Master AIV1 S7
TP 9 Segmentation supervisée par analyse d'histogramme

ludovic.macaire@univ-lille.fr

6 mars 2025

1 Binarisation par Règle de Bayes

L'objectif du TP est de segmenter une image en niveaux de gris par le seuillage supervisé des niveaux de gris via des macros python que vous allez concevoir.

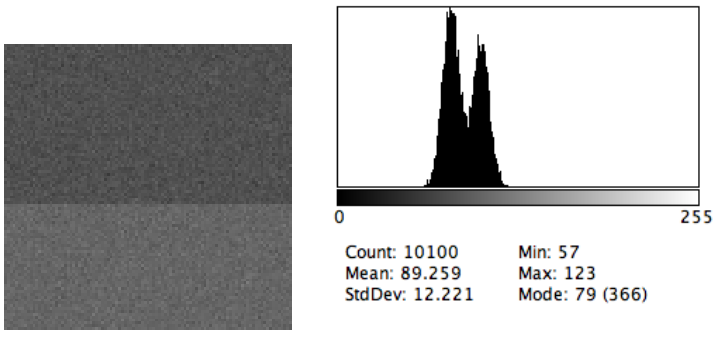
Vous devez rendre un fichier pdf décrivant les réponses aux questions et décrivant via les commentaires les macros python que vous avez développées.

Soit l'image I ($'2classes\_100\_100\_8bits.png'$) composée de 2 classes ω_1 et ω_2 de pixels.

Q1. Seuillage fixe

A partir du code ci-dessous, binariser l'image avec les seuils $\hat{\mathbf{X}}$ fixés aux valeurs 125, 135, 145.

```
1
2 ################
```



(a) Image (b) Histogramme

```
3
4  nom='./Fichiers_utiles_TP1/2classes_100_100_8bits_2016.png'
5  nomFenetre='original'
6  image = cv2.imread(nom, cv2.IMREAD_UNCHANGED)
7  # Affichage
8  cv2.namedWindow(nomFenetre)
9  cv2.imshow(nomFenetre, image)
10 # calcul de l'histogramme
11 hist = cv2.calcHist([image],[0],None,[256],[0,256])
12
13 #seuillage avec valeur de seuil
14 (retVal, ImgSeuil) = cv2.threshold(image, seuil, 255, cv2.THRESH_BINARY)
15
16 cv2.namedWindow('Resultat')
17 cv2.imshow('Image seuil', ImgSeuil)
18
19 cv2.waitKey(0)
20 cv2.destroyAllWindows()
21
22 ###############
```

Le seuil $\hat{\mathbf{X}}$ définit les 2 classes de pixels $\hat{\omega 1}$ et $\hat{\omega 2}$:

$$\hat{\omega 1} = \{P \in I | I(P) < \hat{\mathbf{X}}\}$$

$$\hat{\omega 2} = \{P \in I | I(P) \geq \hat{\mathbf{X}}\}$$

A quoi correspondent ces valeurs dans l'histogramme ?

Quel est le taux de bonne classification des pixels obtenu pour chacune de ces valeurs ? Pour ce faire, il faut calculer la matrice de confusion à partir de l'image $'2classes\_100\_100\_8bits\_GT.png'$.

Comme l'indique le listing ci-dessous, il faut transformer chaque image en vecteur 1D avant d'appeler la fonction confusion du package 'sklearn.metrics'. Les termes diagonaux de la matrice cm indiquent le nombre de pixels bien classés. On peut alors en déduire le taux de bonne classification des pixels en divisant le nombre de pixels bien classés par la taille de l'image.

```
1          from sklearn.metrics import confusion_matrix
2          GT = cv2.imread(nom_GT, cv2.IMREAD_UNCHANGED)
3          GT_1D = np.zeros((height*width,1),dtype=int)
4          for i in range(height):
5                  for j in range(width):
6                          val =GT[i,j]
7                          GT_1D[i*width + j] = val
8          cm = confusion_matrix(GT_1D, ImageSeuil_1D)
```

Q2. Seuillage automatique (Bayes)

Dans le dossier 'Q2-20225, l'image '$train\_bon\_omega1\_omega_2$' représente les deux classes ω_1 et ω_2 de l'image '$train\_bon$' qui vont vous être utiles pour le seuillage automatique supervisé. Dans la macro Python (mise en annexe dans le rapport)), écrire une boucle permettant de parcourir les seuils et de calculer la probabilité d'erreur $\varepsilon(t)$.

Il faudra alors estimer le minimum de $\varepsilon(t)$ et retenir le niveau correspondant.

Rappel de l'algorithme :

— Pour t compris entre 0 et 255, calculer
  — $\varepsilon(t) = \sum_{i=0}^{t} \frac{h_2(i)}{h(i)}.P(\omega_2) + \sum_{i=t+1}^{255} \frac{h_1(i)}{h(i)}.P(\omega_1)$
— le seuil optimal $\hat{t} = \arg\min(\varepsilon(t))$

Donner la valeur de seuil optimale et montrer l'image ainsi seuillée.

Q3. Détermination des images avec problème

Seuiller les autres images avec ce seuil et trouver une règle simple basée sur la population des classes qui permet de déterminer automatiquement si le nombre de trous est correct.

Q4. Classification en 3 classes de l'image

Identifier les 3 ROI correspondant aux 3 classes de l'image $'train\_3classes\_bis.bmp'$ du dossier 'Q4-2025'. La probabilité de bonne assignation à maximiser est donc

$P(\overline{\varepsilon}/\hat{\mathbf{X}}) = \sum_{k=1}^{K} P(\mathbf{X} \in \hat{\omega_k}, \omega_k)$

Rappel de l'alogorithme :

— Pour $t1$ et $t2$ compris entre 0 et 255, calculer
  — $B(t1,t2) = \sum_{i=0}^{t1} \frac{h_1(i)}{h(i)}.P(\omega_1) + \sum_{i=t1+1}^{t2} \frac{h_2(i)}{h(i)}.P(\omega_2) + \sum_{i=t2+1}^{255} \frac{h_3(i)}{h(i)}.P(\omega_3)$
— le seuil optimal $(\hat{t1}, \hat{t2}) = \arg\max(B(t1,t2))$

Ajouter la macro python en annexe, l'image segmentée (label 0, 127 et 255) et commenter les résultats. Pour ce faire, examiner la fonction cv2.threshold et combiner les résultats de 2 binarisations comme selon le listing exemple ci-dessous :

```
1 (retVal1, ImageSeuil1) = cv2.threshold(image, seuil1, 127, cv2.THRESH_BINARY)
2 (retVal2, ImageSeuil2) = cv2.threshold(image, seuil2, 255, cv2.THRESH_BINARY)
3 [dest] = np.maximum([ImageSeuil1],[ImageSeuil2])
```

Comme comme pour la question 2, ajouter les instructions pour calculer le taux de bonne classification des 3 classes, à partir de l'analyse de l'image verité terrain.

Classifier les autres images avec les 2 seuils identifiés.

Master Informatique - module AIV1 - S8
TP 10 Segmentation NON supervisée par analyse d'histogramme

ludovic.macaire@univ-lille.fr

19 mars 2025

Q1.Binarisation non supervisée par méthode d'OTSU

L'objectif du TP est de segmenter une image en niveaux de gris par le seuillage NON supervisé des niveaux de gris par des macros python que vous allez concevoir.

Vous devez rendre un fichier pdf décrivant les réponses aux questions et décrivant via les commentaires les macros python que vous avez développées. Certaines macros devront également être déposées sur moodle. Dans le dossier 'Q2-20225, l'image '$train\_bon\_omega1\_omega_2$' représente les deux classes ω_1 et ω_2 de l'image '$train\_bon$' qui vont vous être utiles pour le seuillage automatique NON supervisé. Soit l'image I ($'train\_bon'$) composée de 2 régions.

Dans la macro Python (mise en annexe dans le rapport)), écrire une boucle permettant de parcourir les seuils afin de retrouver la dispersion inter-classe la plus élevée.

Rappel de la méthode :

Pour t compris entre 0 et 255, calculer

— $\sigma(t) = P(\omega_1(t)).P(\omega_2(t)).(\mu(\omega_1(t)) - \mu(\omega_2(t)))^2$

le seuil optimal $\hat{t} = \arg\max(\sigma(t))$

avec

$$P(\omega_1(t)) = \sum_{i=0}^{t} \frac{h(i)}{N} \text{ et } N(\omega_1(t)) = \sum_{i=0}^{t} h(i)$$

$$P(\omega_2(t)) = 1 - P(\omega_1(t)) \text{ et } N(\omega_2(t)) = N - N(\omega_1(t))$$

$$\mu(\omega_1(t)) = \frac{\sum_{i=0}^{t} i.h(i)}{N(\omega_1(t))}$$

$$\mu(\omega_2(t)) = \frac{\sum_{i=t+1}^{255} i.h(i)}{N(\omega_2(t))}$$

Donner la valeur de seuil optimale et montrer l'image ainsi seuillée.

Quel est le taux de bonne classification obtenu (en se basant sur l'image '$train\_bon\_omega1\_omega_2$') ? Comparer ce taux avec celui obtenu lors du TP1 (seuillage supervisé).

Q2 Classification non supervisée en 3 classes de l'image par la méthode d'Otsu

Concevoir une macro python pour retrouver via la méthode d'Otsu les 3 classes de l'image $'train.bmp'$ du dossier 'Q4-2025'.

Ajouter la macro python en annexe, l'image segmentée (label 0, 127 et 255). Pour ce faire combiner les résultats de 2 binarisations comme indiqué dans le TP1.

Quel est le taux de bonne classification obtenu à partir de l'image $'train\_3classes\_bis.bmp'$ du dossier 'Q4-2025'. ? Comparer ce taux avec celui obtenu lors du TP1 (classification supervisée).

Q3 Segmentation d'une image couleur par binarisation des canaux

Vous allez segmenter l'image couleur 'image3d-v2025.bmp'. Pour ce faire, il faut

- séparer l'image couleur en 3 canaux grâce à cv2.split
- calculer l'histogramme de chaque canal (R, G et B)
- binariser de manière non supervisée chaque canal R, G, B via la méthode d'Otsu de la Question 1
- fusionner les canaux binarisés adéquats pour fournir une image de labels (où les deux disques concentriques sont fusionnés en une seule région) grâce à cv2.merge.

Combien de classes de pixels ont été identifiées dans l'image segmentée ? Comment sont segmentés les deux disques concentriques ?

Comment procéder pour extraire les deux disques concentriques ?

Master Informatique - module AIV1 - S8
TP 11 Analyse en composantes principales pour la segmentation d'images couleur

ludovic.macaire@univ-lille.fr

27 mars 2025

1 Q1 Analyse en composantes principales

Vous allez représenter l'image couleur 'image3d_v2025.bmp' selon ses 3 composantes principales.

1.1 Transformation en matrice de données

Pour ce faire, il faut convertir l'image dans une matrice de données $\mathbf{X}$ de dimension ($width.height \times 3$) , où $width.height$ représente le nombre de pixels de l'image et 3 le nombre de composantes couleur. N'hésitez pas à utiliser la fonction flatten (qui transforme un canal 2D en un vecteur 1D). Par exemple les instructions pour la composante R pourraient être :

```
R = image[:,:,0]
R_1d = R.flatten()
X = np.zeros((width.height,3),dtype=int)
X[:,0] = R_1d[:]
```

1.2 Fonction ACP

La matrice $\mathbf{X}$ sera passée en paramètre de la fonction ACP($\mathbf{X}$) que vous allez créer :

— Centrage à la moyenne des termes de la matrice :

```
X_meaned = X - np.mean(X , axis = 0)
```

— Calcul de la matrice $\mathbf{\Sigma}$ de co-variance des données $\mathbf{X}$:

```
cov_mat = np.cov(X_meaned , rowvar = False)
```

— Recherche et tri des 3 vecteurs propres $\mathbf{w}_i$ de $\mathbf{\Sigma}$: $\mathbf{\Sigma}.\mathbf{w}_i = \lambda_i.\mathbf{w}_i$ où $\lambda_i =$ sont les d valeurs propres les plus élevées et $i = 1, .., 3$.
— La matrice solution représentant la base est $\mathbf{W} = (\mathbf{w}_1, .., \mathbf{w}_3)$

```
eigen_values , eigen_vectors = np.linalg.eigh(cov_mat)
sorted_index = np.argsort(eigen_values)[::-1]
sorted_eigenvalue = eigen_values[sorted_index]
sorted_eigenvectors = eigen_vectors[:,sorted_index]
```

— Les données projetées sont alors $\mathbf{Y} = \mathbf{W}^T.\mathbf{X}$

```
Y_projected = np.dot(eigenvectorvectors.transpose() ,
 X_meaned.transpose() ).transpose()
```

1.3 Transformation de la matrice projetée vers 3 canaux

Soit XPCA la matrice de données projetée par ACP. Cette matrice doit maintenant être transformée en une image couleur RGB pour la binarisation de ses canaux via les étapes suivantes :

— Normalisation de XPCA entre 0 (la valeur la plus petite de XPCA) et 255 (la valeur la plus grande de XPCA)
— Conversion de XPCA en entier non signé

```
XPCA=XPCA.astype('uint8')
```

— Transformation de chaque composante de XPCA en un canal 2D (dans le listing ci-dessous C0PCA). On obtient ainsi 3 canaux C0PCA,C1PCA et C2PCA.

```
RPCA=np.reshape(XPCA[:,0],(height,width))
```

1.4 Binarisation des 3 canaux

Il faut enfin combiner la binarisation non supervisée via OTSU en appliquant les instructions développées pour répondre à la question Q3 du TP10 sur les 3 canaux C0PCA,C1PCA et C2PCA. Pour rappel, il faut

— calculer l'histogramme de chaque canal (C0PCA, C1PCA et C2PCA)
— binariser de manière non supervisée chaque canal C0PCA, C1PCA et C2PCA via la méthode d'Otsu
— fusionner les canaux binarisés adéquats pour fournir une image de labels grâce à cv2.merge.

Combien de classes de pixels ont été identifiées dans l'image pseudo-couleur segmentée ? Comment sont segmentés les deux disques concentriques ?

2 Q2 Application à 3 images

Appliquer l'algorithme ACP en retenant pour chaque cas les canaux adéquats pour identifier les chiffres par binarisation via OTSU dans les 3 images suivantes :

```
nom='./Fichiers_utiles_TP11/cas_4_dalton29.bmp'
nom='./Fichiers_utiles_TP11/cas_1_dalton26.bmp'
nom='./Fichiers_utiles_TP11/cas_2_dalton73.bmp'
```

Justifier pour chaque image le choix des composantes retenues et commenter les résultats de segmentation.

M1 RVA-AIV1 – Module VISA - TP 11 Segmentation d'une image couleur par classification non supervisée de pixels – version 2025

L'objectif du TP est de segmenter une image couleur par classification de pixels avec des macros Python que vous allez concevoir.

Le package nécessaire est le suivant: from sklearn.cluster import KMeans.

L'objectif de ce TP sera d'extraire des images les chiffres par segmentation et de comparer les résultats fournis par l'ACP avec ceux en RGB.

1 SEGMENTATION DES IMAGES par K-Means en RGB

Le programme devra

– 1.1. Segmenter l'image 'cas_2_dalton7.bmp' par K-Means. **Attention le nombre de classes est à fixer avec attention, tout en étant le plus faible possible**.

```
# n : nombre de classes
k_means = KMeans(n_clusters=n)
# a : vecteur de dimension nbpixels x 3
k_means.fit(a)
# centroids : vecteur de dimension n x 3
centroids = k_means.cluster_centers_
# labels: vecteur de dimension nbpixels x 1
labels = k_means.labels_
# labels: vecteur de dimension nbpixels x 3
a2 = centroids[labels]
```

– 1.2. Segmenter automatiquement les autres images du 'cas_2' (sélection du nom de l'image par l'utilisateur) à l'aide des centroids identifiés en 1.1 grâce à la fonction

```
test_labels = k_means.predict(test_color_image_vector)
– test_centroid_vector = centroids[test_labels].
```

– Rendre la macro commentée sur moodle avec comme paramètre d'entrée l'autre image cas_2 à analyser.

2 COMPARAISON DES SEGMENTATIONS DES IMAGES 'cas_1_dalton42.bmp' et 'cas_2_dalton73.bmp' par K-Means via RGB ou ACP.

Appliquer les segmentations sur les images codées en RGB ou ACP et comparer les résultats (en prenant le même nombre de classes).