

# **Rapport du projet**

## **« Traitement d'images et vision »**

### **Travail élaboré par :**

**Baghrosse Rachid**  
**Bayechou Abdelhadi**  
**Bounda Elie**  
**Jaridi Mustapha**

### **Encadré par :**

**Mme. Alice Porebski**

# Table de matières

<b><i>I. Introduction</i></b>	<b>3</b>
<b><i>II. Partie théorique</i></b>	<b>3</b>
II.1. Les outils	3
II.2. Les algorithmes	3
<b><i>III. Apprentissage sans couleurs</i></b>	<b>5</b>
III.1. Apprentissage	5
III.2. La classification	7
III.3. Teste de modèle	8
<b><i>IV. Apprentissage avec couleurs</i></b>	<b>12</b>
<b><i>V. Conclusion</i></b>	<b>14</b>

# I. Introduction

La reconnaissance faciale est une technique basée sur l'intelligence artificielle, permettant d'identifier une personne sur une photo ou une vidéo en comparant son visage avec ceux sauvegardés dans une base de données.

La reconnaissance de visages est de plus en plus utilisée de nos jours, notamment pour des aspects sécuritaires.



## II. Partie théorique

### II.1. Les outils

Pendant la réalisation de ce TP, on a eu besoin des outils suivants :

- Une base de données contenant 50 classes, chaque classe contient 12 images.
- Matlab pour le développement des algorithmes de notre application.
- CascadeObjectDetector pour la détection de la forme de visage dans une image.
- Attributs de texture pour la reconnaissance des images.

### II.2. Les algorithmes

#### a. Transformation d'une image couleur en une image en niveau de gris

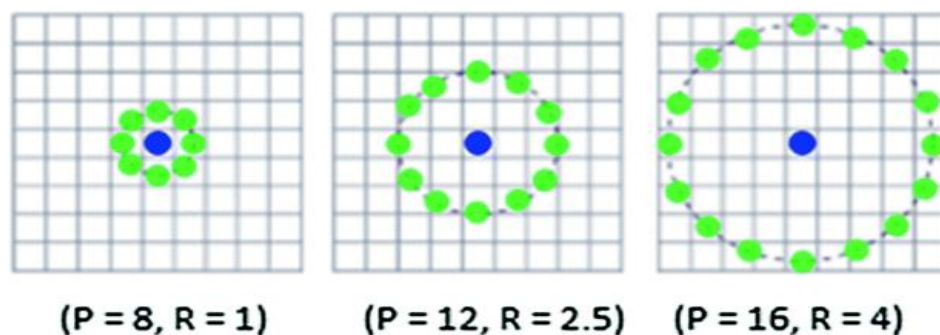
Algorithme qui permet de passer de l'espace RGB vers l'espace de niveau de gris. Cet algorithme est codé sous forme d'une fonction en Matlab « `rgb2gray()` ».

#### b. Motif binaire local (LBP: Local Binary Patterns)

Permet de calculer les motifs locaux binaires d'une image monochrome avec différents paramètres.

- Le nombre de voisins à analyser du voisinage spatial
- Le rayon de cercle sur lequel les voisins se situent
- Le mapping qui permet d'avoir accès aux variantes des LBP grâce à la fonction `getmapping` :
  - 0 : LBP classique
  - 'u2' : LBP uniforme qui représente une version réduite de LBP classique
  - 'ri' : LBP invariant en rotation
  - 'riu2' : LBP uniformes invariant en rotation
- Mode : qui définit le type de retour de la fonction LBP
  - 'h' ou 'hist' : pour avoir l'histogramme de LBP
  - 'nh' pour obtenir l'histogramme normalisé des LBP

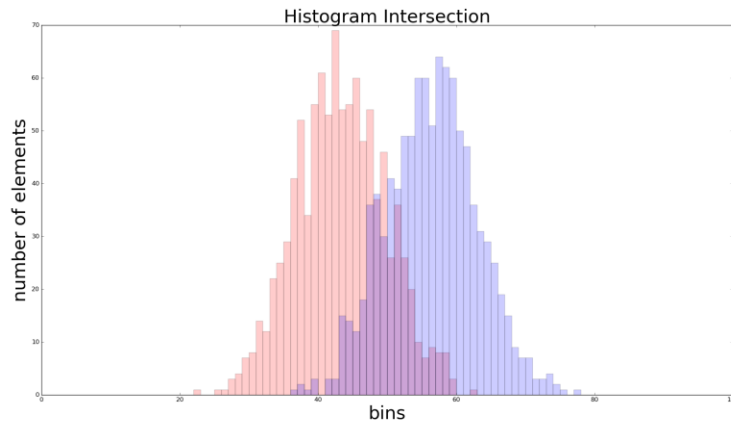
Pour tester ces paramètres, on fait varier les 3 premiers paramètres en fixant le dernier pour visualiser leur effet sur le temps de traitement et le taux de classification, dont ces deux derniers représentent deux facteurs de la rigueur de notre application.





### c. Intersection de l'histogramme

Intersection d'histogramme est une technique qui permet de mesurer la similarité entre deux images, il se calcule par la formule suivante :



Soit deux histogramme **I** et **M** de deux images différents, l'intersection de leur histogramme est :

$$S = \sum_{j=0}^n \min (I_j, M_j)$$

La valeur de S est à maximiser, plus la valeur de S est grande plus les images I et M sont similaires.

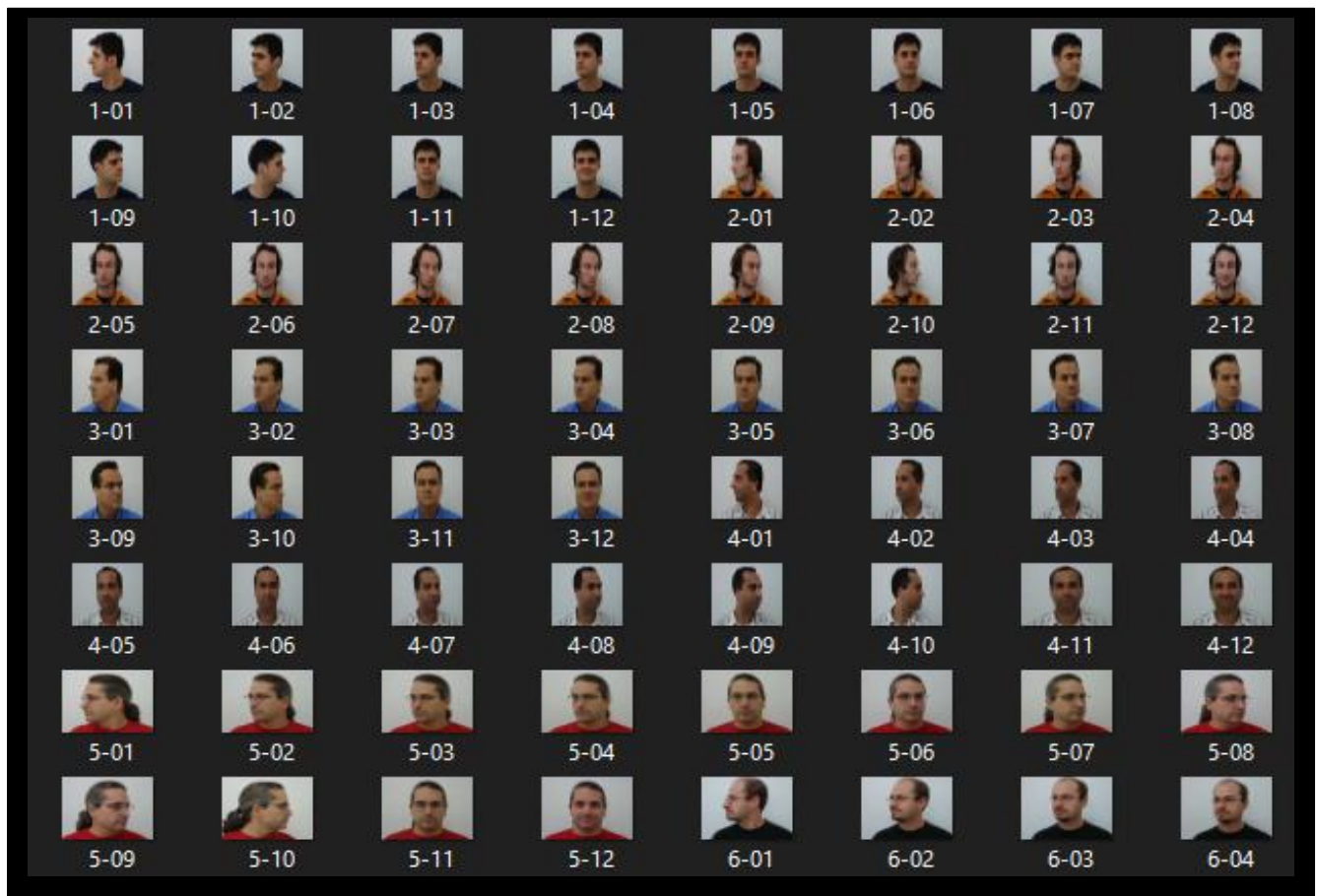
## III. Apprentissage sans couleurs

### III.1. Apprentissage

#### a. Création de la base d'apprentissage

La première étape dans notre projet consiste à construire une base de

données pour la reconnaissance, on dispose d'une collection d'images composée de 50 classes, où chaque classe contient 12 images, notre base d'apprentissage sera extraite de cette collection en prenant les images impaires, les images restantes serviront comme base de teste.

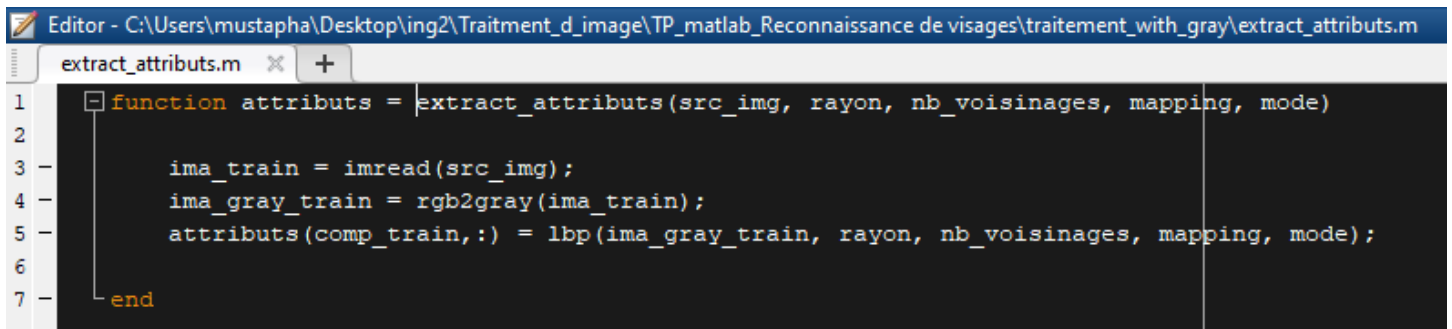


## **b. Extraction des attributs de texture**

Pour la suite du TP, on base notre reconnaissance sur les motifs locaux binaires, pour cela on utilise la fonction *lbp* afin d'attribuer chaque image à une classe.

L'utilisation de la fonction *lbp* permettait de calculer les motifs locaux binaires de nos images avec les paramètres suivants :

- Le nombre N de voisins à analyser et le rayon R du cercle sur lequel ces voisins se situent.
- Le paramètre « mapping » permettant l'accès aux variantes des motifs locaux binaires grâce à la fonction *getmapping*.
- Le paramètre « mode » qui prend les valeurs 'h' ou 'hist' pour obtenir l'histogramme des LBP, et la valeur 'nh' afin d'obtenir l'histogramme normalisé des LBP comme montre la figure suivante :



The screenshot shows a MATLAB script editor window titled 'Editor - C:\Users\mustapha\Desktop\ing2\Traitement\_d\_image\TP\_matlab\_Reconnaissance de visages\traitement\_with\_gray\extract\_attributs.m'. The script defines a function named 'extract\_attributs' with the following code:

```

1 function attributs = extract_attributs(src_img, rayon, nb_voisinages, mapping, mode)
2
3     ima_train = imread(src_img);
4     ima_gray_train = rgb2gray(ima_train);
5     attributs(comp_train,:) = lbp(ima_gray_train, rayon, nb_voisinages, mapping, mode);
6
7 end

```

## III.2. La classification

L'extraction des attributs de textures nous a permis de calculer un vecteur d'attributs à partir d'une image précise, et donc de mettre en place la procédure de « classification ». Cette dernière consiste en premier lieu de construire des classes à partir des images d'apprentissage dont les visages seront représentés par un ensemble d'attributs. Nous construisons alors un tableau « attributs » qui contient les attributs de chaque image de la base.

```

19 - for i=1:nb_classe * nb_image_par_class
20 -     if(mod(i,2) ~= 0)
21 -         num_classe(comp_train) = floor((i-1)/nb_image_par_class) + 1;
22 -         num_image = 1 + mod(i-1,12);
23 -         if(num_image < 10)
24 -             fichier_train = [chemin int2str(num_classe(comp_train)) '-0' int2str(num_image) '.jpg'];
25 -         else
26 -             fichier_train = [chemin int2str(num_classe(comp_train)) '-' int2str(num_image) '.jpg'];
27 -         end
28 -
29 -         attributs(comp_train,:) = extract_attributs(fichier_train, rayon, nb_voisinages, mapping, mode);
30 -
31 -         if(i < 599) %pour ne pas dépasser 300
32 -             comp_train = comp_train + 1;
33 -         end
34 -     end
35 - end

```

### III.3. Teste de modèle

Afin de tester notre solution, on prend cette fois-ci les images paires de la collection pour tester si notre algorithme est capable d'attribuer la bonne classe à chaque image.

On parcourt les images paires en extrayant pour chacune les motifs locaux binaires et en les comparant avec toutes les images de la base d'apprentissage dont le but est de trouver l'image qui ressemble le plus à l'image de teste.

```

for i=1:nb_classe * nb_image_par_class
    if(mod(i,2) == 0)
        num_classe_origin = floor((i-1)/nb_image_par_class) + 1;
        num_image = mod(i-1,12) + 1;
        if(num_image < 10)
            fichier_test = [chemin int2str(num_classe_origin) '-0' int2str(num_image) '.jpg'];
        else
            fichier_test = [chemin int2str(num_classe_origin) '-' int2str(num_image) '.jpg'];
        end

        given_image = rgb2gray(imread(fichier_test));
        given_image_gray_lbp= lbp(given_image, rayon, nb_voisinages, mapping, 'h');
    end
end

```

- ❖ La classe attribuée à chaque image est calculé à l'aide de l'algorithme de « plus proche voisin ».



## a. La méthode du plus proche voisin PPV :

L'objectif étant de trouver laquelle des images ressemblant au maximum à l'image testée, La méthode de plus proche voisin est basée sur la ressemblance des histogrammes.

Soit par l'intersection des histogrammes ou la distance de Manhattan.

### Par l'intersection des histogrammes

```
function [max_distance,returned_image] = ppv(given_image_gray_lbp, comp_train, attributs, nb_bins)
    S = zeros(comp_train,1);
    for i = 1 : comp_train
        S(i) = hist_intersection(given_image_gray_lbp, attributs(i,:), nb_bins);
    end
    [max_distance,returned_image] = max(S);
end

function distance = hist_intersection(vector_1, vector_2, length)
    distance_temp = 0;
    for i=1:length
        distance_temp = distance_temp + min(vector_1(i), vector_2(i));
    end
    distance = distance_temp;
end
```

### Par la distance de Manhattan

```
function [max_distance,returned_image] = ppv_manhattan(given_image_gray_lbp, comp_train, attributs, nb_bins)
    S = zeros(comp_train,1);
    for i = 1 : comp_train
        S(i) = manhattan(given_image_gray_lbp, attributs(i,:), nb_bins);
    end
    [max_distance,returned_image] = min(S);
end

function distance = manhattan(vector_1, vector_2, length)
    distance_temp = 0;
    for i=1:length
        distance_temp = distance_temp + abs(vector_1(i) - vector_2(i));
    end
    distance = distance_temp;
end
```

- ❖ D'après les tests, on a constaté que les deux méthodes donnent presque le même résultat. Ainsi, pour la suite du TP on a décidé d'utiliser la méthode de Manhattan pour déterminer le plus proche voisin.

### **b. Temps et taux de classification :**

Après le teste de toutes les images constituant la base de teste, on calcule le taux de classification par la relation suivante :

$$Taux = \frac{\text{Nombre des images bien classés}}{\text{Nombre totale des images testées}}$$

Le temps de classification est le temps écoulé pour classer les images.

Ces deux mesures dépendent des paramètres de la fonction lbp().

Dans l'intention de déterminer la bonne combinaison des paramètres donnant les meilleurs résultats au terme du taux et du temps de classification, on fait un jeu de paramètres.

*1<sup>er</sup> cas mapping=0*

On fixe le paramètre mapping à 0 et on fait varier le rayon R et le voisinage V

R	V	Taux	Temps
1	8	85.5333	65.3665
2	12	93.3363	85.8445
4	16	97.2336	213.1635

On constate que pour R=2 et V=12 on a une solution optimale au terme du

taux et du temps, pour cela on fixera pour la suite  $R = 2$  et  $V = 12$ .

2ème cas  $R=2, V=12$

Mapping	Taux	Temps
<b>0</b>	<b>93.6997</b>	<b>85.3336</b>
<b>Ri</b>	<b>74.4565</b>	<b>107.8558</b>
<b>U2</b>	<b>88.3336</b>	<b>106.6895</b>
<b>RiU2</b>	<b>60.5366</b>	<b>103.2995</b>

MAPPING = 0,  $R=2$  et  $V = 12$  est la combinaison qui donne les bons résultats.

## IV. Apprentissage avec couleurs

On suit la même procédure sauf qu'on extrait cette fois ci 3 LBP, Pour les composantes R, G et B des images en couleurs. Comme montre la figure suivante :

```
ima_train = rgb2lab(imread(fichier_train));
lbp_red = lbp(ima_train(:,:,1), rayon, nb_voisinages, mapping, 'h');
lbp_green = lbp(ima_train(:,:,2), rayon, nb_voisinages, mapping, 'h');
lbp_blue = lbp(ima_train(:,:,3), rayon, nb_voisinages, mapping, 'h');
lbp_vector = [lbp_red lbp_green lbp_blue];
attributs(comp_train,:) = lbp_vector;
```

On se basant sur les résultats de la section précédente (mapping=0, R=2, V=12), afin d'améliorer la précision de la reconnaissance on se projette sur différents espaces de couleur.

Espace couleur	Taux	Temps
RGB	95.0000	256.4666
HSV	95.6667	261.7050
YIQ	96.3333	278.8225
YCbCr	90.0000	245.4858
Lab	96.0000	309.7007

D'après le tableau l'espace qui nous permet d'augmenter le taux de classification en gagnant aussi au terme du temps par rapport aux autres espaces de couleur est **YIQ**.

Puisque on s'intéresse au visage il est pratique de rogner les images en sauvegardant les visages dans l'intention de construire une autre base pour l'apprentissage, ce qui permettra de réduire la taille des histogrammes.

## CascadeObjectDetector

Grace à cet outil on récupère les coordonnées des visages présents dans une image, en utilisant ces derniers, on fait appel à la fonction « imcrop » pour rogner les images, cette procédure a été effectué à l'aide du script suivant :

```
end
given_image = imread(fichier_test);
faceDetector = vision.CascadeObjectDetector;
bboxes = step(faceDetector, given_image);
if(size(bboxes,1) == 1)
    cropped_image = imcrop(given_image, bboxes);
else
    cropped_image = imcrop(given_image, bboxes(2,:));
end
chemin_cropped_image = [chemin_crop int2str(num_classe_origin) '-0' int2str(j) '.jpg'];
imwrite(cropped_image, chemin_cropped_image);
```

Sur la nouvelle base, on effectue le traitement précédent et on obtient le tableau ci-dessous.

Espace couleur	Taux	Temps
RGB	84.5000	30.7846
HSV	88.0000	30.9483
YIQ	90.5000	33.3502
YCbCr	87.5000	31.0502
Lab	86.0000	39.4841



Encore l'espace YIQ est le plus adapté car il est quatre fois plus rapide avec un taux de classification acceptable d'où l'intérêt de ne pas prendre en considération que de nécessaire.

## **V. Conclusion**

Ce TP nous a permis de découvrir le domaine de traitement d'images et surtout les technologies utilisées dans la détection des visages, et de confronter des problématiques pratiques dans ce domaine et de ne pas se limiter sur la théorie. Pendant la réalisation de ce projet on a surmonté plusieurs difficultés notamment le travail en groupe qui était difficile pendant la première séance et la mal gestion du temps.

Suite à ce travail, nous comptons faire des extensions à notre application. En effet on compte appliquer une autre approche pour améliorer la détection, il consiste à décomposer une image en plusieurs imageries de même taille.