Ministère de l'enseignement supérieur et de la recherche scientifique
Faculté des Sciences de Monastir

Détection Des Objets par Valeurs RGB et Niveaux de Gris

Mohamed Chaker Jouini

LSI1 : TD3/TP6

Nov/Dec 2023

JouiniMohamedChaker@proton.me

INTRODUCTION GÉNÉRALE

INTRODUCTION GÉNÉRALE

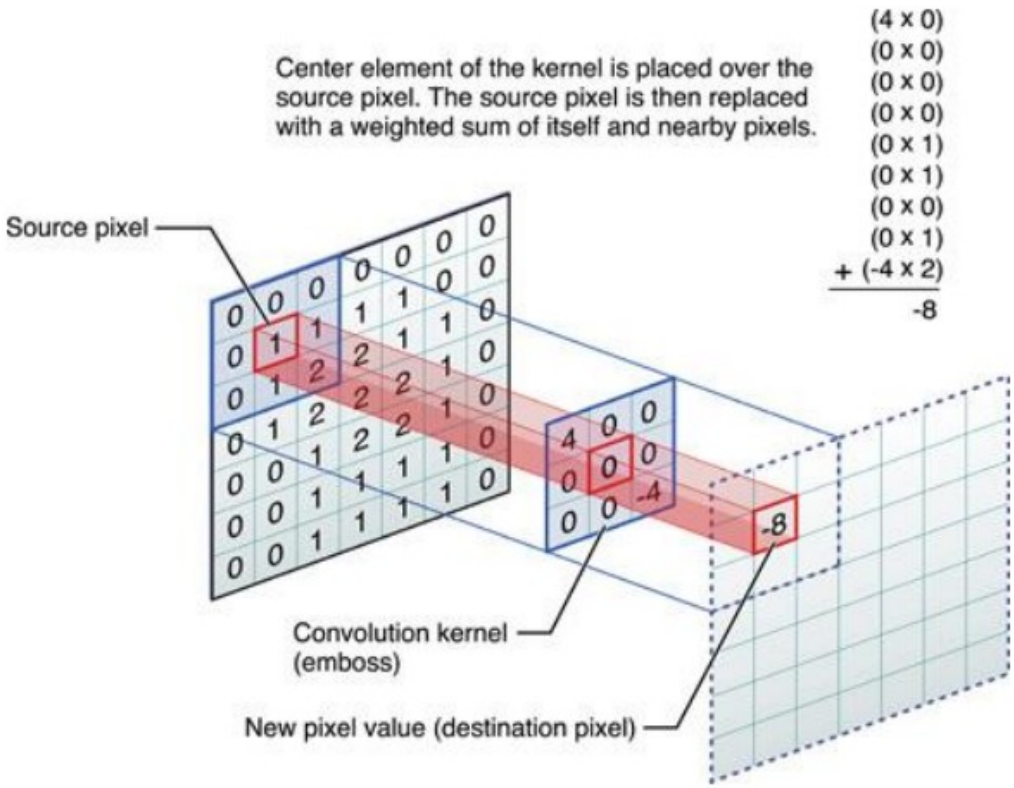
Le domaine du traitement d'image concerne la manipulation, l'analyse et la compréhension des images. Ce qui inclus un large éventail de sous-domaines tel que : la restauration d'image, la segmentation, la reconnaissance de formes, la compression, la reconnaissance de motifs et la vision par ordinateur. En effet chaque objectif a ses propres méthodes et algorithmes spécifiques pour traiter les données visuelles.

Ainsi, les applications du traitement d'image sont vastes et variées, allant de la médecine (comme l'imagerie médicale pour le diagnostic) à la surveillance et la sécurité (la détection d'objets ou de mouvements), en passant par la réalité augmentée, la photographie numérique, l'industrie du divertissement et bien d'autres domaines.

Néanmoins, les défis du traitement d'image sont souvent la gestion de la complexité des données visuelles, la précision des algorithmes pour extraire des informations utiles et la capacité à interpréter ces données de manière significative et correcte.

En constante évolution, le domaine du traitement d'image continue de repousser ses limites grâce aux avancées technologiques telles que l'intelligence artificielle, l'apprentissage automatique et les réseaux de neurones, ouvrant ainsi de nouvelles possibilités pour améliorer la qualité, la vitesse et la précision du traitement d'image.

Pour attaindre ces objectifs, divers logiciels peuvent etre utiliser chacune à utilité spécifique; notamment les logiciels du calcule matricielle (comme Matlab ou Ocatve) qui interagie avec l'image d'un point de vue mathématique. C'est du fait qu'une image peut etre vue comme une fonction à 2 dimensions où chaque point (appelé Pixel) est associé à un valeur numérique représentant un ou des attributs visuels comme la luminosité, la couleur…
Les images numériques sont souvent représentées sous forme de matrices ou de tableaux pour simplifier leur traitement. Dans une image en niveaux de gris, chaque pixel est représenté par une seule valeur, généralement entre 0 et 255, où 0 correspond au noir et 255 au blanc. Dans une image en couleur, chaque pixel peut être représenté par trois valeurs pour les couleurs rouge, vert et bleu (RVB ou RGB en anglais) par exemple, ou 4 valeurs pour les couleus cyan, magneta, jaune et noir (CMJN ou CMYB en angalis) ou plus , selon l'espace colorimétrique utilisé.
En somme, les images d'un point de vue mathématique sont des ensembles de données structurées qui peuvent être manipulées, analysées et interprétées à l'aide d'outils et de concepts mathématiques pour obtenir des informations précieuses et prendre des décisions utiles.
Comme outils on peut utilise Octave qui est un logiciel Open Source paralléle à MATLAB qui est propriétaire. Les deux offre une synatxe conviviale pour effectuer des calculs mathématique complexe, des simulations et d'autre fonctionnalités avancées qu'on peut utiliser pour traiter les images. Dans ce dernier les opérations mathématiques de base telles que l'addition la soustraction la multiplication et la division peuvent être appliquées sur les matrices d'image pour l'ajout de bruit, la modification de la luminosité ou le contraste, on peut aussi appliquer des filtres par des opération de convolution qui consiste à une opération de multiplication de deux matrices de tailles différentes (généralement une petite et une grande), mais de même dimensionnalité (1D, 2D...), produisant une nouvelle matrice (également de même dimensionnalité).



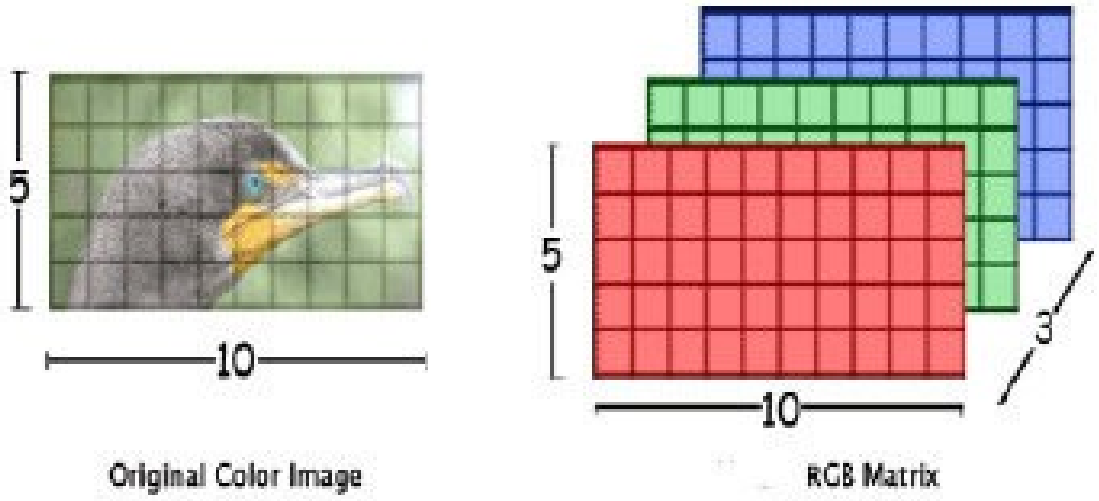
Exemple de convolution 2D (source [1])

Finalement, il ne reste que de traduire nos idées mathématiques dans Octave and expérimenter !

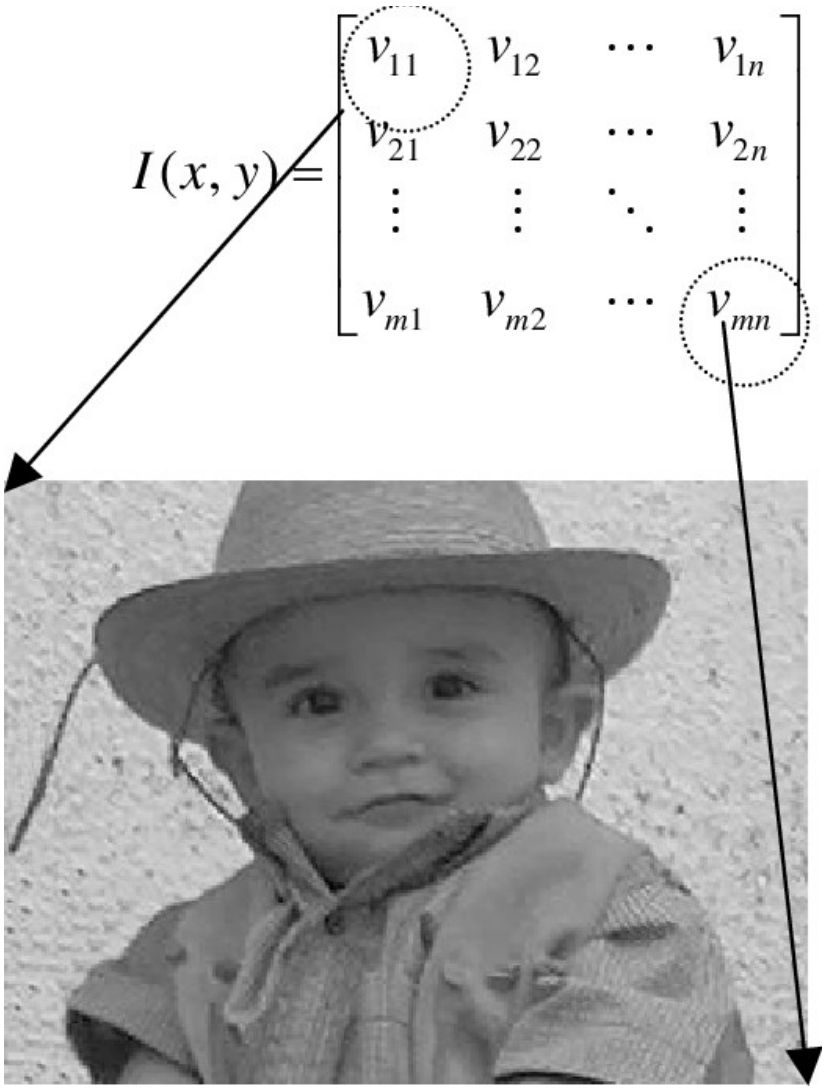
CHAPITRE 1 :
Partie Théorique

CHAPITRE 1:
PARTIE THEORIQUE

Le modèle RGB (Red,Green,Blue) ou RGV est une représentation colormétrique ou les couleurs sont créés par le mélange de trois canaux de couleurs primaire : rouge vert et bleu. Chaque pixel dans une image est donc définit par un mélange de ces trois couleurs où chacun est représenté par un entier qui varie entre 0 et 255.
Cette représentation nous permet de convertir une image à une matrice à 3 dimensions , le troixieme contient le triplet des valeurs et les deux autres dimensions varie de taille par à rapport à l'image:



Un autre modèle souvent utilisée est le Grayscale ; les images en niveaux de gris, contrairement aux images en couleurs ils ne contiennet pas d'information sur la couleur puisque chaque pixel est représenté par une seule valeur qui indique son luminosité. Les valeurs les plus élevées représentent généralement des tons clairs qui sont plus proche du blanc, tandis que les valeurs plus basses représentent des tons plus sombres qui sont proche du noir.
Ce modèle nous permet de représenter une image par une matrice à 3 dimensions mais le troisième contient un seul valeur qui varie entre 0 et 255 :



On va utiliser ces modèles de représentation, RGB et Grayscale, pour essayer de détecter les objets dans une image donné et puis comparé les résultats.

Pour les images RGB on peut implémenter un idée simple pour identifier les objets : grouper les pixels ayant des couleurs similaires. On parcours la matrice en comparant les valeurs des deux pixels adjacents dans une meme ligne par définir un variable limite qui va controler le niveau de similarité des couleurs.

Exemple : Un pixel P1 ayant comme valeurs RGB (100,123,233) et P2 ayant (105,150,230) , P1 et P2 sont similaires si le variable limite est égale à 27 puisque la différence entre les valeurs de ces deux vecteurs (pixels) nous donne :

résultat= Abs(100,123,233) – (105,150,230))= (5,27,3)

Puis on fait notre comparaison, en pseudo-code c'est :

Début

Pour i de 1 à 3 faire

si le i ème élément de la résultat est supérieur à la variable limite : les pixels sont différents

fin pour

sinon ils sont similaires

Fin

On ennregistre les indices des pixels similaire pour les identifiers aprés le 1er parcours de l'image, Aprés on peut parcourir l'image un autre fois est affecter des valeurs nuls aux pixels qui ne sont pas enregistrer, Ou on peut créer un nouvelle image vide de meme taille et transférer les valeurs RGB de l'originale par leurs indices vers la nouvelle dans le meme emplacement. On a plusieurs manière de visualiser les résultats mais le plus important est le faitre qu'on a les indices des pixels similaires , qui sont par notre raisonnement des objets potentiels .

De l'autre coté on va utilier l'image en niveaux de gris pour créer un image binaire par comparer le valeur dans chaque pixel par un valeur limite qui signifie l'intensité accepté pour un objet, c'est bien plus utile à séparer l'arriére plan des objets en face mais c'est une identification aussi. L'implentation est plus simple, en pseudo-code c'est :

Début

Covertir l'image en grayscale par des fonctions prédéfinis

créer une image binaire de meme taille

Variable limite = entier entre 0 et 255

Pour i de 0 à (nombre des lignes) -1

pour j de 0 à (nombre des colonnes) -1

si Valeur de gris de Pixel(i,j) de l'image gris>variable limite

Pixels(i,j) de l'image bianaire prend valeur 1

sinon il prend 0

c’est suffisant pour atteindre notre objectif mais on peut parcourir l’image originale et maintenir son valeur RGB si le pixel correspondant dans l’image a 1 comme valeur, si c’est 0 on change le couleur en blanc ou noir. C’est juste pour bien visualiser les résultats.

CHAPITRE 2 :
Partie Pratique

CHAPITRE 2:
PARTIE PRATIQUE

Méthode du Model RGB :

```
img= imread("/***/l'image à traiter");

[originalRows, originalColumns, ~] = size(image);
%on prend les dimensions de l'image

%fonction pour comparer les valeurs RGB
function result= similaire(pixel1, pixel2)
  %variable de configuration du similarite acceptee dans le groupement
    limite = 10;
    %on calcule la differnce des valeurs
    diff = abs(pixel1 - pixel2);
    %et on les compare avec le variable de configuration
    result = all(diff <= limite);
endfunction

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%groupement des indices des pixels a couleurs similaires
%la comparaison et seulement faite horizentalement (pixels de meme ligne)
cluster = [];

%iteration des pixels de l'image
for i = 1:rows
  for j = 1:(columns-1)
```

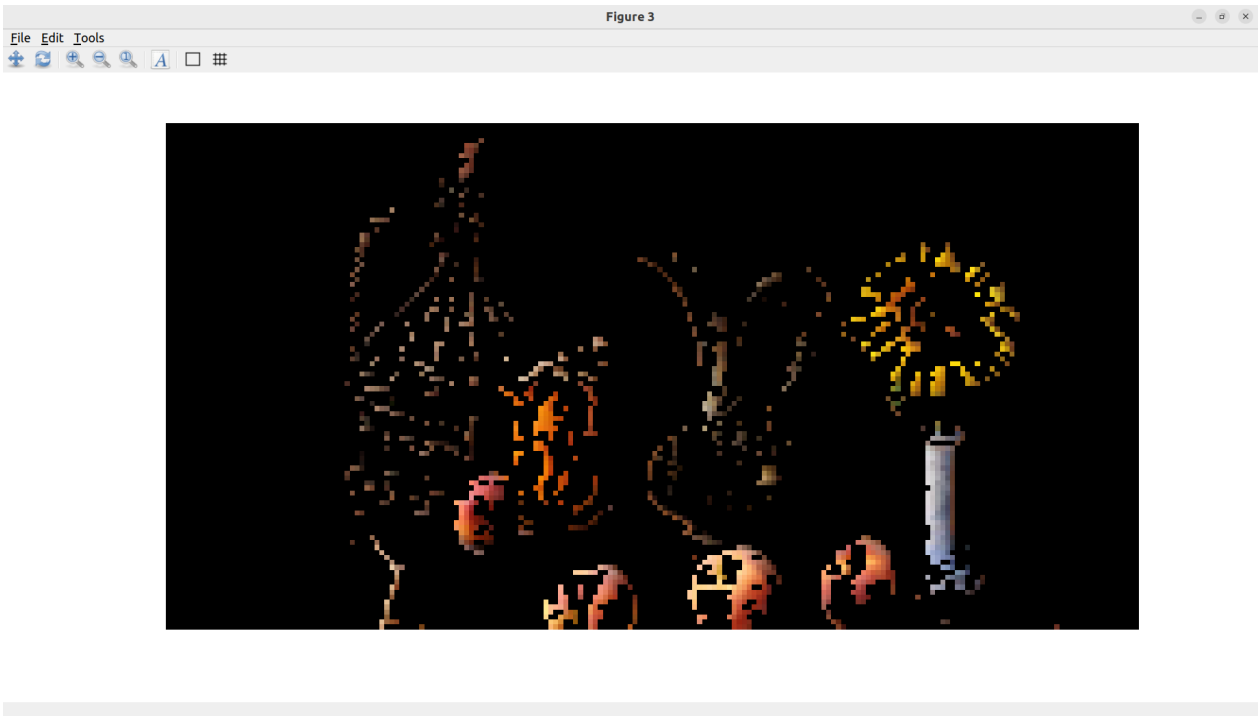
```
    if j+1 <= columns  %le compteur doit respecter les dimensions de l'image
      if similaire(img(i, j, :), img(i, j+1, :))
%img(i,j,:) est un vecteur à 3 dimensions qui contient les valeurs RGB du pixel(i,j)
        cluster = [cluster; i, j; i, j+1];
%on ajoute les indices des pixels similaires
      endif
    endif
  endfor
endfor

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%identifier les groupements et affecter des valeurs RGB nuls aux pixels
for k = 1:2:size(cluster, 1)
   img(cluster(k, 1), cluster(k, 2), :) = 0;
   img(cluster(k+1, 1), cluster(k+1, 2), :) = 0;
%on les rend noir pour visuliser les groupements trouvés
endfor
figure;imshow(img); %affichage du résultat
```

Résultats de cette méthode :

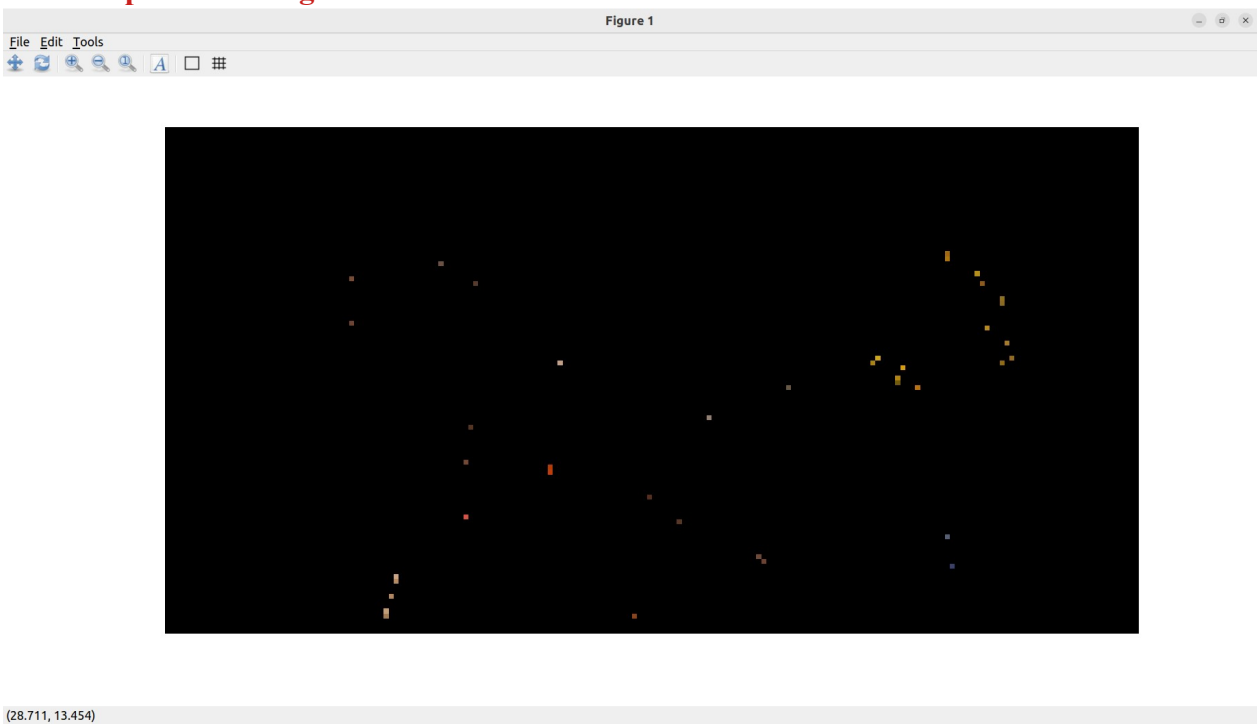
image originale:

Résultat pour limite égale à 10:

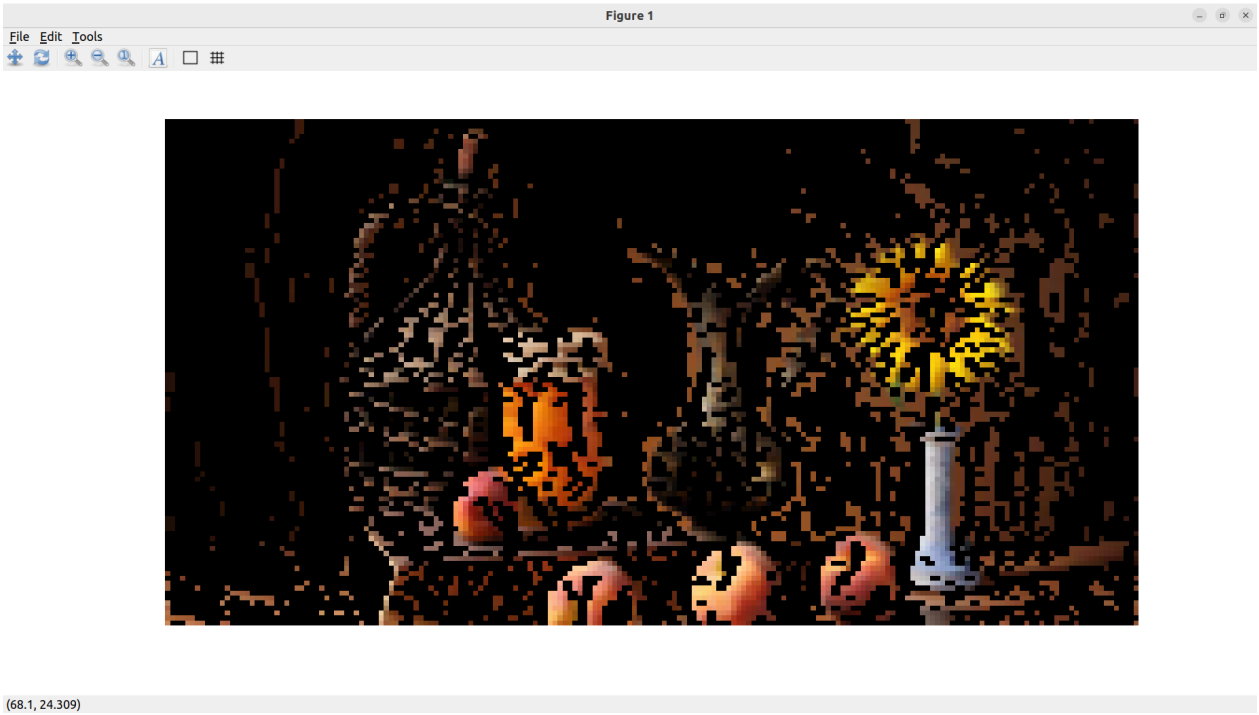
meilleur cas



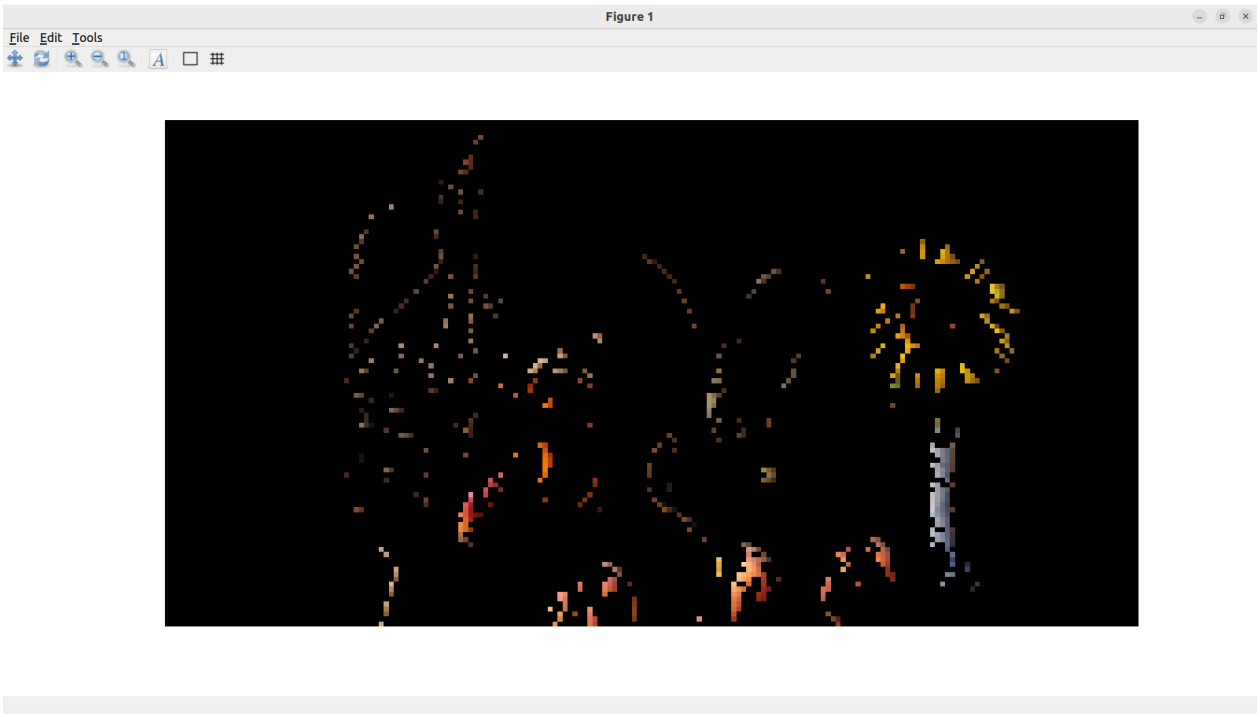
Résultat pour limite égale à 50:



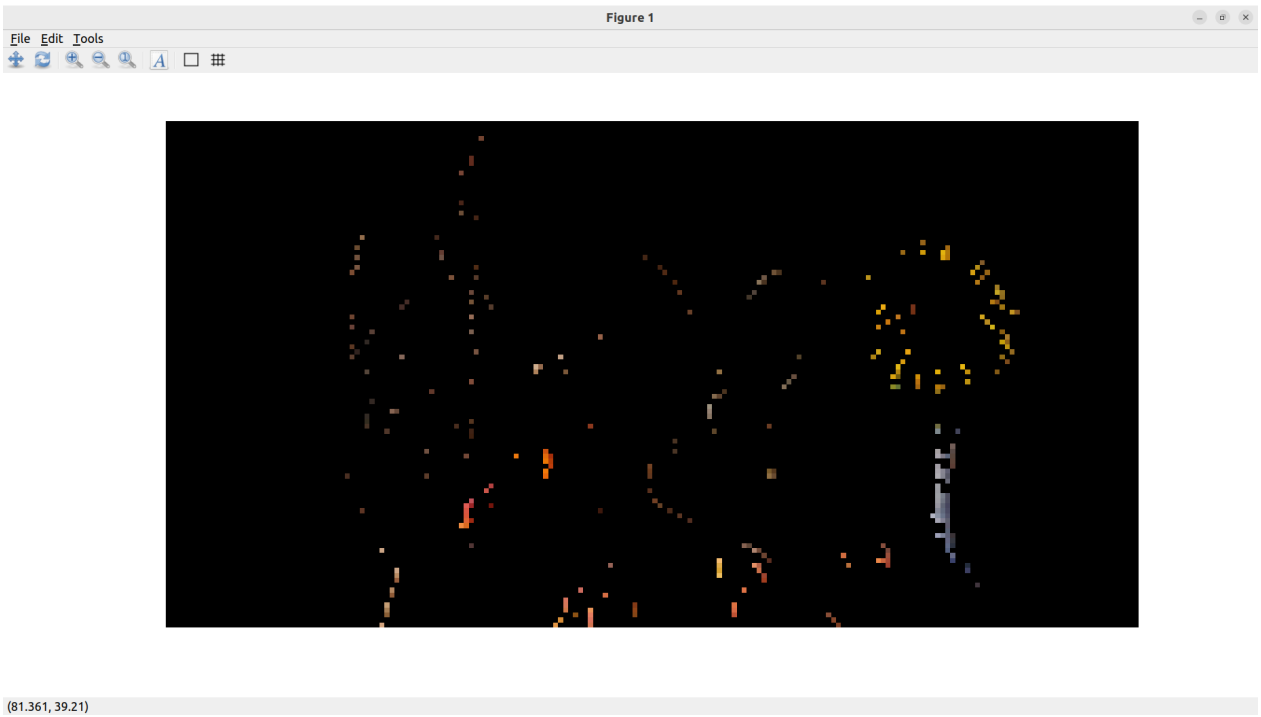
Résultat pour limite égale à 1 :



Résultat pour limite égale à 20 :



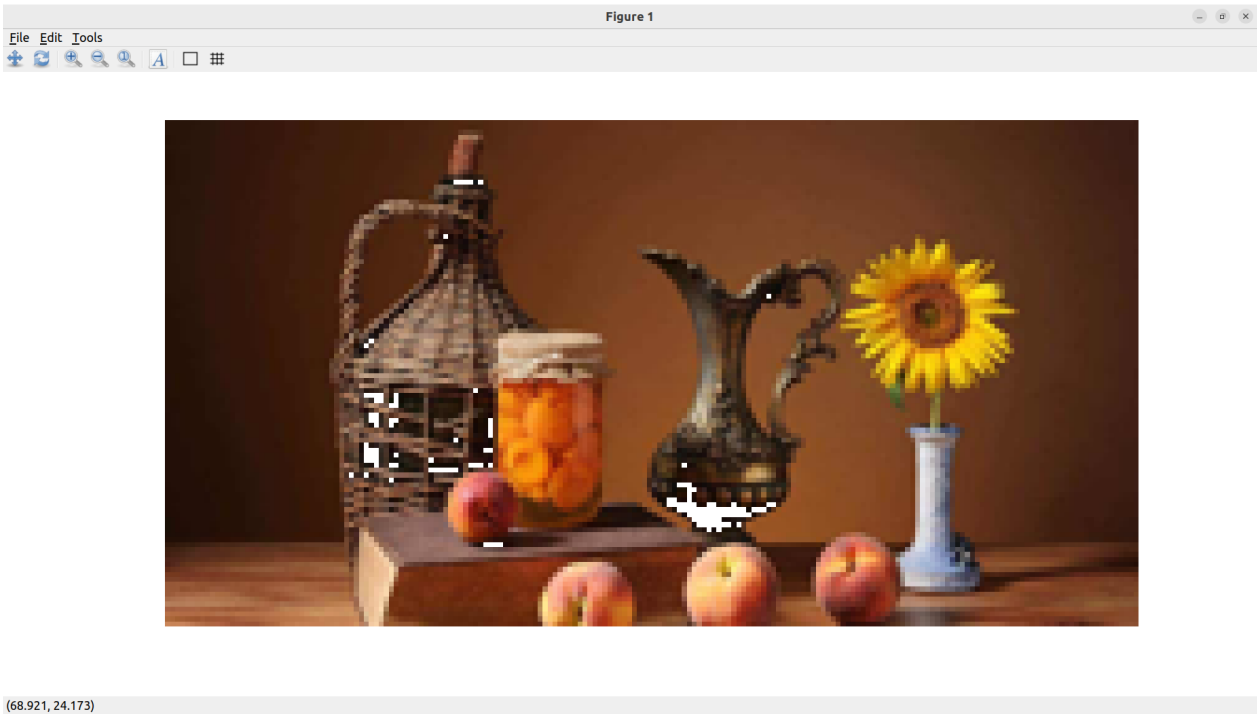
Résultat pour limite égale à 30 :



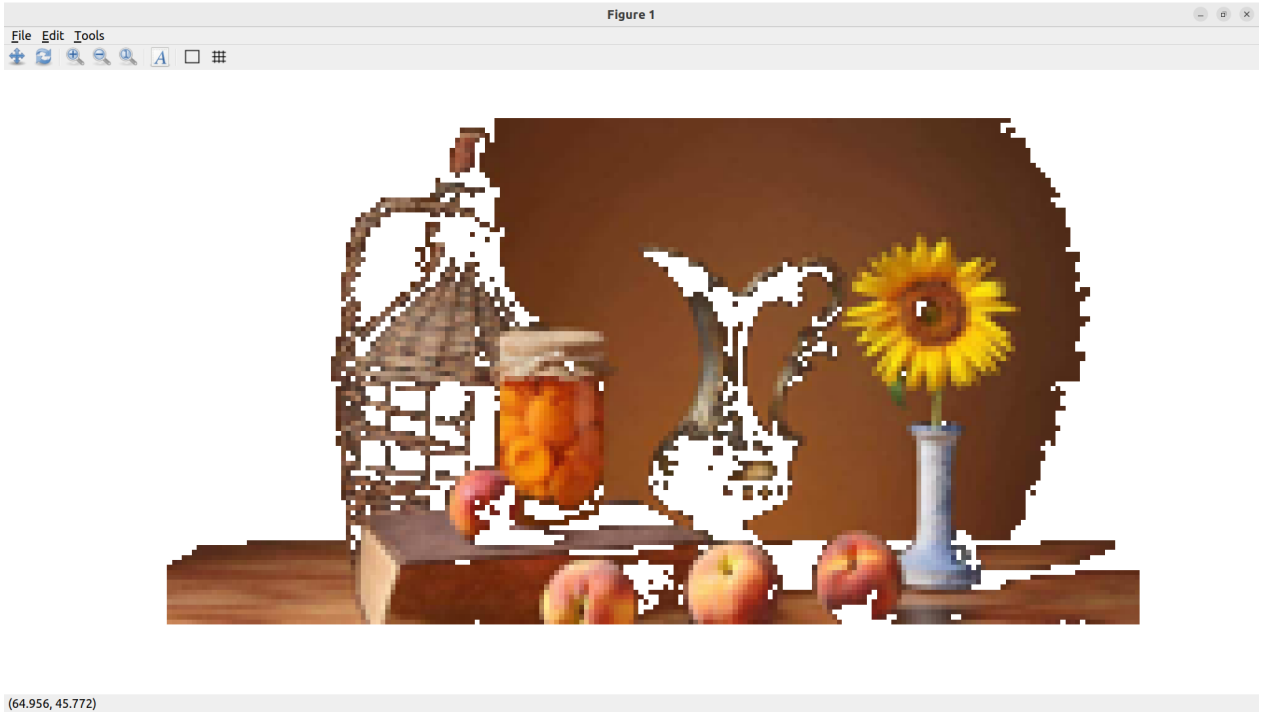
Méthode du Model Grayscale :

```
image = imread("/***/notre image");
[originalRows, originalColumns, ~] = size(image);
%on prend les dimensions de l'image
gray = rgb2gray(image);
%conversion de l'image RGB en Grayscale par une fonction prédéfinie
limite = 190; %le variable limite
imagebinaire = gray> limite; %creation de l'image binaire
%cette copmaraison va créer un matrice de meme dimensions que gray et affecter 1 si la
%comparaison est vrai (valeur d'une case est supérieure à la variable limite) sinon 0
[L, num] = bwlabel(imagebinaire);
%la fonction prédéfinie bwlabel étiquette les composants connectés dans l'image binaire et
%renvoie l'image étiquetée L ainsi que le nombre de composants connectés num
[rows, columns] = size(L);
%Récupèrer le nombre de lignes et de colonnes dans l'image étiquetée L
highlight = uint8(repmat(image, [1, 1, 1]));
%créer une copie de l'image originale
%Les boucles suivantes parcourent chaque pixel de l'image étiquetée L
for i = 1:rows
   for j = 1:columns
      if L(i, j) == 0
         highlight(i, j, :) = [255, 255, 255];
%Si le pixel correspondant dans L est étiqueté comme 0 on change le pixel dans l'image highlight
%en blanc
      endif
   endfor
endfor
figure;imshow(highlight); %afficher la résultat
```

Résultat pour limite égale à 10:

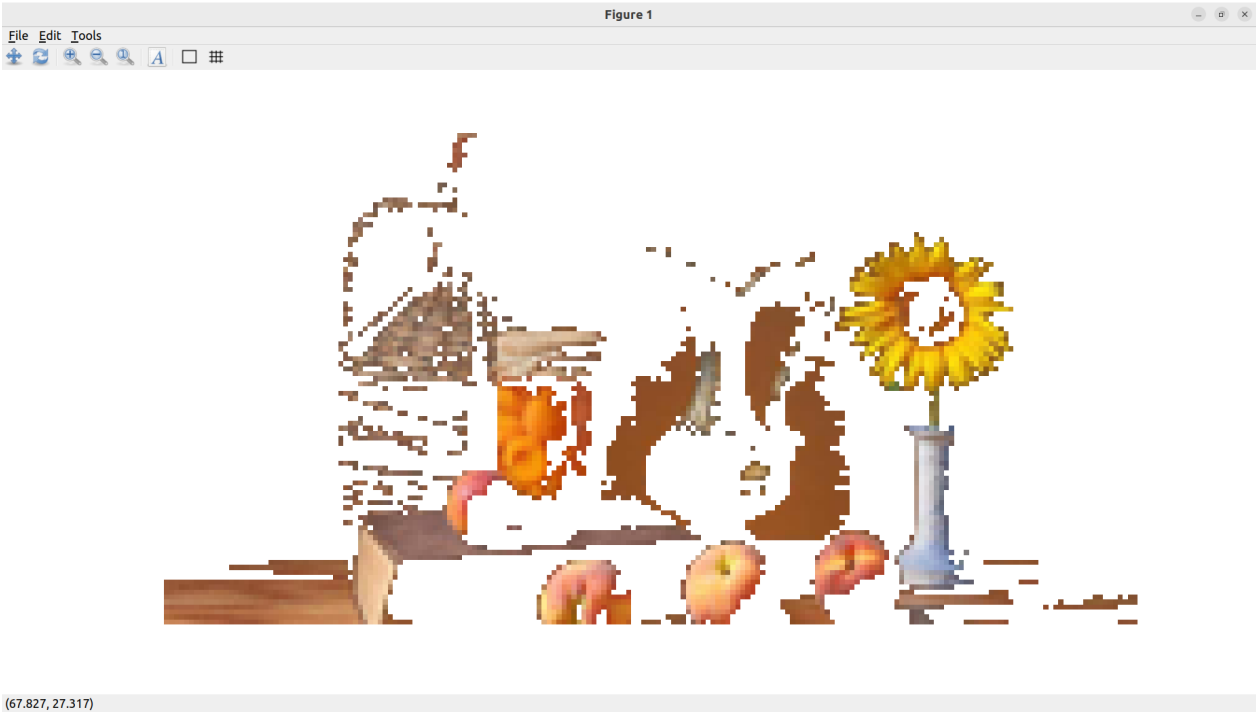


Résultat pour limite égale à 50 :

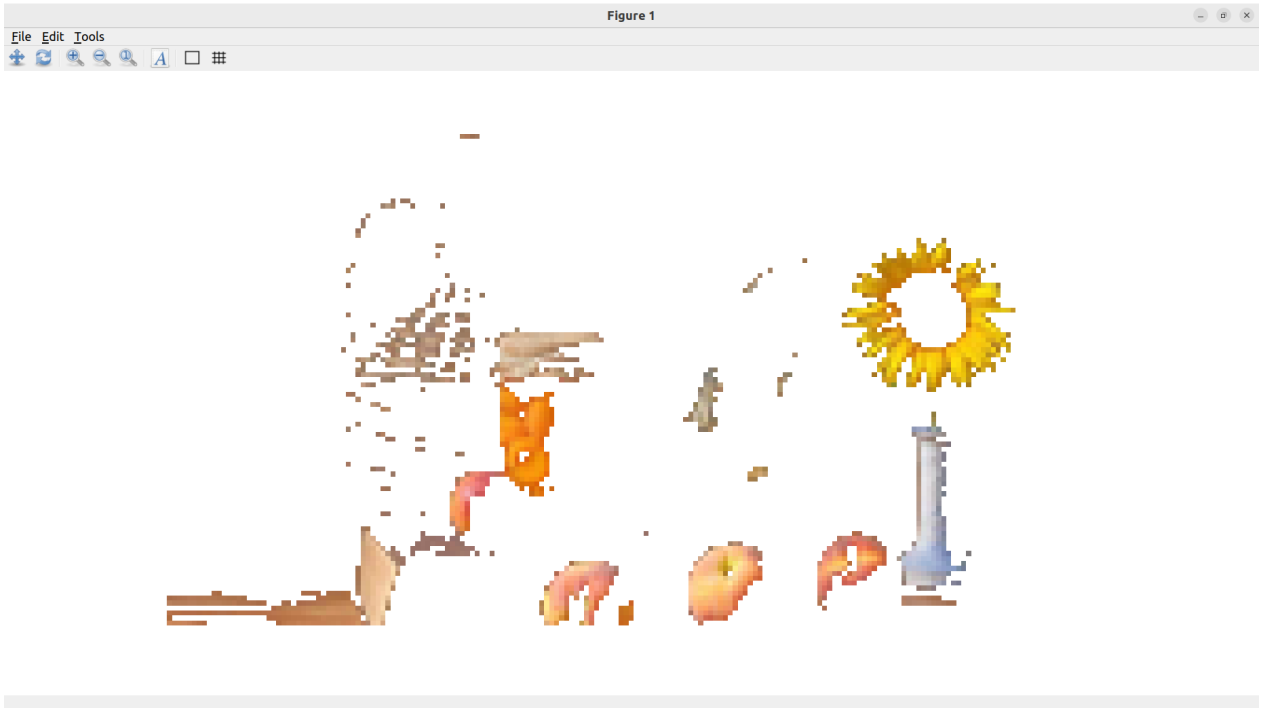


Résultat pour limite égale à 90 :

meilleur cas



Résultat pour limite égale à 120 :



Résultat pour limite égale à 160 :