y a eu une altération au niveau de la zone de l'image A.

2.3.6 L'étude de la densité du bruit

Le bruit d'une image numérique est la présence d'informations parasites qui s'ajoutent aux détails de l'image. Le bruit numérique est principalement dû à des contraintes physiques comme le capteur de l'appareil photo. Il peut aussi provenir du traitement numérique subit par l'image, comme par exemple une compression.
Des méthodes proposées par T. Julliand [1] comparent le bruit local d'une image au bruit global de cette même image afin de déterminer une zone où le bruit est

différent. On peut alors déterminer si une image est falsifiée ou non et identifier les zones altérées.

Chapitre 3

Rapport de conception

3.1 Choix de la méthode implémentée

Pour commencer notre implémentation nous nous sommes basés sur une méthode déjà existante : **L'étude de la densité du bruit (cf. 2.3.6)**. Nous nous sommes alors inspirés du travail de **Thibault Julliand** dans sa thèse *Automatic noise-based detection of splicing in digital images*.
La thèse de Thibault Julliand présente une méthode basé sur l'étude du bruit pour détecter si une image est falsifiée. L'objectif est d'analyser des histogrammes basés sur la densité du bruit local. Le but étant d'utiliser la propriété d'uniformité du bruit pour détecter des zones différentes. A l'aide de ces histogrammes, on peut alors détecter une zone dont le bruit local suit une répartition différente, et ainsi déterminer une zone falsifiée. Nous avons choisi de partir de cette méthode car elle nous semblait réalisable dans le temps imparti du projet, et la thèse de Thibault Julliand était un bon moyen pour nous de savoir comment envisager notre projet.

3.2 Structure générale des prototypes

La majorité de nos prototypes suit le processus suivant : tout d'abord, nous avons lu les images en entrée et stocké les valeurs de chaque pixel dans un tableau.

Ensuite, nous avons réalisé une extraction du bruit sur l'image que nous voulions traiter afin d'obtenir une image de bruit sur laquelle travailler. Ensuite, nous avons fait différents traitements sur cette image que nous avons stocké dans un tableau, et créé une ou plusieurs images résultant de nos traitements.

```
Data: image à tester
Result: image binaire résultat
read(imgIn) ;
imgBruit = extractBruit(imgIn) ;
expansionDynamique(imgBruit) ;
// Calcul des caractéristiques globales de l'image (Par exemple
   les densités de probabilité)
densiteGlobale = calcDensite(imgBruit) ;
// Pour tous les pixels de l'image de bruit
foreach pixel in imgBruit do
    // Calcul des caractéristiques locale du pixel
    densiteLocale = calcDensite(pixel.getVoisinnage()) ;
    // Comparaison des caractéristiques globales et locales
    if abs(densiteGlobale - densiteLocale) > seuil then
        imgOut[pixel] = 255;
    else
        imgOut[pixel] = 0;
    end
end
// Amélioration du résultat
ouverture(imgOut);
fermeture(imgOut);
// Ecriture de l'image résultat
write(imgOut);
```

Algorithm 1: Structure générale des prototypes

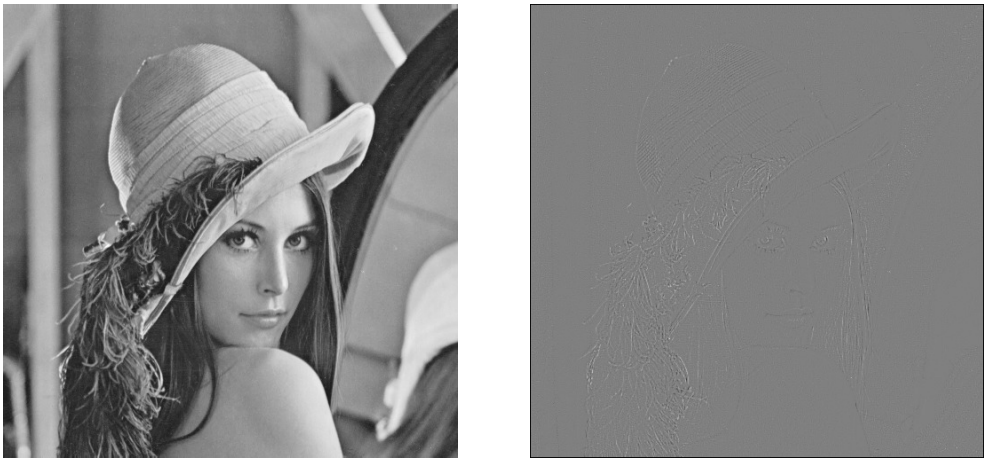
3.3 Les étapes de l'implémentation

Nous avons tout d'abord cherché à savoir comment caractériser le bruit d'une image, pour ensuite pouvoir l'étudier. Nous avons travaillé sur des images en niveaux de gris pour simplifier le travail sur le premier prototype puis ensuite sur des images couleur.

3.3.1 Caractériser le bruit d'une image

Pour isoler le bruit d'une image nous avons d'abord commencé par appliquer un filtre médian [1] à notre image, puis nous avons fait la différence de l'image obtenue avec l'image d'origine.
La différence des deux images permet de ne conserver que la valeur du bruit pour chaque pixel. Pour visualiser cette image de bruit, nous avons également ajouté 128 à cette valeur afin d'obtenir une image centrée à 128 (car les valeurs du bruits peuvent être négatives).
Nous avons fait le choix d'utiliser un filtre médian car c'est un filtre très souvent utilisé pour la réduction de bruit, de plus il possède des propriétés intéressantes car il permet d'éliminer les valeurs aberrantes sans contaminer les valeurs voisines avec cette valeur aberrante, contrairement au filtre moyen. Cependant, l'image de bruit obtenue est fortement influencée par la texture et les contours.

1. Le filtre médian est un filtre numérique, souvent utilisé pour la réduction de bruit. Le principe de ce filtre est de remplacer chaque entrée par la valeur médiane de son voisinage.



(a) (b)

FIGURE 3.1 – Exemple d'extraction du bruit sur une image en niveaux de gris

3.3.2 Traitement sur les images en niveaux de gris

Repérer une falsification par signature du bruit d'une image

Notre première approche a été de comparer des critères locaux de cette image de bruit avec des critères globaux (comme par exemple la moyenne, l'écart type ou la variance). Si la différence entre le critère local et son équivalent global est supérieure à un seuil définit (fixé empiriquement dans un premier temps), cela signifie qu'il y a de fortes chances que ce pixel appartienne à une partie modifiée de l'image. Cependant, cette méthode n'étant pas assez précise nous avons étudié les histogrammes de bruit pour déterminer une falsification d'image.

L'étude des histogrammes de bruit

Nous avons ensuite appliqué ce principe à l'étude des histogrammes, et nous avons donc comparé les densités de probabilité du bruit au voisinage de chaque pixel avec la densité de probabilité du bruit globale. En effet, chaque image possède une densité de probabilité de bruit différente, une zone de l'image possédant une densité de probabilité de bruit différente du reste de l'image a de fortes chances d'avoir été

falsifiée.

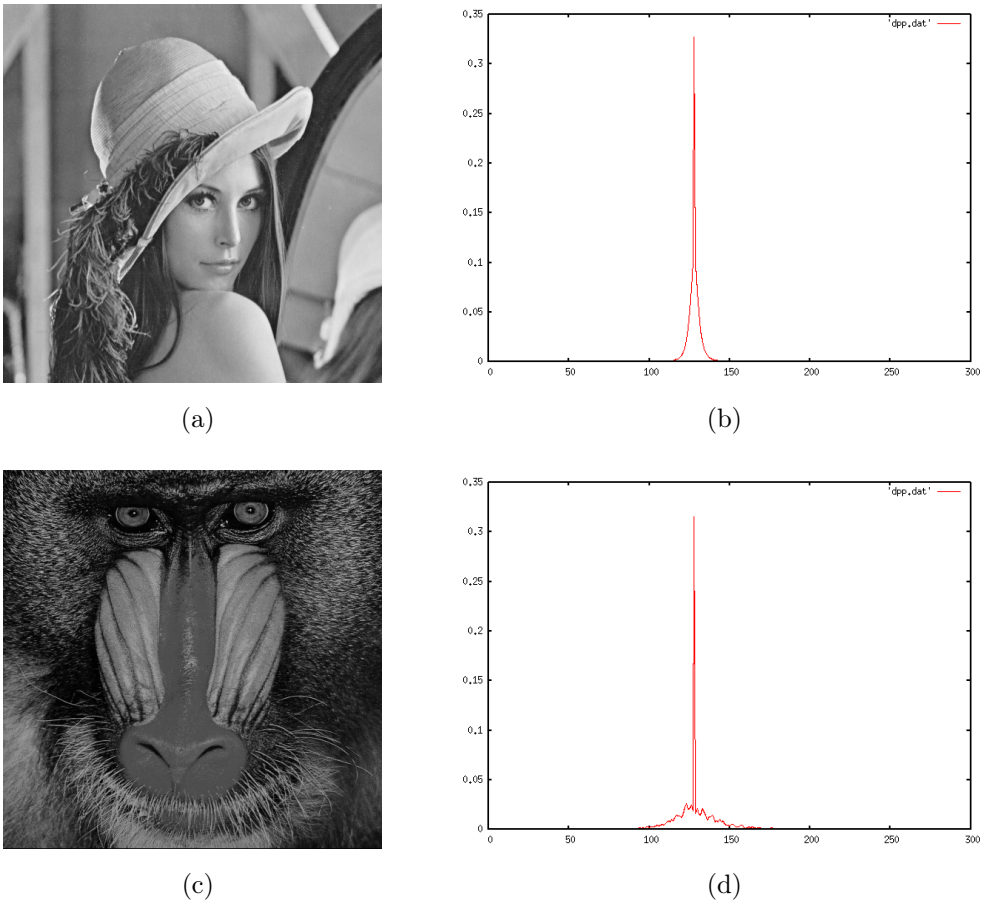


FIGURE 3.2 – Exemple de densités de probabilité de bruit

Pour analyser les densités de probabilité de bruit, nous allons tout d'abord calculer l'histogramme global du bruit de l'image. Ensuite, il nous faut définir la densité globale de ce bruit et stocker nos données dans un fichier *.dat*. Nous allons alors calculer la différence entre la densité de probabilité globale avec celle de blocs de pixels de taille variable dans notre image. Enfin, nous allons afficher les pixels en blanc si la différence est supérieure au seuil.

```
for (int i = 0; i < 256; ++i) {
    histoGLobale[i] = 0 ;
}
for (int i=0; i < nH; i++) {
    for (int j=0; j < nW; j++) {
        histoGLobale[ImgBruit[i*nW+j]] ++ ;
    }
}
double somme = 0 ;
for (int i = 0; i < 256; ++i) {
    densiteGlobale[i] = (double) histoGlobale[i] / nTaille ;
    somme += densiteGlobale[i] ;
}
double densiteMoyenne = somme/256 ;
FILE* fichier = fopen("dpp.dat", "w+");
for (int i = 0; i < 256; ++i) {
    fprintf(fichier, "%f\n", f[i]);
}
fclose(fichier);
```

Algorithm 2: Remplissage de la densité globale du bruit

```
for (int i = n; i < nH - n; i++) {
    for (int j = n; j < nW - n; j++) {
        for (int k = 0; k < 256; k++) {
            histoLocal[k] = 0 ;
        }
        for (int k = -n; k <= n ; k++) {
            for (int l = -n ; l <= n ; l++) {
                histoLocal[ImgBruit[(i+k)*nW+(j+l)]] ++ ;
            }
        }
        for (int k = 1; k < 256; k++) {
            densiteLocale[k] = (double) histoLocal[k] / nTailleCarre ;
        }
        somme = 0 ;
        for (int k = 0 ; k < 256 ; k++ ) {
            dif[k] = absDouble(densiteGlobale[k] - densiteLocale[k]);
            somme += dif[k] ;
        }
        difMoyenne = (double)somme/256 ;
        if (difMoyenne > seuil) {
            ImgOut[i*nW+j] = 255 ;
        } else {
            ImgOut[i*nW+j] = 0 ;
        }
    }
}
```

Algorithm 3: Calcul de la densité locale et comparaison

Résultats obtenus

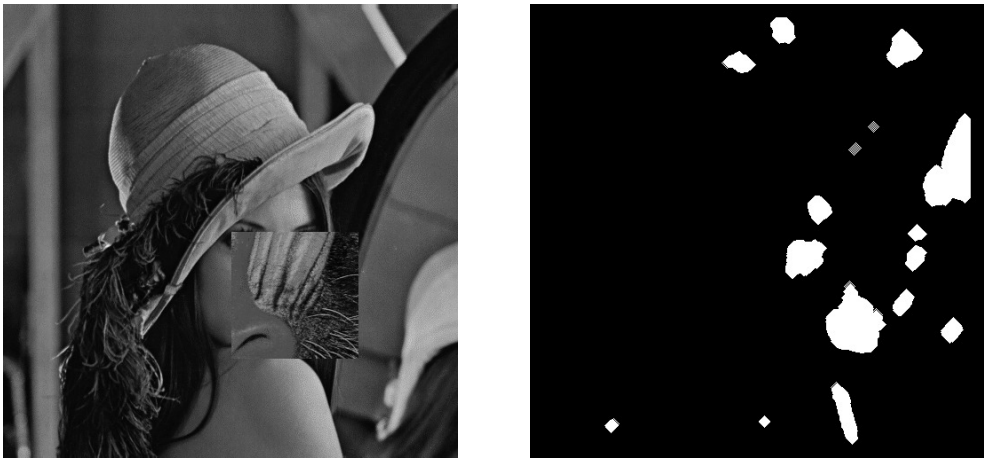
Dans un premier temps nous avons testé notre programme en modifiant une zone de l'image. Nous avons augmenté le bruit de cette zone pour voir si notre mé-

thode de détection était fiable. Ensuite nous avons testé la méthode sur une image que nous avons falsifié par un simple *copier-coller* de deux images et par une image falsifiée obtenue sur la base Columbia.

Notre programme prend une image en entrée et va sortir une image seuillée (binaire composée uniquement de pixels blanc et noir). Dans cette image binaire, les pixels en blanc sont considérés comme positifs : on considère qu'ils sont différents de l'image originale donc qu'ils ont été rajouté dans l'image et les pixels noir comme des négatifs, c'est à dire qu'ils n'ont pas été rajouté dans l'image, ce sont des pixels de l'image d'origine. On parle alors de **vrais positifs** lorsqu'il s'agit d'un pixel blanc qui était réellement falsifié, et de **faux positifs** lorsqu'il s'agit d'un pixel blanc qui fait parti de l'image originale. Un **vrai négatif** est un pixel noir qui est à l'origine un pixel non falsifié de l'image, tandis qu'un **faux négatif** est un pixel qu'on considère comme provenant de l'image de base mais qui appartient finalement à la zone falsifiée.

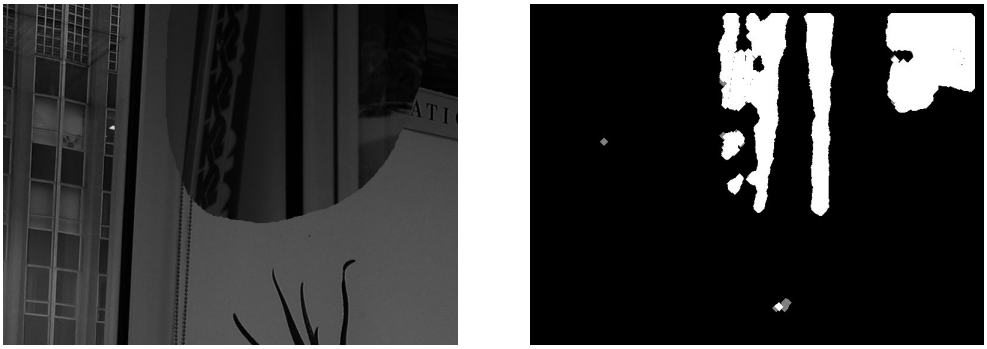
(a) (b)

FIGURE 3.3 – Résultat obtenu sur une image avec une zone ayant un bruit additif aléatoire augmenté artificiellement



(a) (b)

FIGURE 3.4 – Résultat obtenu sur une image falsifiée par nos soin



(a) (b)

FIGURE 3.5 – Résultat obtenu sur une image de la base Columbia

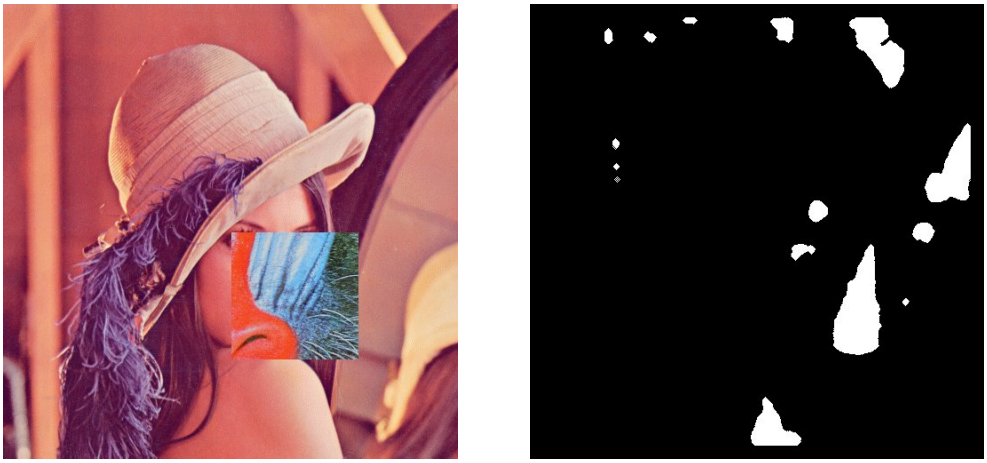
Dans notre premier exemple, on peut voir que le nombre de faux positifs est quasiment nul. Cependant, dans le deuxième exemple les résultats sont plus mitigés, notre traitement ne détecte pas clairement la zone de l'image falsifiée. Pour l'image de la base de Columbia, les résultats sont plus encourageant. En effet, le nombre de vrais positifs est élevé, cependant le nombre de faux négatifs reste élevé. Notre méthode ne détecte pas l'ensemble de la zone falsifiée. Si on fait le choix de prendre un seuil plus grand, le résultat sera trop dépendant de la texture. Nous avons donc revu notre méthode pour essayer d'avoir un traitement du bruit moins dépendant de la texture (cf. 3.3.4).

3.3.3 Traitement sur les images couleur

L'étude des histogrammes de bruit

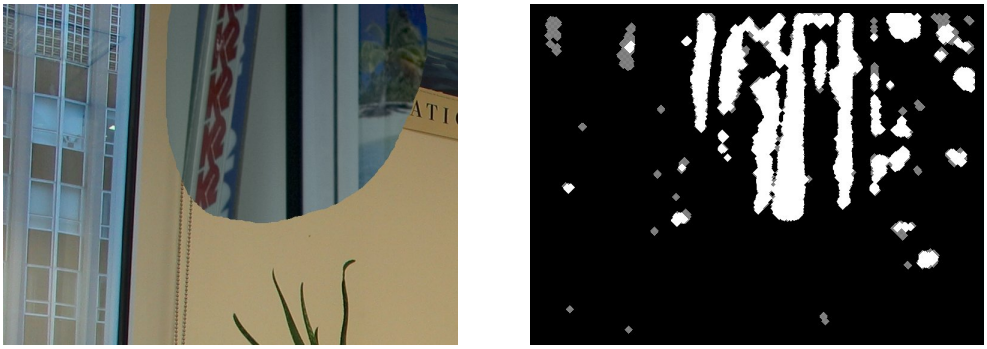
Pour le traitement d'images en couleurs, nous avons appliqué l'algorithme de comparaison d'histogrammes aux trois composantes de couleur de l'image. Nous avons donc comparé pour chaque pixel les densités de probabilité de bruit rouge, vert et bleu locales avec les densités de probabilité globales, et nous avons considéré que le pixel appartient à une zone modifiée si la somme de ces différences est supérieure à un certain seuil.

Résultats obtenus



(a) (b)

FIGURE 3.6 – Résultat obtenu sur une image couleur falsifiée



(a) (b)

FIGURE 3.7 – Résultat obtenu sur une image couleur de la base Columbia

Nos résultats sur les images couleurs sont à peu près équivalents à nos résultats sur les images en niveaux de gris. On peut voir qu'on obtient un meilleur résultat sur l'image provenant de la base Columbia. (Les zones visuellement grises sur l'image binaire de la figure 3.7 (b) sont en réalité des damiers de pixels blancs et noirs, c'est ce qui leur donne cet aspect gris.)

3.3.4 Amélioration de l'analyse du bruit

Suite à nos premiers résultats, nous nous sommes rendu compte que notre méthode de base pour caractériser le bruit d'une image ne permettait pas de différencier efficacement le bruit de la texture, comme le montre la carte de chaleur ci-dessous qui représente la différence entre le critère local et le critère global. En effet, plus la différence est grande, plus le pixel est clair. Bien que la zone falsifiée soit légèrement plus claire, on observe des zones blanches aux endroits fortement texturés.

(a)

FIGURE 3.8 – Carte de chaleur

Nous avons donc exploré d'autres pistes, notamment celle de la détection de texture, afin d'appliquer le filtre médian uniquement aux endroits peu texturés, et ainsi avoir un bruit moins influencé par la texture.
Cependant, cela rend plus difficile la détection de falsification sur une zone très texturée car les pixels appartenant au contour (pixels en noir ci dessous) sont considérés comme inexistants par le reste de l'algorithme pour ne pas contaminer le bruit avec la texture.

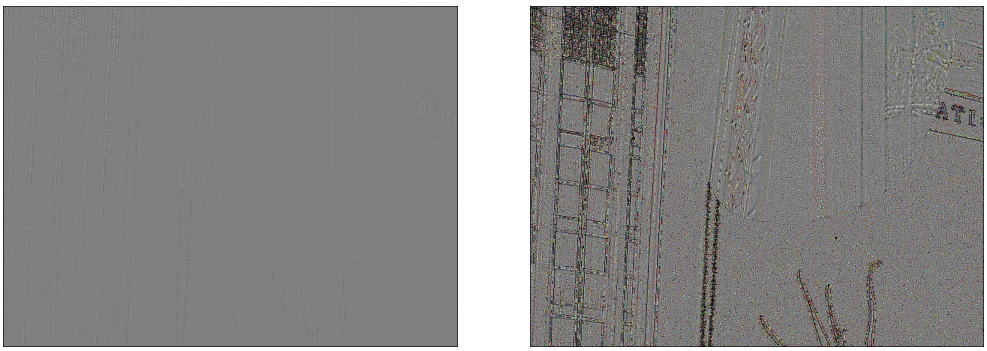
```
Data: image à tester
Result: image de bruit
read(imgIn) ;
// Calcul de l'écart type de l'image
ecartType = calcEcartType(imgIn) ;
// Initialisation du seuil à partir duquel on considère le pixel
   comme faisant partie du contour
seuilTexture = ecartType / 4 ;
// Pour tous les pixels de l'image en entree
foreach pixel in imgIn do
    // Calcul de la différence entre la valeur du pixel et la
       valeur médiane du voisinage du pixel
    if abs(imgIn[pixel] - getMediane(pixel)) > seuilTexture then
        // Si la valeur est supérieure au seuil, on n'applique pas
           le filtre median
        imgMedian[pixel] = imgIn[pixel] ;
        // Le pixel est mis à zéro dans l'image bruit
        imgBruit[pixel] = 0 ;
    else
        // Sinon, on applique le filtre median
        imgMedian[pixel] = getMediane(pixel) ;
        // Le pixel prend la valeur du bruit centré à 128 dans
           l'image de bruit
        imgBruit[pixel] = 128 + (imgIn[pixel] - imgMediane[pixel] ;
    end
end
return imgBruit ;
```

Algorithm 4: Amélioration de l'extraction de bruit

Comparaison des résultats

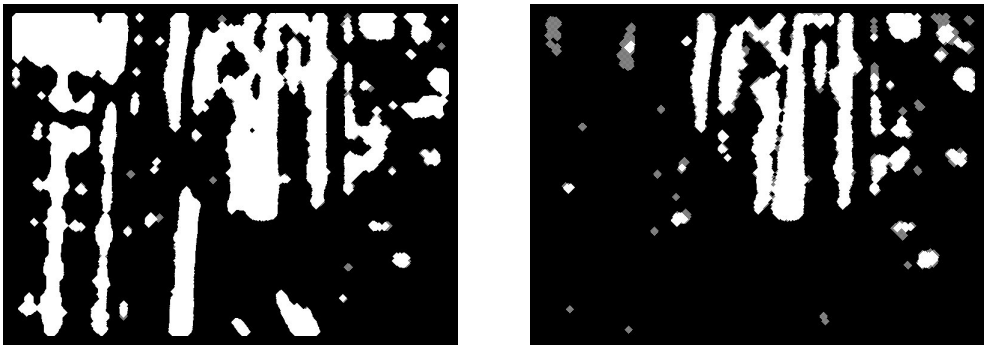
(a)

FIGURE 3.9 – Image originale



(a) (b)

FIGURE 3.10 – Images obtenues par extraction du bruit sur la figure 3.8 sans l'analyse de la texture (a) et avec l'analyse de la texture (b)



(a) (b)

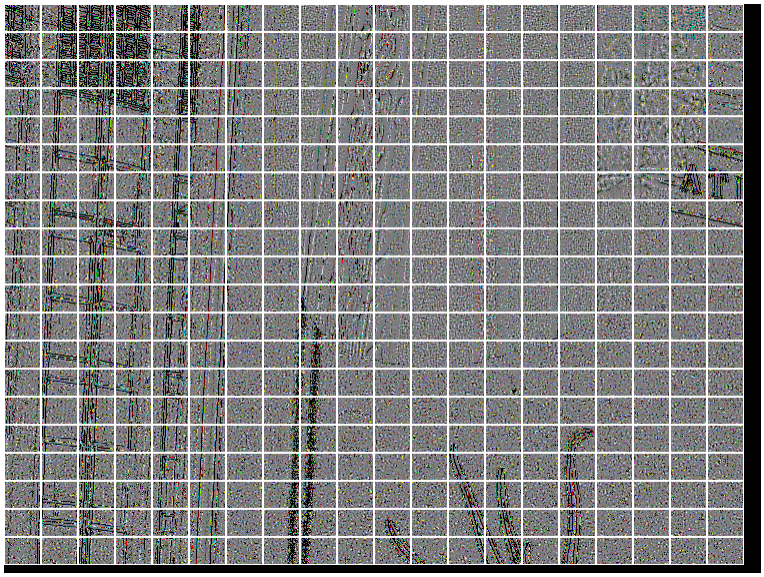
FIGURE 3.11 – Comparaison des résultats obtenus

On peut voir que notre amélioration du traitement du bruit a permis d'éliminer beaucoup de faux positifs et d'obtenir un résultat bien plus cohérent grâce au traitement sur les zones texturées.

3.3.5 Découpe d'une image en patchs

Nous avons également développé une méthode utilisant la comparaison d'histogrammes, mais cette fois ci, au lieu de traiter le voisinage de chaque pixel, nous avons divisé l'image en patchs, et comparé les histogrammes de chaque patch avec les histogrammes globaux. Ainsi, on peut déterminer quels patchs ont des valeurs éloignées des valeurs globales.

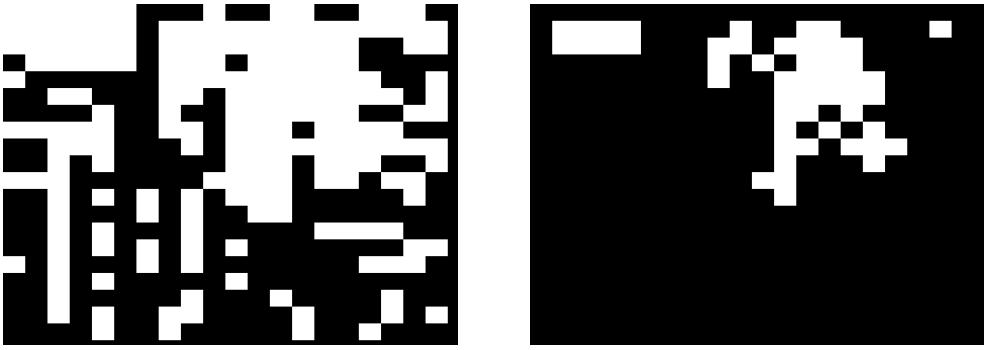
D'abord nous avons repris l'image de la figure 3.8 et l'avons divisé en une image de 20 patchs par 20 comme sur la figure çi-dessous où les bords des patchs ont été rendus visible pour constater l'état des patchs. On constatera une bordure noire sur les côtés à droite et en bas car la largeur et la hauteur de 568 x 757 ne sont pas des mutiples de 20.



(a)

FIGURE 3.12 – Quadrillage obtenu par division de l'image en 400 blocs (20 x 20) rendu visible par mise à blanc des bordures des patchs de l'image de bruit de la figure 3.9

L'image binaire en sortie ne nous convenait pas, car il y avait beaucoup de trous (voir la figure 3.12, image a). Nous avons alors créé une nouvelle image binaire miniature (de taille 'nombre de patchs en largeur' x 'nombre de patchs en hauteur') qui pour chaque pixel prenait la valeur d'un patch de l'image binaire précédente. Cela donne donc la même image en plus petit. Et nous y avons appliqué une simple érosion pour boucher les trous blancs. Enfin nous avons réagrandit l'image miniature en attribuant à chaque patch de l'image binaire originale la valeur du pixel de l'image miniature associée(voir la figure 3.12, image b).



(a) (b)

FIGURE 3.13 – Image du bruit patchée et binarisée de sortie (a) et cette même image miniaturisée érodée et réagrandie (b)

3.3.6 Utilisation d'une courbe ROC

Toutes nos méthodes de comparaison d'histogrammes nécessitant un seuil différent pour chaque image afin de fonctionner de manière optimale, nous avons utilisé des courbes ROC [2] pour déterminer efficacement le meilleur seuil pour un ensemble d'images dans le cas où une vérité de terrain est disponible.

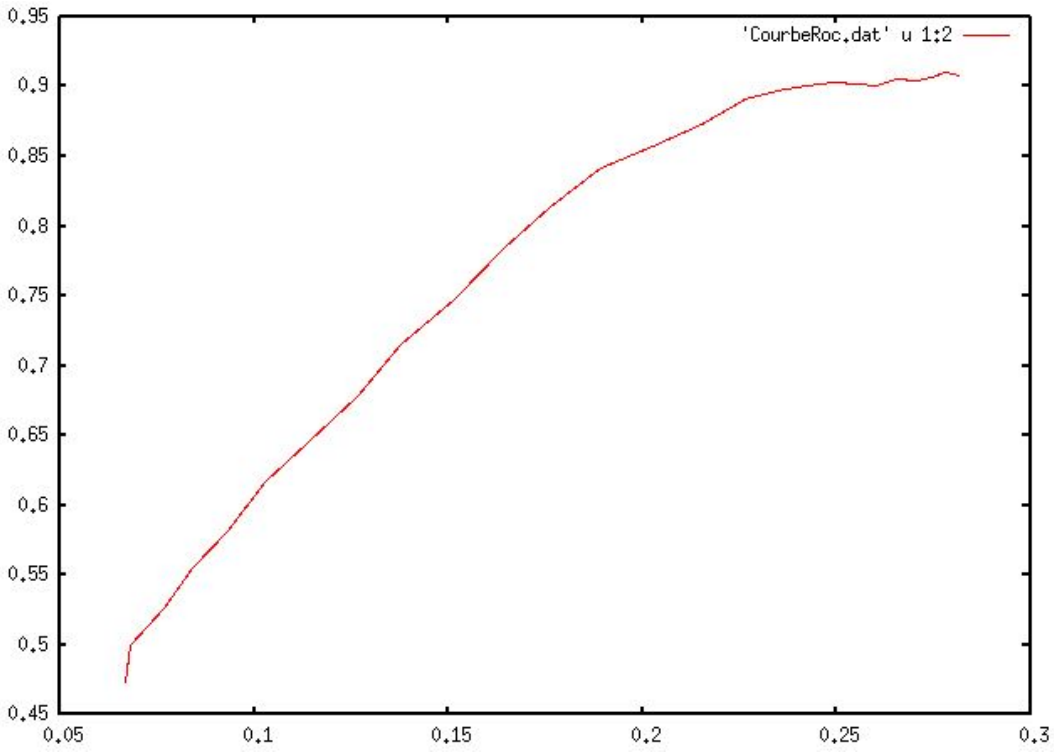
Pour chaque image à tester, nous calculons plusieurs images de résultats avec différents seuils afin de construire la courbe en comparant ces résultats avec la vérité de terrain. Une fois la courbe créée nous gardons uniquement l'image correspondant au point de la courbe qui minimise la distance avec le point de coordonnées (0 ;1), qui correspond à un cas avec aucun faux positif et aucun faux négatif (par conséquent parfaitement exact).

2. La courbe ROC de l'anglais Receiver Operating Characterisitic représente une courbe qui donne le taux de vrais positifs en fonction du taux de faux positifs. Elle permet donc de mesurer l'efficacité d'un classificateur binaire.

Résultats courbe ROC

(a) (b)

FIGURE 3.14 – Image utilisée et sa vérité de terrain



(a)

FIGURE 3.15 – Courbe ROC obtenue

Chapitre 4

Rapport technique

4.1 Langage de développement

Pour l'ensemble du projet nous avons travaillé sur le langage **C++**. C'est un langage de programmation compilé parmi les plus utilisés dans les applications où la performance est critique. En effet, pour le traitement d'images il est important d'optimiser les performances afin de limiter le temps le traitement (notamment pour la lecture et l'écriture des pixels d'une image).
Au cours de l'enseignement *Traitement des Images* du Master IMAGINA, nous avons uniquement développé sur du C++. Il était donc pour nous logique de continuer à travailler sur ce langage. De plus, nous avons exploité des méthodes provenant de la base de M. Puech qui nous a été fournie lors de cet enseignement. Ces méthodes nous ont permis par exemple de lire et d'écrire sur des images aux formats PGM ou PPM.

```
void ecrire_image_ppm(char  nom_image[], OCTET *pt_image, int nb_lignes, int nb_colonnes) {
   FILE *f_image;
   int taille_image = 3*nb_colonnes * nb_lignes;

   if( (f_image = fopen(nom_image, "wb")) == NULL) {
         printf("\nPas d'acces en ecriture sur l'image %s \n", nom_image);
         exit(EXIT_FAILURE);
    } else {
         fprintf(f_image,"P6\r") ;
         fprintf(f_image,"%d %d\r255\r", nb_colonnes, nb_lignes) ;

         if( (fwrite((OCTET*)pt_image, sizeof(OCTET), taille_image, f_image))
             != (size_t)(taille_image)) {
               printf("\nErreur d'ecriture de l'image %s \n", nom_image);
               exit(EXIT_FAILURE);
         }
         fclose(f_image);
    }
}
```

Algorithm 5: Méthode pour écrire dans une image PPM

4.2 Environnement de travail

Nous avons travaillé dans un environnement **Linux** (Ubuntu) pour compiler nos fichiers C++ et pour analyser nos images PPM et PGM plus simplement. Nous avons utilisé principalement l'éditeur de texte **Sublime Text** pour sa simplicité. Ce rapport a été conçu sur le site **ShareLatex** qui permet de créer des documents en Latex sur le cloud, d'y donner l'accès à plusieurs participants et ainsi de pouvoir modifier et visualiser le rapport en groupe et en temps réel. Le **LateX** est un langage de description de document permettant d'uniformiser notre document en donnant des instructions typographiques et de facilement introduire des formules mathématiques ou du code.

4.3 Formats d'images

Par habitude avec notre travail dans l'enseignement *Traitement des Images* nous avons travaillé sur des images **PGM et PPM**.

4.3.1 Le format PGM

Ce format est utilisé pour des images en niveaux de gris. Chaque pixel d'une image PGM dispose d'un niveau de gris qui est codé par une valeur entre 0 et 255 proportionnellement à son intensité (un pixel noir est codé par la valeur 0 et un pixel blanc par 255).

4.3.2 Le format PPM

Ce format est utilisé pour les images couleurs. En effet, chaque pixel d'une image PPM est codé par trois valeurs : Le rouge, le vert et le bleu. Chaque valeur est elle aussi codée de 0 à 255. Par exemple, un pixel vert est codé par *0 255 0*.

Chapitre 5

Rapport d'activité

5.1 Organisation du travail

5.1.1 Organisation du groupe

Nous avons fait le choix de travailler ensemble sur toutes les parties du projet (le développement, la publication du rapport et les tests).
Cependant par nature nous nous sommes plus ou moins focalisés sur une partie en particulier :

— **Nicolas Cellier** était responsable du développement technique,
— **Simon Hamery** responsable du versionnage et de l'intégration
— et **Tristan Cabantous** responsable de la publication.

Le responsable de chaque catégorie doit veiller au bon déroulement de son domaine et vérifier le travail de chacun.
Nous ne souhaitions pas hiérarchiser le groupe pour que chaque personne s'investisse au même niveau dans chacune des disciplines du projet. Lorsque nous définissions la liste des exigences à chaque phase, nous fixions des tâches respectives à chaque membre du groupe jusqu'à la fin de la phase.

5.1.2 Organisation du temps de travail

Nous nous sommes réunis toutes les semaines pour travailler ensemble sur le projet sur des séances d'environ 4 heures. De plus, nous avons travaillé chez nous à hauteur de une ou deux heures par semaine en fonction de la quantité de travail. Sur 16 semaines, nous comptabilisons environ 88 heures par personne, soit un total de **264 heures**.

5.1.3 Planification des tâches

Dans un premier temps, nous avons réalisé un planning prévisionnel dans le cadre de l'enseignement **Conduite de projet** de Monsieur Bourreau. Ce planning nous a permis d'avoir un aperçu des différentes phases de notre projet et de commencer à répartir les tâches des membres du groupe. Cependant, ce planning n'était pas très représentatif du travail et pas assez détaillé pour le suivre à la lettre. De plus, l'absence d'un membre du groupe nous a obligé à revoir la grande majorité de ce planning. Malgré des différences avec le planning prévisionnel, nous avons réussi à réaliser des itérations fonctionnelles en améliorant notre prototype, et à utiliser le modèle de développement en cascade efficacement.
Pour évaluer notre travail et définir de nouvelles pistes de développement, nous organisions une réunion avec l'équipe d'encadrement à chaque phase du projet (environ toutes les deux semaines). Nous définissions ensuite des tâches en fonction du compte rendu de la réunion et nous les ajoutions dans notre outil de gestion de tâches (**cf 5.2.2**). A la fin de notre projet nous avons obtenu le diagramme de Gantt [1] ci-dessous (**Figure 5.1**).

1. Le diagramme de Gantt permet de visualiser dans le temps les diverses tâches composant un projet.

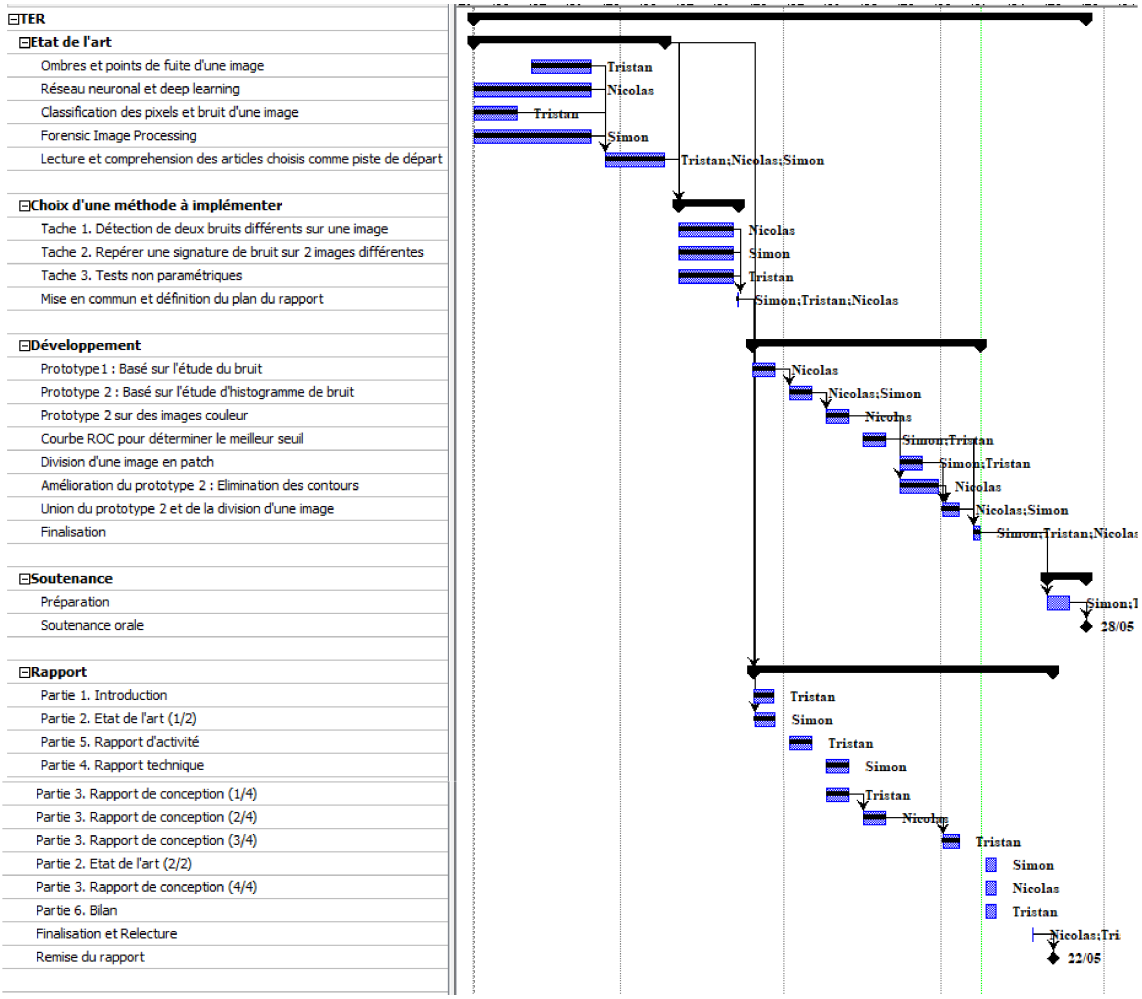


FIGURE 5.1 – Diagramme de Gantt

5.1.4 Le modèle de développement

Étant donné la nature de notre projet, nous avons développé notre application suivant le modèle en cascade, sur plusieurs phases.
L'avantage de ce modèle est qu'il nous permet de facilement gérer les différentes phases de l'élaboration du projet et de suivre un développement simple :

— Définir la **liste des exigences**,
— **Conception** du prototype,
— **Implémentation**,
— **Vérification** : Phase de test,
— **Correction** des bugs.

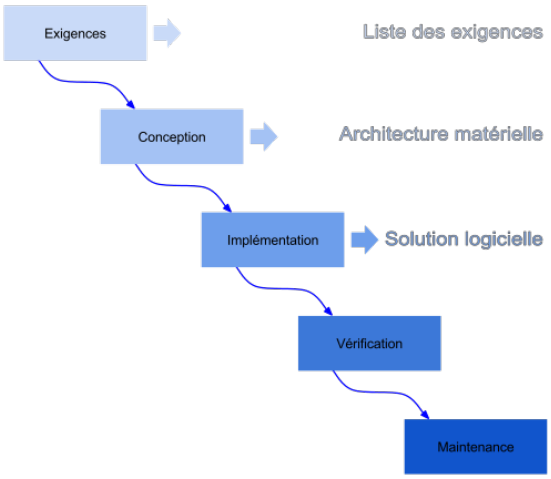


FIGURE 5.2 – Le modèle en cascade

A chacune des phases du projet, nous exploitions les phases précédentes en améliorant notre prototype. Chaque livrable est testé et chaque phase est publiée avant le début d'une prochaine phase de développement. De plus, le modèle en cascade permet de facilement se fixer des dates d'échéance et de limiter le risque. En effet, en cas d'erreur on remonte à la phase précédente. Un inconvénient à utiliser ce modèle est que les erreurs peuvent être détectées tardivement, ce qui implique une perte de temps importante et nous a obligé parfois à revoir nos ambitions à la baisse.

5.2 Méthodes et outils utilisés

5.2.1 Le versionnage

Il était important pour notre projet de versionner nos prototypes à chaque phase. Il nous permet notamment de tous travailler sur la même base, mais aussi de repartir sur un protype précédent en cas d'erreur. Pour versionner notre projet nous avons décider d'utiliser **Github**. Nous avons choisi cette solution car nous étions habitués à l'utiliser, et cela nous a permis de facilement mettre en place notre versionnage.

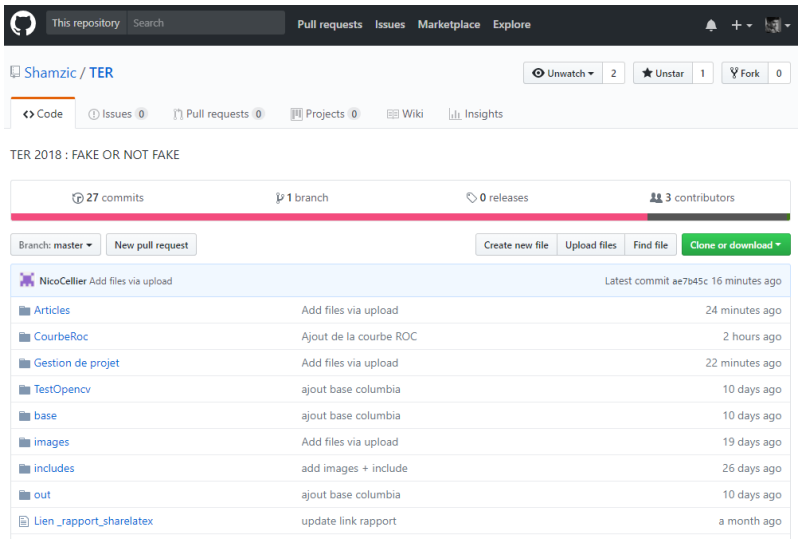


FIGURE 5.3 – Le projet sur Github

5.2.2 Outil pour la planification des tâches

Pour tenir au courant l'équipe de nos avancées, et pour gérer notre projet dans la globalité nous avons utilisé **OpenProj**. OpenProj est un logiciel gratuit et libre de gestion de projet. Il permet de planifier des tâches avec association de dépendances et de ressources, il nous a aidé à suivre l'avancée de chaque phase et du rapport et nous a permis d'obtenir le diagramme de Gantt de notre projet.

5.2.3 Méthodes de communication

Pour communiquer entre membres du groupe nous avons principalement utilisé l'outil **Messenger** de Facebook. En effet, il nous a permis de communiquer facilement et intuitivement durant l'ensemble du projet. De plus, l'outil nous a aussi permis de créer rapidement des sondages pour s'organiser dans notre travail.

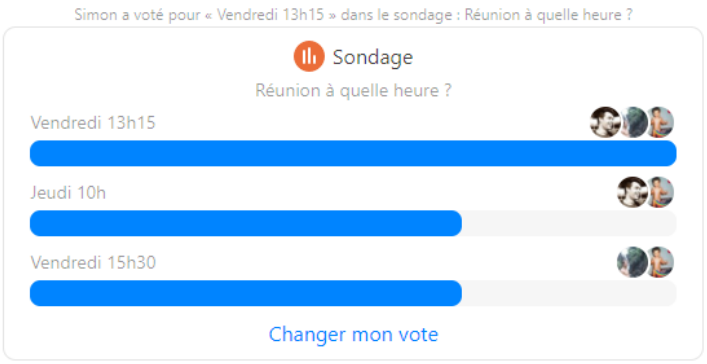


FIGURE 5.4 – Exemple de sondage sur Messenger

Chapitre 6

Bilan

6.1 Bilan technique

6.1.1 Résultats obtenus

Nous avons testé nos résultats sur 10 images d'une base de Columbia. Sur chaque image, nous avons utilisé la vérité de terrain pour calculer le nombre de vrais positifs par rapport au nombre de positifs, et le nombre de vrais négatifs par rapport au nombre de négatifs. Si l'on fait une moyenne de ces résultats, on obtient une moyenne de **67%** de vrais positifs et **75.7%** de vrais négatifs.

Numéro d'image	Vrais positifs	Vrais négatifs
0	89.6%	75.8%
1	75.8%	75.9%
2	83.4%	75.1%
3	32.3%	77.9%
4	72.6%	79.6%
5	62.3%	78.7%
6	43.7%	54.3%
7	65.2%	67.1%
8	85.9%	80.9%
9	59.2%	92.2%
Moyenne	67.0%	75.7%

FIGURE 6.1 – Pourcentage de vrais positifs et de vrais négatifs sur 10 images de la base Columbia

On remarque que les résultats sont globalement positifs, avec une moyenne de 67% de vrais positifs. Cependant, cette valeur est fortement influencée par les deux images sur lesquelles la détection n'a pas été concluante (images 3 et 6). Sans ces deux valeurs, la moyenne de vrais positifs augmente à **74.25%**.

6.1.2 Limites perceptibles

A la suite des premiers résultats, nous nous sommes rendu compte que nos travaux sur le bruit dépendaient énormément de la texture de l'image. Nous avons réussi à améliorer notre méthode à l'aide de la détection des textures mais cette dépendance persiste. Le filtre médian pour séparer le bruit de l'image ne nous permet pas de traiter efficacement cette dépendance. Notre méthode de détection de falsification reste efficace dans certains cas mais cette dépendance fausse nos résultats sur des images très texturées.

De plus, nos prototypes ne traitent que des images dans le format PGM et PPM.

Certaines compressions comme une compression JPEG peuvent déformer nos résultats. En effet, une compression JPEG supprime le bruit par suppression des hautes fréquences et ajoute des artefacts sur l'image comme les effets de blocs.

6.2 Bilan humain

6.2.1 Autocritique

Nous sommes globalement satisfaits de notre gestion de travail. Nous avons passé beaucoup de temps sur l'état de l'art car cette partie était nécessaire pour la bonne compréhension de notre sujet. De plus, nous avons dû revoir nos ambitions à la baisse dû à l'absence d'un membre du groupe. Enfin, à cause d'une mauvaise analyse du bruit nous avons eu dans un premier temps de mauvais résultats sur nos programmes de détection de falsification. Cependant, nous avons su réagir et améliorer notre traitement pour obtenir de meilleures détections par la suite.

6.2.2 Enseignements tirés

Les difficultés que nous avons rencontré tout au long du développement du projet vont nous permettre de mieux évaluer nos prochains projets en terme de temps et de capacité technique. Travailler sur un projet de recherche était nouveau pour nous tous.
Nous avons pu comprendre **comment s'organiser**, et être **autonomes** dans un processus de recherche. De plus, ce projet nous a aussi permis de nous confronter à nos capacités techniques, et à notre capacité à travailler à plusieurs. Même si nous avons perdu du temps de développement, avoir passé beaucoup de temps à analyser l'état de l'art nous a permis de mieux envisager nos ambitions et nos prototypes de détection. Ce projet a été un challenge pour nous trois et nous permet **de mieux envisager notre avenir professionnel**.

Bibliographie

[1] Thibault Julliand (2017), *Automatic noise-based detection of splicing in digital images*, Université Paris-Est Marne-la-Valée.

[2] Thibault Julliand, Vincent Nozick, Isao Echizen, Hugues Talbot (2017), *Using The Noise Density Down Projection To Expose Splicing In JPEG Images.*

[3] Tian-Tsong Ng and Shih-Fu Chang (2004), *A model for image splicing*, Department of Electrical Engineering, Columbia University, New York

[4] Gilbert Peterson (2005), *Forensic Analysis of Digital Image Tampering*, Part of the IFIP — The International Federation for Information Processing book series (IFIPAICT, volume 194)

[5] *Hippolyte Bayard, biographie wikipédia*, `https://fr.wikipedia.org/wiki/Hippolyte_Bayard`

[6] Minati Mishra and Munesh Chandra Adhikary (2013) *Digital Image Tamper Detection Techniques - A Comprehensive Study* - International Journal of Computer Science and Business Informatics

[7] Alin C Popescu and Hany Farid (2004) *Exposing Digital Forgeries by Detecting Duplicated Image Regions*, Department of Computer Science, Dartmouth College

[8] *Méthode Otsu*, `https://fr.wikipedia.org/wiki/M%C3%A9thode_d%27Otsu`

[9] Zhigang Fan and Ricardo L. de Queiroz (2003) *Identification of Bitmap Compression History : JPEG Detection and Quantizer Estimation(VOL. 12, NO. 2S)*, Senior Member, IEEE