

# Rapport de Projet : Traitement d'image Android

Omar Belkebir

22 avril 2020



FIGURE 1 –

## Table des matières

<b>1 Présentation du Projet</b>	<b>3</b>
1.1 Présentation du sujet et cahier de charge . . . . .	3
1.2 Cadre de Travail . . . . .	3
1.3 Principaux objectifs du sujet . . . . .	3
<b>2 Conception des solutions algorihmiques</b>	<b>4</b>
2.1 Conception fonctionnalités de bases . . . . .	4
2.1.1 Fonctionnalité d'annulation de modifications . . . . .	4
2.1.2 Fonctionnalités de chargement d'enregistrement . . . . .	4
2.1.3 Fonctionnalité de Zoom/Scroll . . . . .	5
2.2 Conception Algorithme traitement d'image . . . . .	6
2.2.1 Manipulation de teintes et de couleurs . . . . .	6
2.2.2 Transformation de l'histogramme . . . . .	8
2.2.3 Algorithmes de convolution . . . . .	10
2.2.4 Fonctionnalités avancées . . . . .	13
<b>3 Performance et complexité</b>	<b>14</b>
3.0.1 Manipulation de teintes et de couleurs . . . . .	14
3.0.2 Transformation de l'histogramme . . . . .	15
3.0.3 Algorithmes de convolution . . . . .	15
<b>4 Interface utilisateur</b>	<b>16</b>
<b>5 Points à approfondir</b>	<b>17</b>
5.1 Utilisation de RenderScript . . . . .	17
5.2 Amélioration de certains algorithmes . . . . .	17
5.2.1 Algorithmes de contraste . . . . .	17
5.2.2 Algorithmes de changement de teintes ou couleur . . . . .	17
5.3 Ajout de fonctionnalités . . . . .	17
<b>6 Conclusion</b>	<b>17</b>

# 1 Présentation du Projet

## 1.1 Présentation du sujet et cahier de charge

Le but de ce projet est l'implémentation d'une application de traitement d'image sous Android. Il s'agit de mettre en place plusieurs méthodes de traitement d'image (réglage de luminosité, réglage de contraste, modification de teintes, plusieurs formes de convolution...). Notre application devra être en mesure de charger une photo pour ensuite l'enregistrer après modification, le tout avec les meilleures performances possibles et une ergonomie adéquate de l'interface utilisateur.

## 1.2 Cadre de Travail

Les langages de programmation utilisés pour ce projet sont le Java et RenderScript. Le language Java est le language de base du projet. En effet 3 classes seront implémentées pour assurer le bon fonctionnement de l'application et une structuration optimisée du code facilitant l'ajout et le retrait de fonctionnalités :

- la classe MainActivity : La classe qui est essentiellement gérée l'interface utilisateur.
- la classe Processing : On y trouve toutes les fonctions de traitement d'image implémentées
- la classe Convolution : qui naturellement hérite de Processing est une classe où se trouvent les différents types de convolution implémentées

Quant au langage Renderscript, son rôle est technique : Il s'agit de tirer facilement partie des différents coeurs du CPU ou du GPU des smartphones sous Android, et donc réduire considérablement les temps d'exécution des algorithmes.

Dans ce rapport, les fichiers ou répertoire seront écrits en *italique* et les classe ,fonctions ou variables seront écrits en **gras**

## 1.3 Principaux objectifs du sujet

La création de l'application se subdivise en trois axes :

1. La création des fonctions de bases.
2. L'implémentation d'algorithmes de traitement d'image.
3. La mise en place d'interface utilisateur adaptée.

Le premier objectif du projet est l'implémentation de fonctionnalités de bases, c'est à dire toute les fonctions qui donneront accès au besoins primaires et donc essentielles à l'utilisation de l'application.

Les fonctions en question sont : Le chargement de l'image, la sauvegarde, l'annulation de modification et l'instauration de fonctionnalités scroll et zoom de l'image à traiter.

Le second est la conception d'algorithme de traitement d'image vu en cours. Il s'agit de proposer des algorithmes performants qui assurent un fonctionnement sûr et rapide des fonctions de traitement d'image.

Et le dernier objectif est d'assurer une interface utilisateur simple et pratique où toutes les précédentes fonctionnalités seront accessibles.

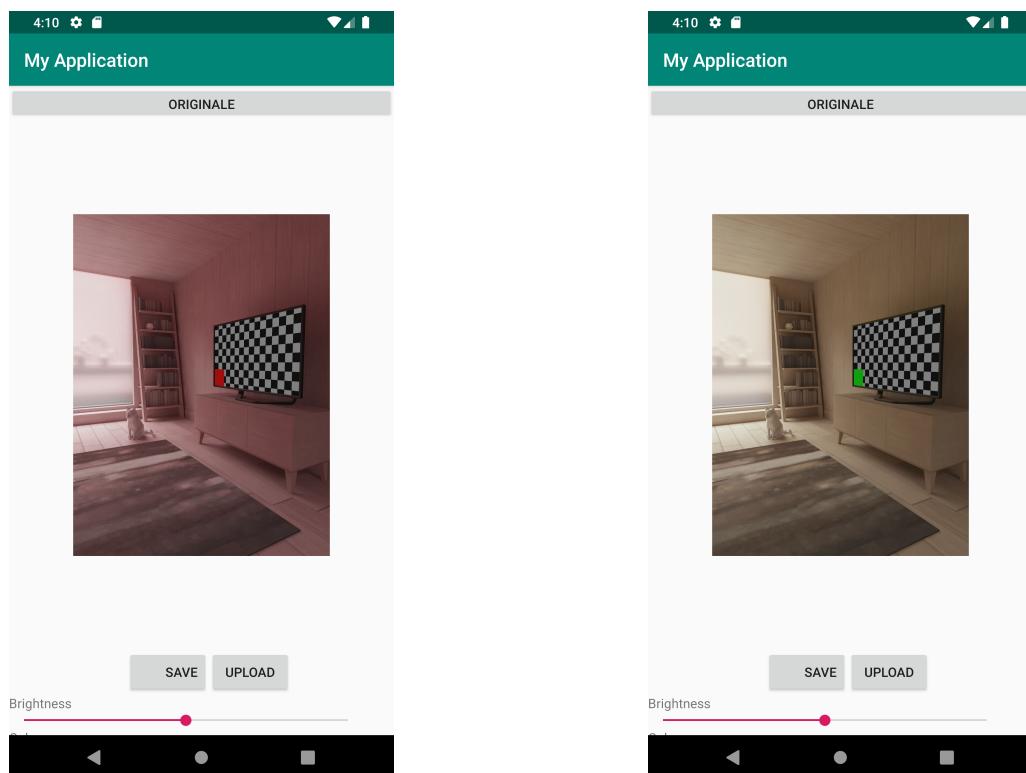
## 2 Conception des solutions algorithmiques

Afin de représenter convenablement les images à traiter dans les algorithmes de bases et de traitement d'image, on utilise la classe `Bitmap`, cette classe donne accès à une modélisation de l'image pratique. En effet elle permet de manipuler des images définies par des données de pixels.

### 2.1 Conception fonctionnalités de bases

#### 2.1.1 Fonctionnalité d'annulation de modifications

Le premier point traité du projet est la fonctionnalité `original` ou `back`, cette fonctionnalité se traduit par un bouton qui permettrait à l'utilisateur depuis son interface d'annuler toutes les modifications qu'a subit une image depuis son chargement.



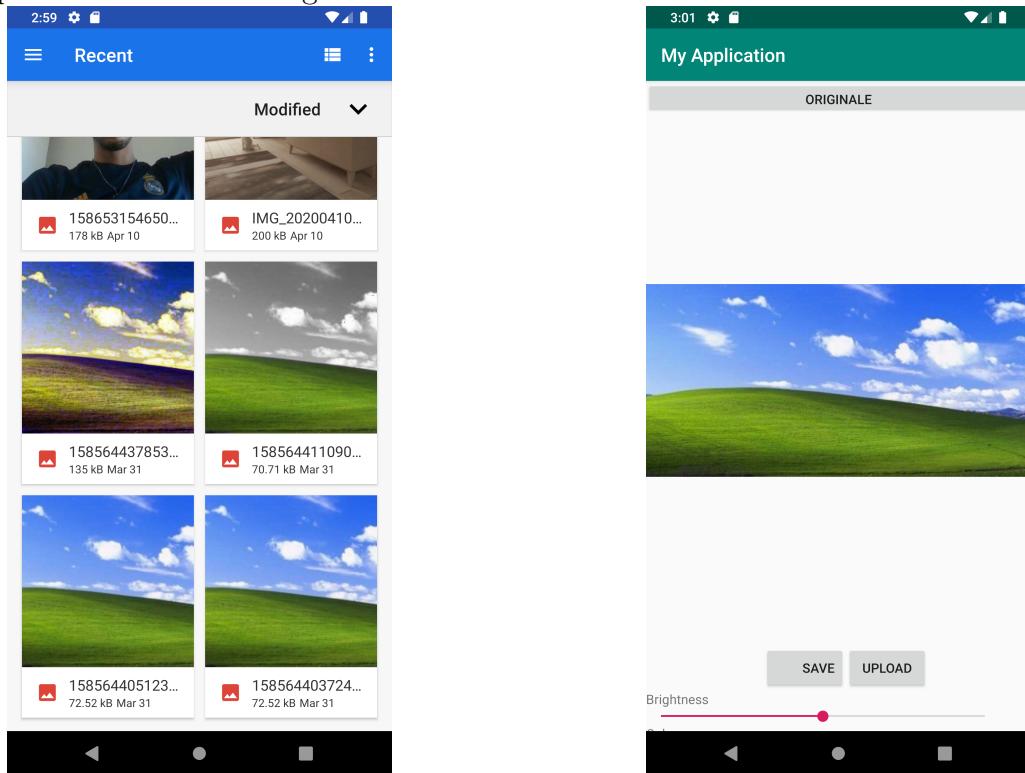
Pour concevoir cette fonctionnalité il a fallut penser à porter en permanence dans la mémoire l'aspect de l'image initiale. Pour cela nous avons pensé ajouter une nouvel attribut à la classe `MainActivity`, `pixels`. Cette variable de type `int []` est réinitialisée chaque fois qu'une nouvelle image est chargée, elle correspond à la valeur des pixels de l'image avant modification. Chaque fois que la fonction `original` est appelée, les valeurs des pixels de l'image actuelle sont remplacées par les valeurs de `pixels`.

#### 2.1.2 Fonctionnalités de chargement d'enregistrement

Avant de penser au fonctionnalités de traitement d'image, il est essentiel de concevoir des méthodes permettant l'accès à la galerie photo de l'appareil et cela dans les deux sens. En effet l'application doit non seulement être en mesure d'accéder à la galerie pour ouvrir une image, mais aussi de l'enregistrer après modifications et traitement.

Pour la fonctionnalité `upload`, elle s'occupe de charger une image directement de la galerie,

pour cela il est essentiel d'envoyer une requête à l'utilisateur pour pouvoir ouvrir la galerie et ainsi lui proposer le choix de l'image à traiter.

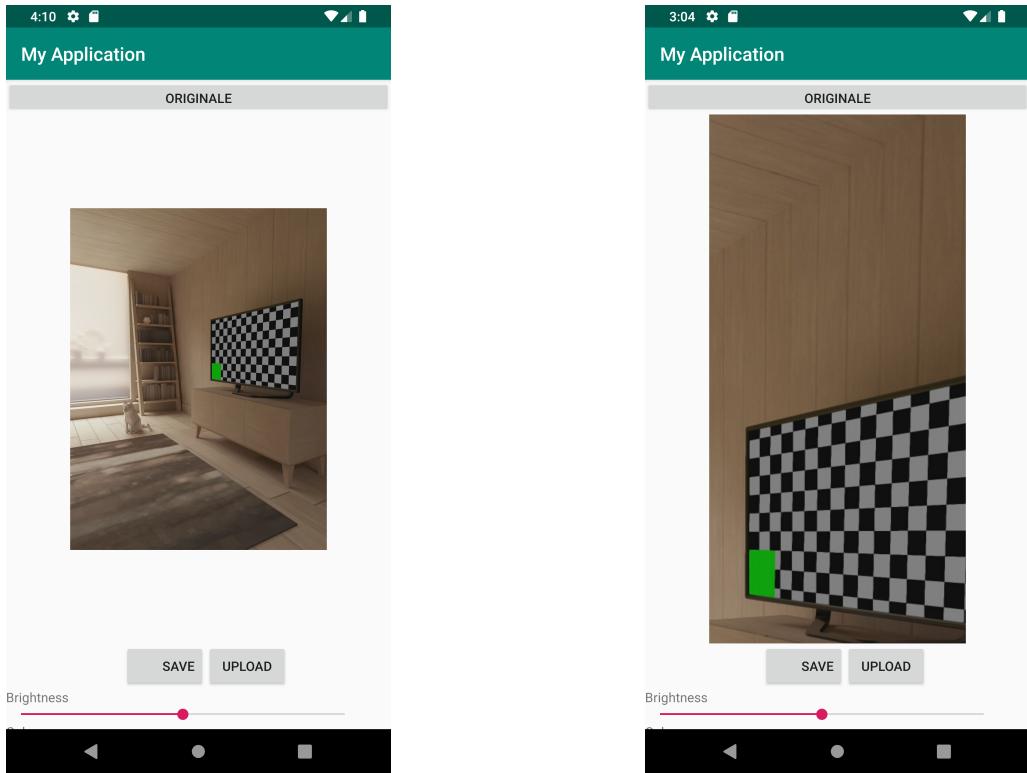


Concernant la fonctionnalité **save**, elle exige aussi l'acceptation de l'utilisateur de la requête d'écriture dans la galerie afin d'y ajouter l'image après modification.

### 2.1.3 Fonctionnalité de Zoom/Scroll

Afin de traiter des images de différentes tailles il est important d'implémenter des méthodes permettant de zoomer et de scroller.

Pour cela nous avons utilisé une bibliothèque en ligne créée par *chrisbanes*. Cette bibliothèque implémente un nouveau type d'image et une classe Java **PhotoView**. Très similaire à **ImageView**, elle donne accès aux mêmes manipulations en addition des fonctionnalités Zoom et Scroll.



## 2.2 Conception Algorithme traitement d'image

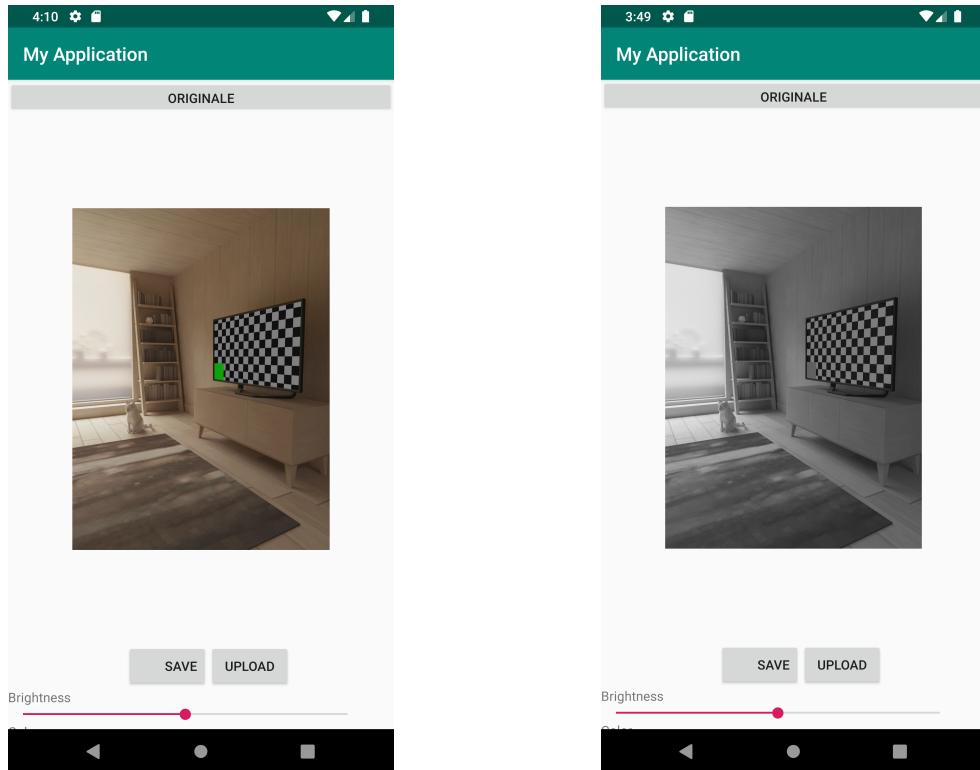
### 2.2.1 Manipulation de teintes et de couleurs

L'une des premières manipulation qui vient à l'esprit est le changement de teinte et couleur. L'application développée doit effectivement être en mesure d'assurer cette manipulation et cela en trois fonctionnalités principales :

- Fonctionnalité `toGray` :

On ne peut concevoir une application de traitement d'image sans implémenter une fonction qui permet de griser l'image. Pour cela on manipulera les composantes RGB des pixels de l'image. En effet chaque pixel contient les composantes Red (rouge), Green (vert) et Blue (bleu).

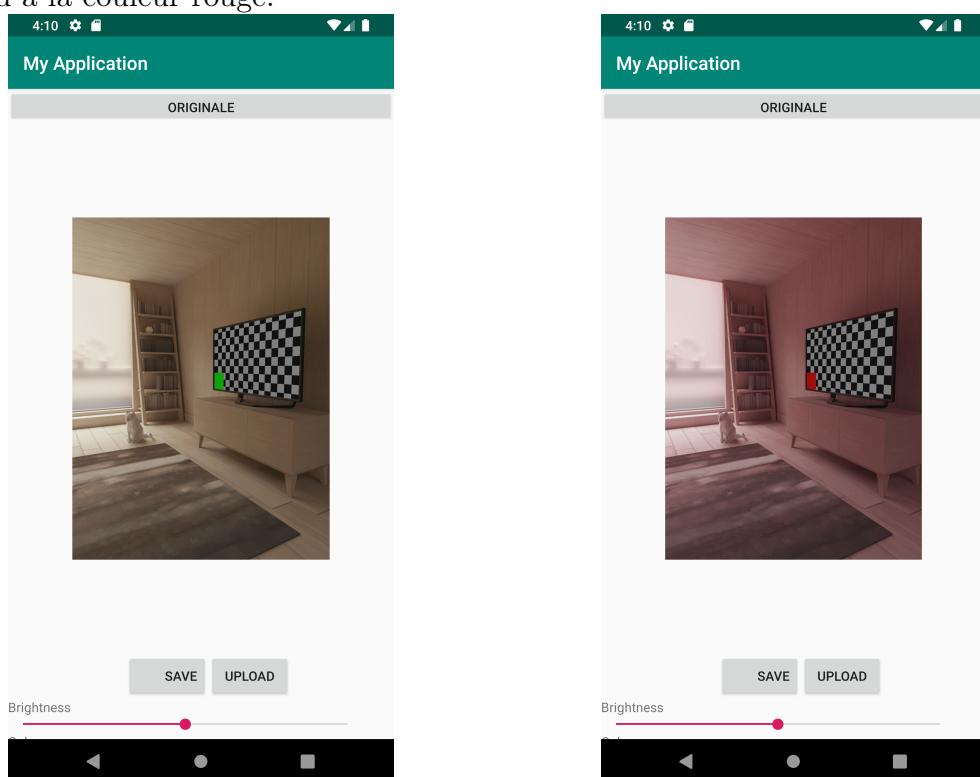
Pour avoir l'image représentée en niveaux de gris il suffit que ces trois composantes aient la même valeur pour chaque pixel. Pour cela on utilise la moyenne pondérée pour chaque pixel  $0.3R + 0.59G + 0.11B$ .



— Fonctionnalité **colorize** :

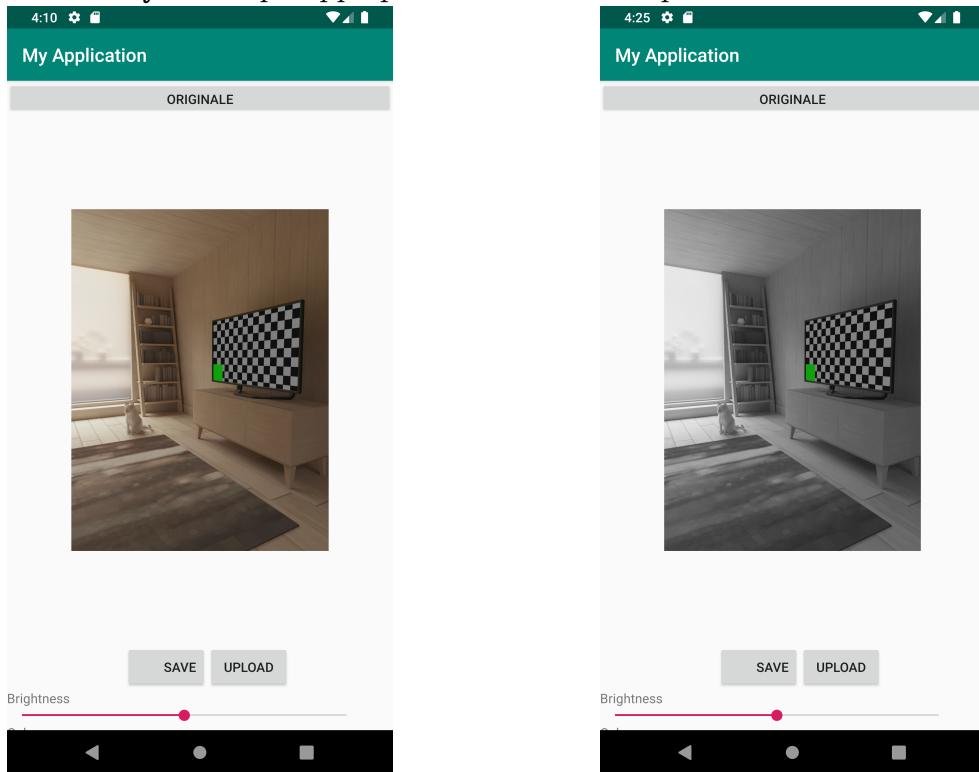
Cette fonctionnalité est très familière dans les applications de traitement d'image connues. Il s'agit de généraliser une teinte sur toute l'image.

Dans ce cas on utilise les composantes HSV de l'image, et on propose à l'utilisateur une seekbar afin de fixer la teinte voulue, la teinte ci-dessous est équivalente à une valeur de  $H=0$  ou à la couleur rouge.



— Fonctionnalité **onlyColor** :

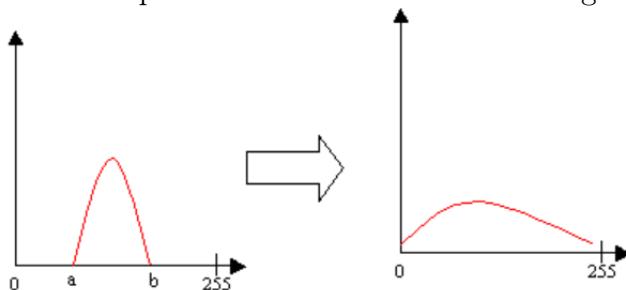
Cette fonctionnalité est aussi fréquente dans la plus part des applications et site web de traitement d'image. `onlyColor` permet comme son nom l'indique de griser l'intégralité d'une image sauf les éléments d'une couleur précise. Pour implémenter l'algorithme en question, il faut faire une analyse de l'image en entier et détecter les pixels qui ne seront pas grisés, pour cela on a recourt aux composantes HSV des pixels pour analyser la teinte H et vérifier qu'elle n'est pas éloignée de la teinte demandée par l'utilisateur. ci-dessous le résultat de `onlyGreen` qui applique la fonctionnalité pour la couleur verte :



### 2.2.2 Transformation de l'histogramme

La transformation de l'histogramme est un élément intéressant dans le cadre de traitement de l'image, il permet de modifier le contraste afin que des formes paraissent plus visibles. Pour cela plusieurs méthodes usuelles ont été implémentées dans le cadre de ce projet :

- La fonctionnalité `toContrastDyn` : Cette fonctionnalité traduit l'extension linéaire de dynamique, elle permet d'améliorer le contraste en agissant sur la forme de l'histogramme. Cette transformation n'est autre que l'élargissement de l'histogramme sur ses valeurs possibles comme le montre l'image ci-dessous :

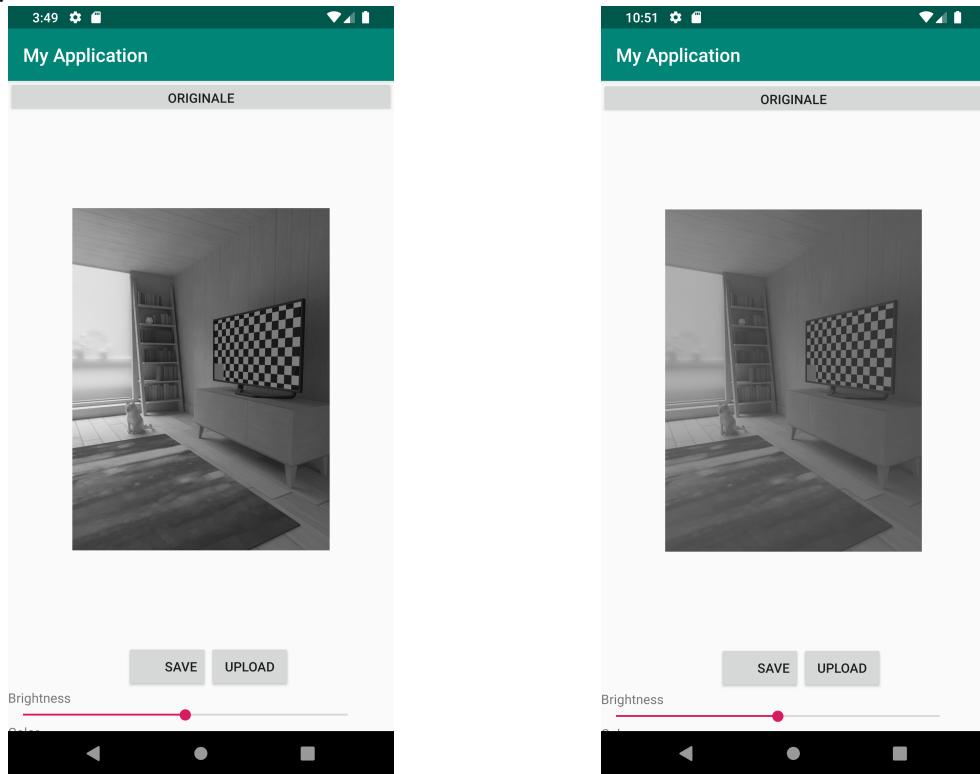


Pour effectuer cet élargissement il est essentiel de retrouver les valeurs `min` et `max` représentés par `a` et `b` dans l'image, la formule traduisant l'image contrastée est de la forme :

$$I'(x,y) = 255 * (I(x,y) - \min) / (\max - \min)$$

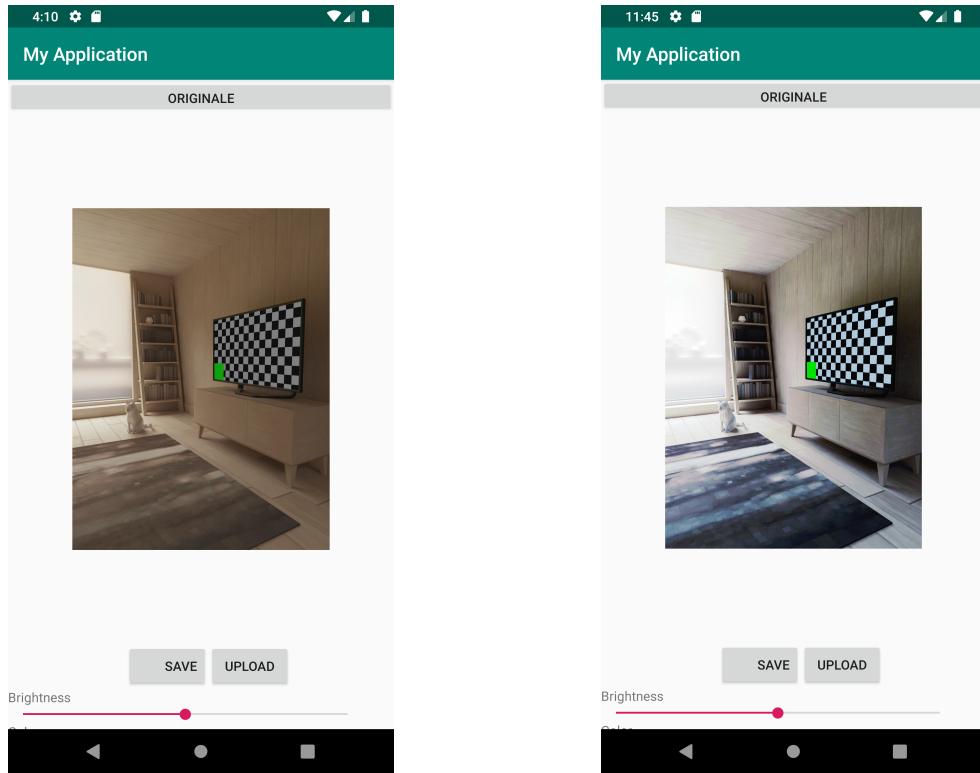
— La fonctionnalité **lessContrast** :

Contrairement à **toContrastDyn** la fonctionnalité **lessContrast** permet de resserrer l'histogramme, ce resserrement produit un effet de baisse de contrast. Pour produire ce resserrement il est impératif de trouver la valeur mediane de l'histogramme puis de rapprocher toutes les autres valeurs de l'histogramme de cette même valeur mediane avec un coefficient de rapprochement  $C < 1$ . ci-dessous un exemple de **lessContrast** pour  $C=1/4$  :



— Fonctionnalité **equalize** :

**equalize** est une fonctionnalité qui permet d'aplatir l'histogramme pour donner une image plus contrasté. L'égalisation d'histogramme permet de mieux répartir les intensités sur l'ensemble de la plage de valeurs possibles, en « étalant » l'histogramme. L'égalisation est intéressante pour les images dont la totalité, ou seulement une partie, est de faible contraste (l'ensemble des pixels sont d'intensité proches). Pour cela on calcule l'histogramme cummulé ( Défini par récurrence  $histogrammeCummule[n] = histogramme[n] + histogrammeCummule[n-1]$  pour  $0 < n < 256$  avec  $histogrammeCummule[0] = 0$ ), ensuite on peut exprimer l'image par  $I'(x,y) = histogrammeCummule[I(x,y)] * 255 / NombreDePixel$ . Pour ce qui est des images en couleurs on peut appliquer l'algorithme sur l'ensemble des composantes rouge, bleu et vert toutefois ceci dégrade les couleurs, et n'est donc pas utilisé en pratique. La méthode utilisée est de réaliser l'égalisation uniquement sur les intensités en utilisant la composante V de l'espace HSV.



### 2.2.3 Algorithmes de convolution

Dans le traitement d'images la convolution est une transformation locale en utilisant le voisinage de chaque pixel. Pour cela une matrice de convolution (ou masque) est utilisée pour le floutage, l'amélioration de la netteté de l'image, le gaufrage, la détection de contours, et d'autres utilisations.

Il est nécessaire d'avoir une fonction qui permet l'application de masque, c'est pour cela qu'on a implémenté `applyConvo` fonction qui renvoie un `int[]` qui n'est autre que le tableau des valeurs de pixels de l'image après application du masque, et qui prend en paramètre, le `Bitmap` représentant l'image à traiter, la matrice de convolution (le masque), un entier représentant la taille du masque, un entier représentant le coefficient de normalisation, et un booléen qui indique si les valeurs finales des pixels doivent être dans [0,255]. Le prototype de la fonction est donc :

```
int[] applyConvo(Bitmap bmp, int[][] mask, int taille, int cof, boolean option)
Voici les filtres de convolution implémentés :
```

- Filtre Moyenneur et filtre de Gauss : Les filtres moyenneur et Gauss sont des filtres passe-bas, ils ont une utilité particulière pour lisser l'image (effet de flou), réduire le bruit et réduire les détails.

Le filtre moyenneur se caractérise par une matrice dont tout les coefficients sont égaux

$$\text{et leur somme} = 1. \text{ La matrice de convolution } 3*3 M = \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

Pour appliquer le filtre moyenneur on utilise la fonction vue précédemment :

```
applyConvo(bmp, M, 3, 9, True)
```

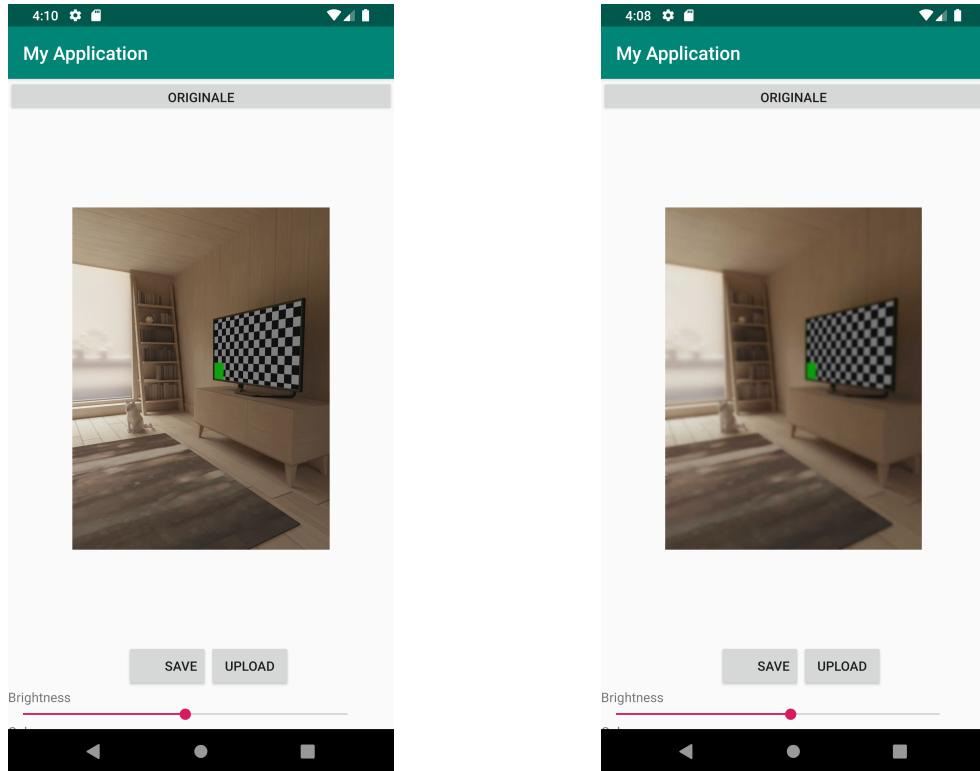
Le filtre de Gauss est plus pratique, il donne un meilleur lissage et une meilleure réduction du bruit que le filtre moyenneur. Sa matrice de convolution se définit par une

gaussienne et que donc les coefficients sont exponentiellement plus faibles quand ils s'éloignent du centre de la matrice. Un exemple de matrice 5\*5 de convolution du filtre

$$\text{de Gauss } G = \frac{1}{98} * \begin{pmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 6 & 8 & 6 & 2 \\ 3 & 8 & 10 & 8 & 3 \\ 2 & 6 & 8 & 6 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{pmatrix}$$

Pour appliquer le filtre Gaussien on utilise la fonction vue précédemment :

`applyConvo(bmp , G, 5, 159, True)`. ci-dessous un aperçu de l'effet du filtre gaussien :



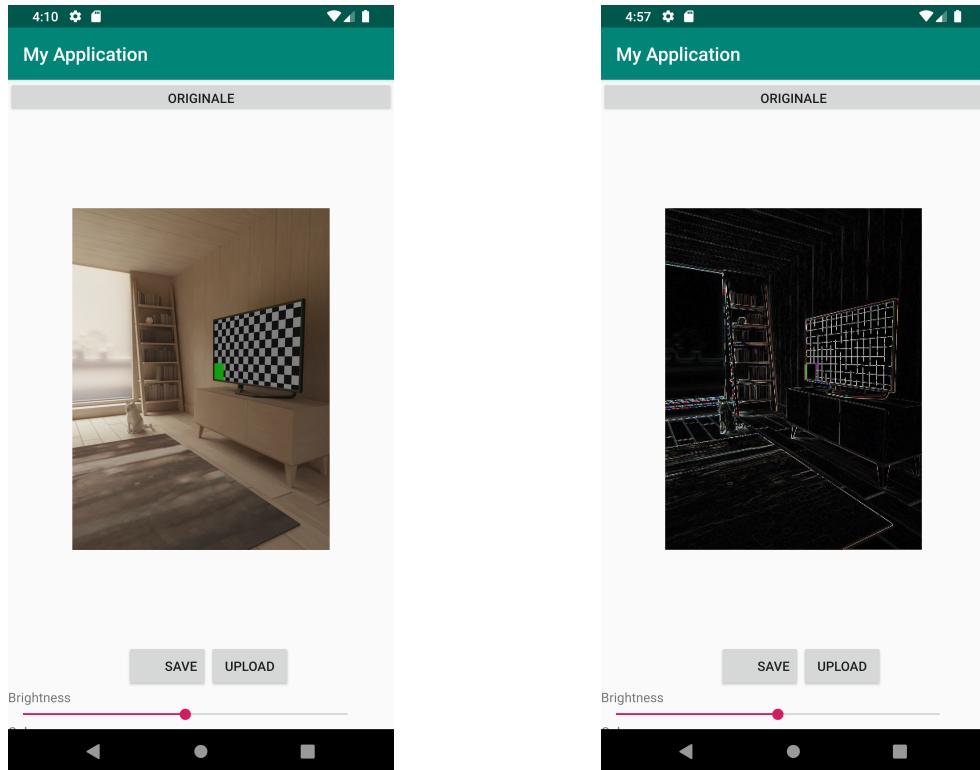
- Filtre de Sobel calcule le gradient de l'intensité de chaque pixel. Ceci indique la direction de la plus forte variation du clair au sombre, ainsi que le taux de changement dans cette direction. On connaît alors les points de changement soudain de luminosité, correspondant probablement à des bords, ainsi que l'orientation de ces bords. Pour cela on utilise deux matrices de convolution (opérateurs de sobel)  $Sx = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$  et

$$Sy = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

On applique chacun des opérateur sur l'image on obtient 2 images  $I_x$  et  $I_y$ . L'image résultante après application du filtre de Sobel se définit par :

$$\forall (x, y) \in [0, Length] * [0, Width], I(a, b) = \sqrt{I_x(a, b)^2 + I_y(a, b)^2} \quad (1)$$

ci-dessous un exemple de l'application du filtre de Sobel :



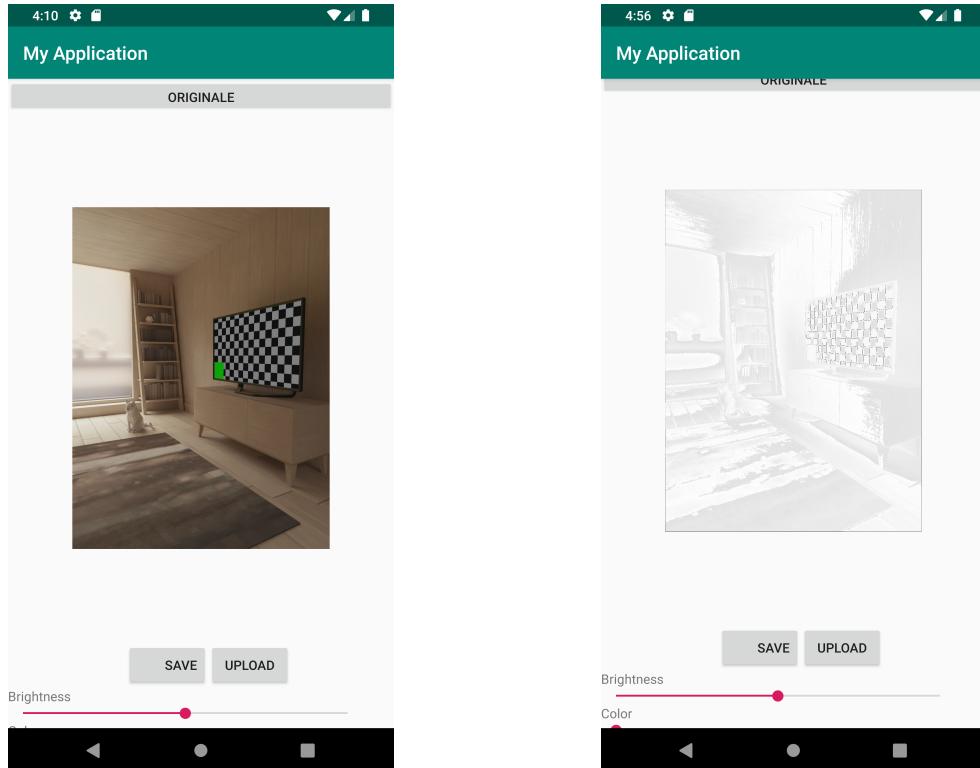
— Filtre laplacien :

Le filtre laplacien est aussi un filtre passe haut pratique pour la reconnaissance de contour, il est équivalent à une dérivée seconde de l'image. Ce filtre se caractérise par sa

$$\text{marice de convolution } L = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \text{ ou } L = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Pour appliquer le filtre Laplacien on utilise la fonction vue précédemment :

`applyConvo(bmp , L, 3, 1, False)`. Ci-dessous un aperçu de l'effet du filtre Laplacien :



## 2.2.4 Fonctionnalités avancées

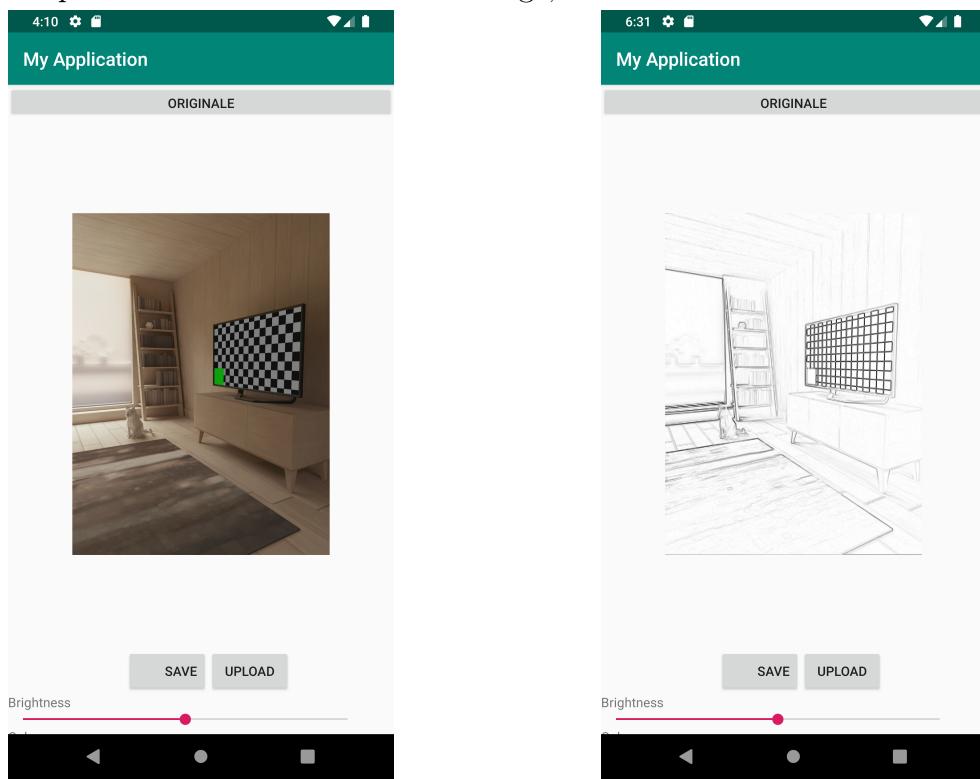
Dans le cadre du projet, on a décié d'implémenter 2 fonctionnalités avancées proposées au choix dans le cahier des charges, ces deux fonctionnalités ont nécessité l'utilisation et la combinaison de plusieurs formes de convolutions :

### 1. Effet crayon :

La fonctionnalité **pencil**, très populaire dans les applications de traitement d'image et très appréciée par le public. Cet effet transforme les images en un dessin d'atriste réalisé au crayon de bois.

Pour implémenter cet effet il a fallu l'analyser afin de comprendre comment cet effet est construit. Les premières remarques qui viennent a l'esprit c'est les couleur noir et blanc généralisées, les contours sont mis en valeur en ayant étant l'élément le plus sombre qui définit l'image de dominance blanche.

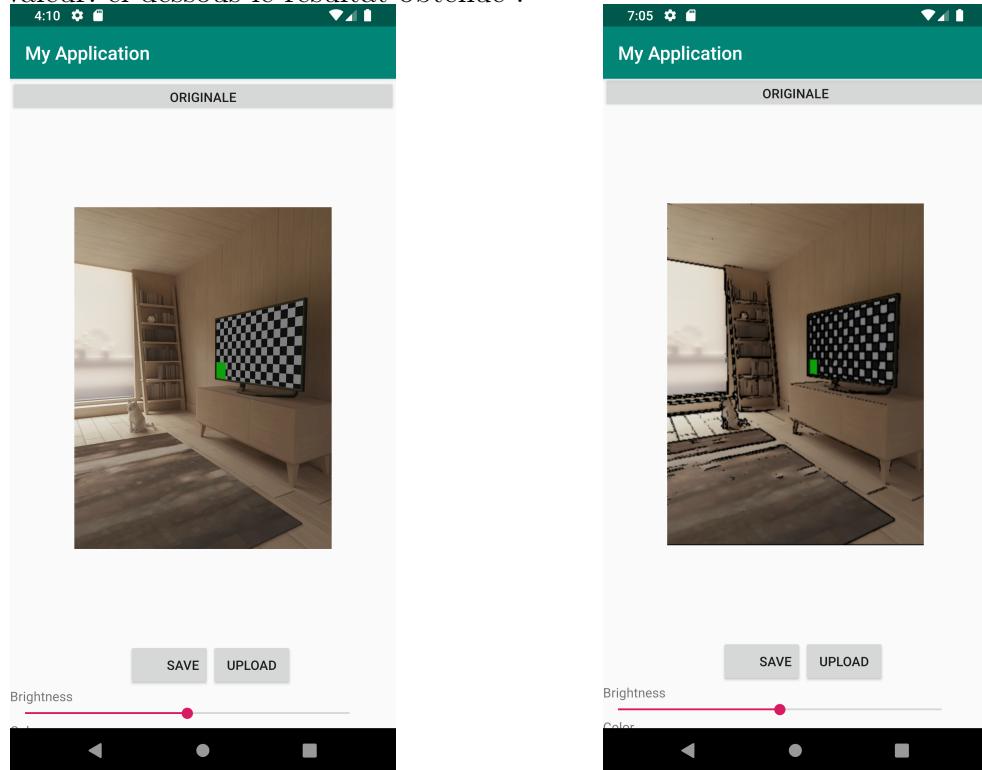
Notre première conclusion était de penser que la superposistion d'un filtre sobel, d'un inversement de couleurs puis d'un grisage de l'image suffirait pour produire l'effet, cependant l'effet attendu n'y était pas. En effet beaucoup de bruit a été gardé et donc "déssiné au crayon". Pour remédier a cela on applique un filtre de gauss a l'image avant traitement pou réduire le bruit et lisser l'image, le résultat est satisfaisant



### 2. Effet cartoon :

L'effet cartoon est un effet très populaire aussi, en effet il donne l'aspect d'une bande dessinée. Pour réaliser cet effet il faut restreindre les couleurs d'une image et renforcer les contours afin de lui donner un effet de bande dessinée. Pour ce qu'il s'agit de restreindre les couleurs on peut utiliser l'espace Hsv pour augmenter la luminance des couleurs. Concernant les contours on peut utiliser notre effet **pencil** qui nous donneras une image avec les contours bien démarqués se rapprochant du noir. En tracant les contours (c.à.d

en rapportant les pixel sombre) sur l'image original en noir on obtient des contours bien mis en valeur. ci-dessous le résultat obtenu :



### 3 Performance et complexité

Maintenant qu'on a conscience des solutions algorithmiques proposées pour l'implémentation de l'application, il serait intéressant d'en analyser la complexité et de comparer les performances :

#### 3.0.1 Manipulation de teintes et de couleurs

En ce qui concerne les algorithmes de manipulation de teintes de couleurs tels que `colorize` ou `onlyGreen` ce sont des algorithme qui demande un seul parcourt de l'image et qui sont donc en complexité linéaire à la taille (hauteur x largeur) de l'image, cependant ces deux algorithmes comportent des opérations très couteuses en termes de temps. En effet le passage de l'espace HSV à partir de l'espace RGB semble être une opération très couteuse qui ralentit l'algorithme comme le montre le profilage CPU.



La solution à ce problème serait de manipuler directement l'espace RGB, et de trouver les formules adéquates afin de modifier la teinte sans besoin du passage à l'espace HSV.

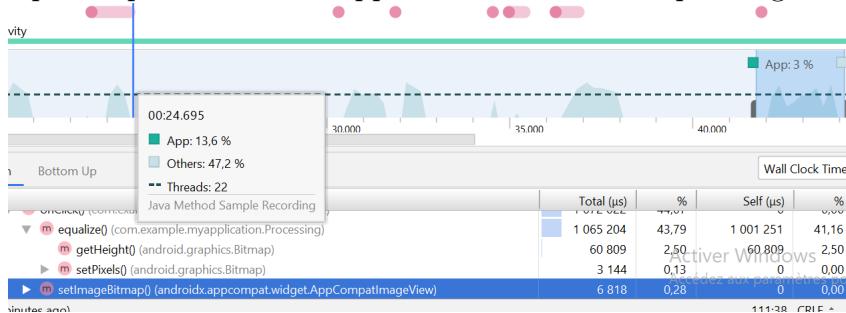
D'ailleurs on rencontre le même problème avec la barre de luminosité.

### 3.0.2 Transformation de l'histogramme

En ce qui concerne les fonctionnalités de transformation de l'histogramme, elle peuvent parfois susciter un espace mémoire particulier (tableau d'histogramme , histogramme cumulés...), cependant la n'est pas le problème. Le profilage CPU montre que ces algorithmes ont recours plusieurs fois (chaque fois dans leurs boucles linéaires) à des appels de fonction inutiles tels que `Bitmap.getWidth()`, des appels couteux dont la suppression serait bénéfique

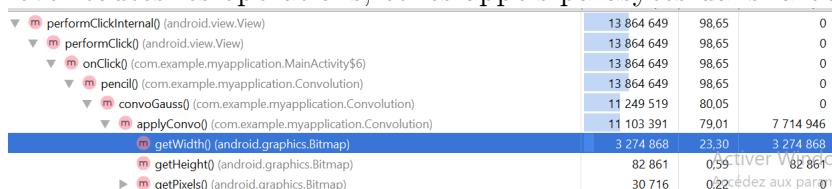


Après suppression de ces appels inutiles on remarque un gain de 50% de temps

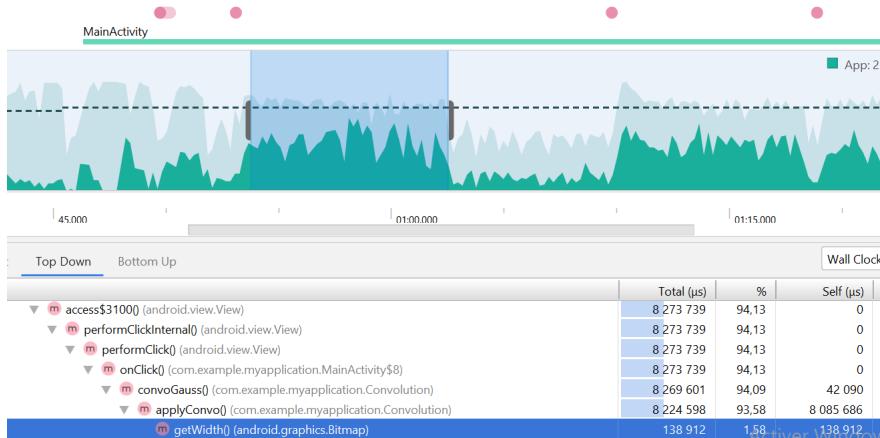


### 3.0.3 Algorithmes de convolution

Toutes les formes de convolution dans le sujet s'appuient sur une seule fonction `applyConv()`, celle-ci a une complexité en temps de  $O(N^2 \cdot n)$  ( $N$  étant la taille de l'image  $h \times w$ , et  $n$  la taille du masque). Les algorithmes de convolution peuvent prendre un certain temps d'exécution puisque par exemple pour appliquer un masque  $5 \times 5$ , cela est équivalent à parcourir l'image 25 fois, si celle-ci possède une grande taille le traitement risque de tarder. Pour cela il est essentiel d'enlever toutes les opérations, et les appels parasites dans la boucle de la fonction principale.

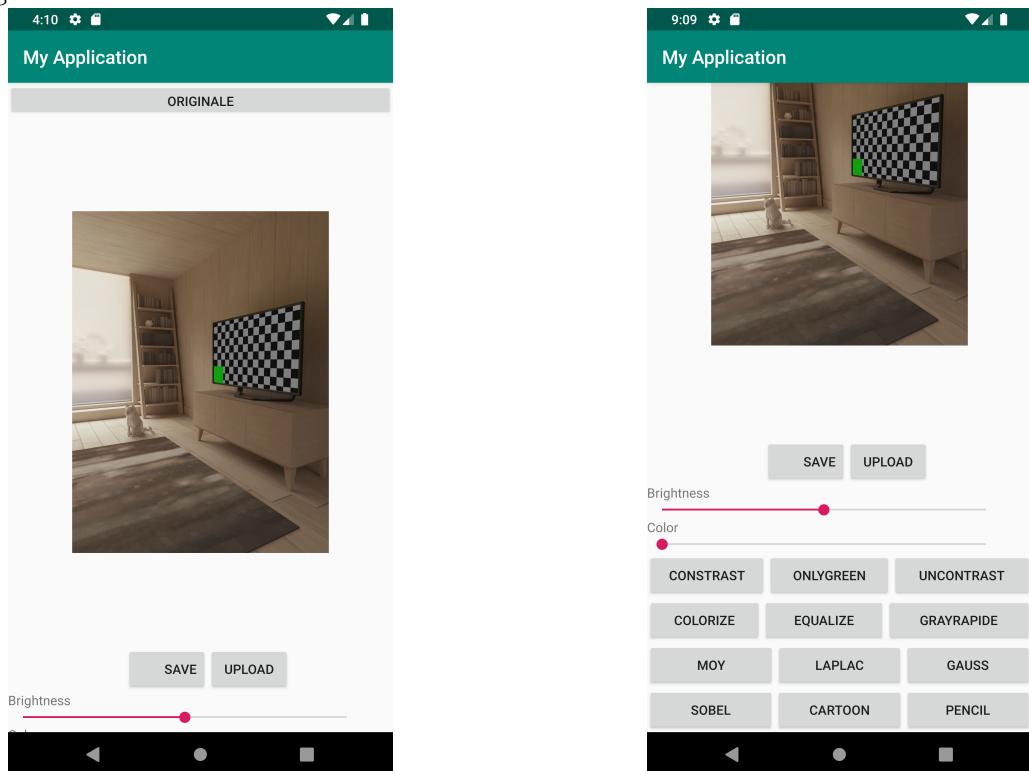


En supprimant ces appels et en optimisant le code de la boucle on réussit à avoir un gain de temps de 25% comparé au temps initial :



## 4 Interface utilisateur

L'interface utilisateur est un point important du projet. En effet l'implémentation d'une interface correcte fait partie des principaux objectifs qu'on a établit dès le début. Pour cela on a pensé à une interface simple et facile d'utilisation, présentant la **totalité des fonctionnalités en une seule page** pour ne pas plonger l'utilisateur dans l'ambiguïté. On a donc décidé d'instaurer une page au contenu scrollable laissant une place conséquante à l'image. Le fait d'instaurer une scrollView ne **prend pas le pas sur la fonctionnalité Zoom/scroll de l'image**, vous l'avez compris l'application fait la différence quand il s'agit de scroller la page ou l'image en elle-même.



## 5 Points à approfondir

### 5.1 Utilisation de RenderScript

L'utilisation du RenderScript est un point essentiel qui n'a pas toujours été présent dans ce projet. En effet l'utilisation de ce language aurait permis d'avoir des performances plus remarquables et cela en tirant facilement partie des différents coeurs du CPU ou du GPU des smartphones sous Android, et donc en réduisant considérablement les temps d'execution des algorithmes.

### 5.2 Amélioration de certains algorithmes

#### 5.2.1 Algorithmes de contraste

L'algorithme `toContrastDyn` reste innachevée car, incapable de traiter les images en couleurs, il ne peut traiter efficacement que les images en noir et blanc.

L'algorithme `lessContrast` marche bien sur les images en couleurs mais par contre on y voit apparaître parfois des zones de couleur uniforme.

#### 5.2.2 Algorithmes de changement de teintes ou couleur

Comme on l'a vu précédemment les algorithmes de changement ou d'analyse de teintes (utilisant HSV) ne sont pas en général performants. Il serait intéressant de trouver un moyen de les optimiser.

### 5.3 Ajout de fonctionnalités

Il serait intéressant d'implémenter de nouvelles fonctionnalités telle que incruster des objets ou détecter la couleur de la peau humaine ...

## 6 Conclusion

Au delà des connaissances techniques acquises (Traitement d'image , Convolutions , développement html, développement Android...), la réalisation du projet S6 m'a inculqué l'importance de travail en groupe, la nécessité de prise d'initiative, l'importance de ne pas perdre espoir, ainsi que la gestion des situations de crises.