***Compte Rendu sur Travail Pratique 3***

**Ce TP a pour but d'explorer différentes méthodes d'extraction de caractéristiques visuelles utilisées en vision par ordinateur :**

* **SIFT (Scale Invariant Feature Transform)** pour détecter et décrire les points d'intérêt d'une image.
* **HOG (Histogram of Oriented Gradients)** pour extraire les caractéristiques basées sur les gradients.
* **LBP (Local Binary Patterns)** pour analyser les textures d'une image.

**Activité 1 : Extraction des points clés avec SIFT**

**Détection des points clés et affichage des descripteurs SIFT**

import cv2

import numpy as np

import matplotlib.pyplot as plt

from PIL import Image

from skimage.transform import rotate, rescale

# Chargement de l'image originale

img\_originale = Image.open('../images/Lenna.png')

# Conversion en niveaux de gris

img = np.array(img\_originale.convert('L'))

# Création de l'objet SIFT

sift = cv2.SIFT\_create()

# Détection des points clés et calcul des descripteurs

keypoints, descriptors = sift.detectAndCompute(img, None)

# Dessin des points clés sur l'image

img\_sift = cv2.drawKeypoints(img, keypoints, None, flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS)

# Affichage du nombre de points clés détectés

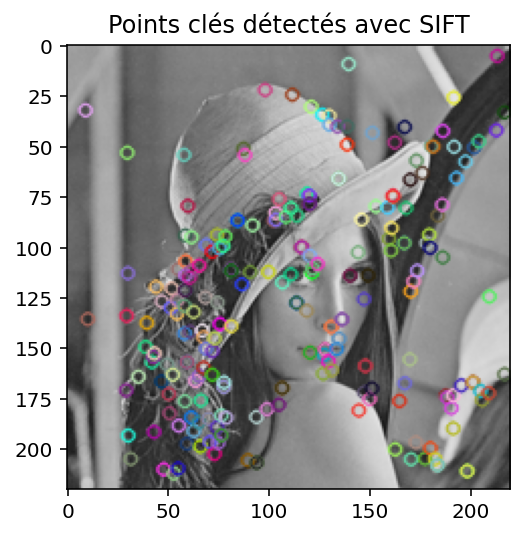
print("Nombre de points clés (original) :", len(keypoints))

# Affichage de l'image avec les points clés

plt.imshow(img\_sift, cmap='gray')

plt.title("Points clés détectés avec SIFT")

plt.show()



**Combien de points clés ont été détectés ?**

Nombre de points clés (original) : 238

**Appliquez différentes transformations rotation 45 degrés, translation de 20 pixels en x et 30 pixels en y et mise à l'échelle à 1.5x à l'image Lenna.png.**

**img\_rotate = rotate(img, angle=45, resize=True)**

**matrice\_trans = np.float32([[1, 0, 20], [0, 1, 30]])**

**img\_translate = cv2.warpAffine(img\_rotate, matrice\_trans, (img\_rotate.shape[1], img\_rotate.shape[0]))**

**img\_transform = rescale(img\_translate, scale=1.5, anti\_aliasing=True)**

**img\_transform = (img\_transform \* 255).astype(np.uint8) keypoints2, descriptors2 = sift.detectAndCompute(img\_transform, None)**

**# Affichage du nombre de points clés détectés après transformation**

**print("Nombre de points clés (après transformation) :", len(keypoints2))**

**# Dessin des points clés après transformation**

**img\_sift1 = cv2.drawKeypoints(img\_transform, keypoints2, None, flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS)**

**# Affichage de l'image transformée avec les points clés**

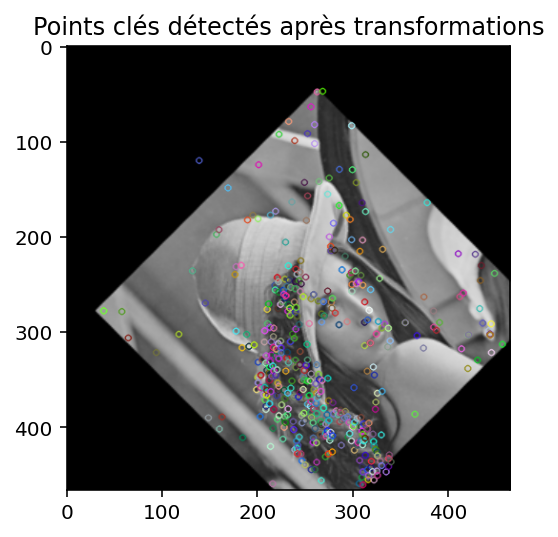
**plt.imshow(img\_sift1, cmap='gray')**

**plt.title("Points clés détectés après transformations")**

**plt.show()**

**Avoir les impacts sur nombre des points clés détectés.**

Nombre de points clés (après transformation) : 473



On voit qu’il y a un petit changement sur certains point intérêt dans image obtenus après transformation.

**On Passe pour comparer les descripteurs :**

bf = cv2.BFMatcher(cv2.NORM\_L2, crossCheck=True)

matches = bf.match(descriptors, descriptors2)

matches = sorted(matches, key=lambda x: x.distance)

# Dessin des correspondances

img\_matches = cv2.drawMatches(img, keypoints, img\_transform, keypoints2, matches[:50], None, flags=cv2.DrawMatchesFlags\_NOT\_DRAW\_SINGLE\_POINTS)

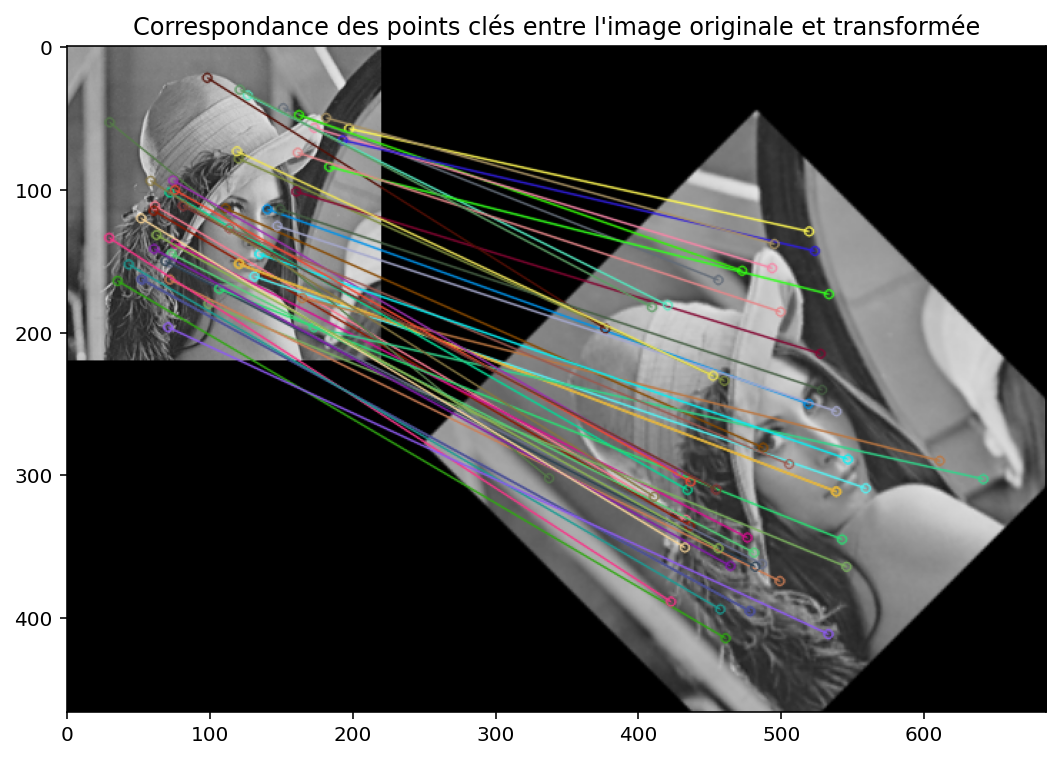
# Affichage des correspondances

plt.figure(figsize=(12, 6))

plt.imshow(img\_matches, cmap='gray')

plt.title("Correspondance des points clés entre l'image originale et transformée")

plt.show()



**Activité 2: Descripteur HOG (Histogram of Oriented Gradients):**

import numpy as np

from skimage.feature import hog

from skimage.transform import resize

from skimage import exposure

from PIL import Image

import matplotlib.pyplot as plt

# ouvrir image et convertir en niveaux de gris

img = Image.open('../images/lenna.png')

# generer la matrice de image

img = np.array(img)

# redimentionner les images

img\_resize = resize(img,(128\*4,64\*4))

# application de calcul descripteur hog

fd , img\_hog = hog(img\_resize, orientations = 16 , pixels\_per\_cell = (8,8) ,cells\_per\_block=(10,2), visualize=True,channel\_axis = 2)

# Amélioration de la visibilité du HOG

hog\_image = exposure.rescale\_intensity(img\_hog, out\_range=(0, 255))

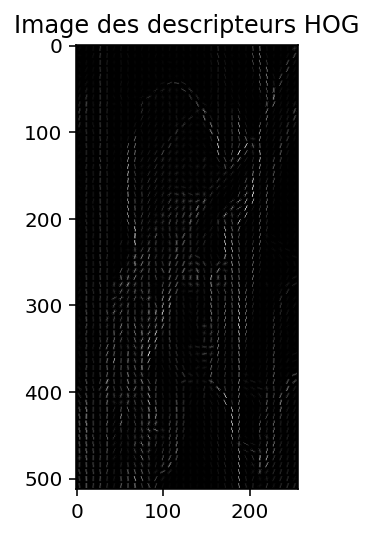
# print(fd.shape)

# Affichage

plt.imshow(hog\_image, cmap='gray')

plt.title("Image des descripteurs HOG")

plt.show()



**Response a les questions:**

* fd : taille de descripteur des images obtenus.
* img\_hog est image obtenus apres calcul descripteur hog
* impact parametres channel\_axis representer les couleurs de image soit RGB ou niveaux de gris ,mais dans cette cas on peut ignorer.
* Nombre de directions ou orientation utiliser pour quantifier les gradients de chaque cellule.

**Activité 3: Descripteur LBP (Local Binary Patterns)**

* **Extraction du descripteur LBP et affichage**

import numpy as np

import matplotlib.pyplot as plt

from skimage.feature import local\_binary\_pattern

from PIL import Image

img = np.array(Image.open('../images/Lenna.png').convert('L'))

# Calcul du LBP

lbp = local\_binary\_pattern(img, P=8, R=1)

# Affichage de l'image LBP

plt.imshow(lbp, cmap='gray')

plt.title("Image LBP")

plt.show()

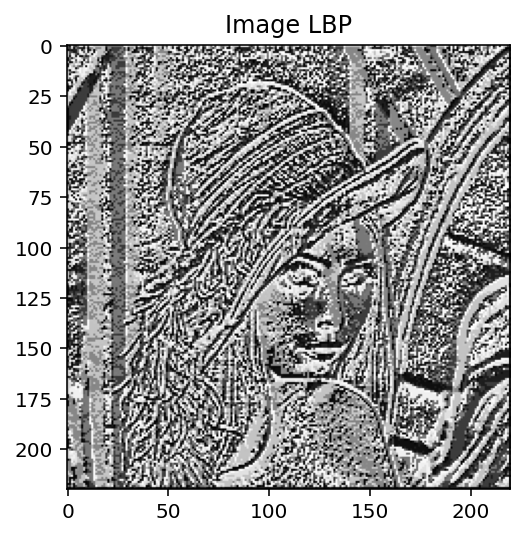
# Affichage de l'histogramme

plt.hist(lbp.ravel(), bins=256, range=(0, 256), color='gray')

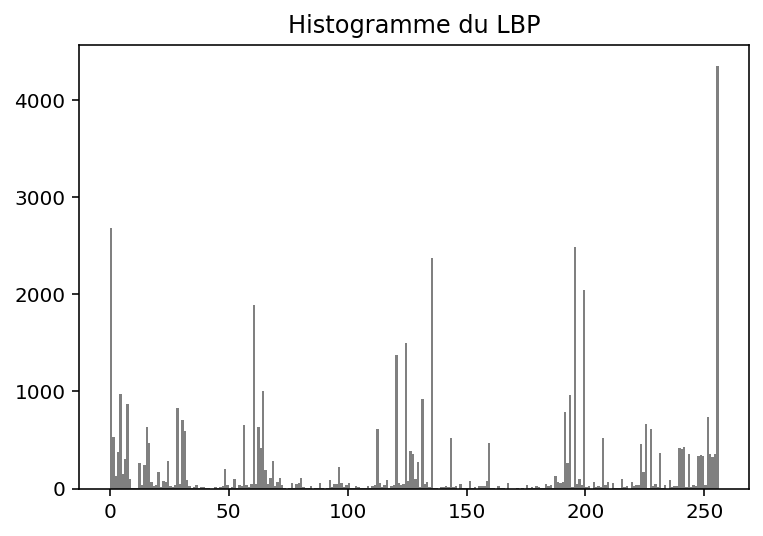
plt.title("Histogramme du LBP")

plt.show()

**Affichage image LBP.**



**Affichage de Histogramme de descripteur LBP.**



* **Segmentation en blocs et calcul des descripteurs**

**Divisez Lenna.png en blocs de 16 × 16 pixels.**

**block\_size = (16, 16)**

**# Ajuster la taille de l'image pour être multiple de 16**

**height, width = img.shape**

**new\_height = height - (height % 16)**

**new\_width = width - (width % 16)**

**img\_cropped = img[:new\_height, :new\_width]**

**# Diviser l'image en blocs de 16x16**

**blocks = view\_as\_blocks(img\_cropped, block\_size)**

**num\_blocks\_x, num\_blocks\_y = blocks.shape[:2]**

**Extraction des descripteurs LBP pour chaque bloc.**

**lbp\_descriptors = []**

**for i in range(num\_blocks\_x):**

**for j in range(num\_blocks\_y):**

**block = blocks[i, j]**

**lbp = local\_binary\_pattern(block, P, R, method="uniform")**

**hist, \_ = np.histogram(lbp.ravel(), bins=np.arange(0, P + 3), density=True)**

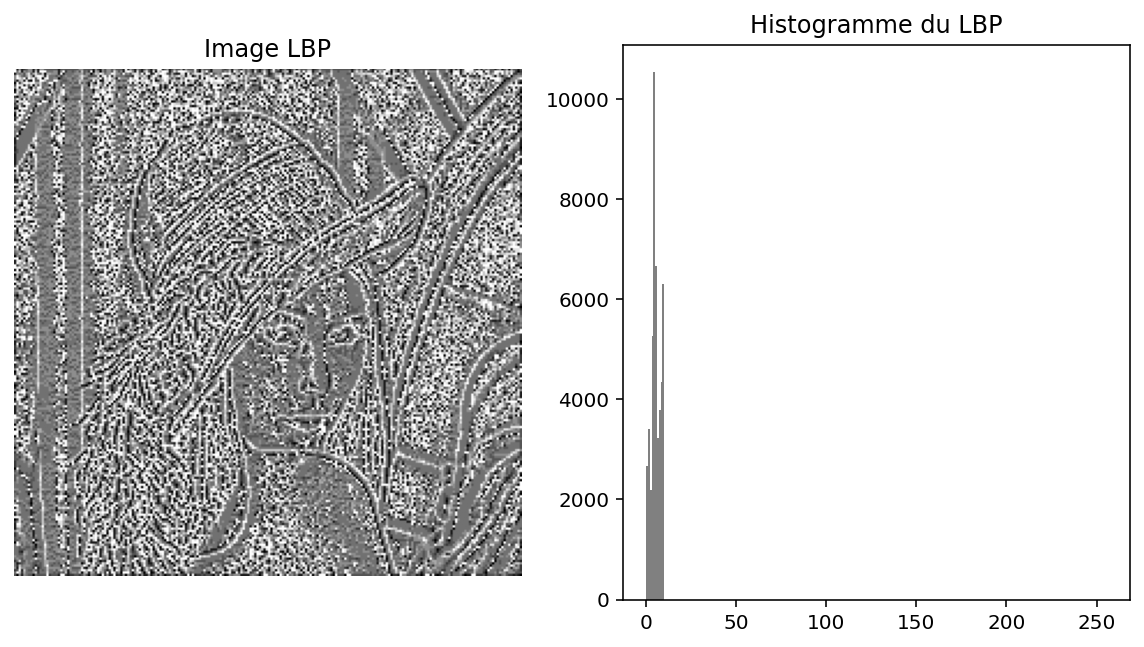
**lbp\_descriptors.append(hist)**

**Concaténation de tous les descripteurs.**

**final\_descriptor = np.concatenate(lbp\_descriptors)**

**print(f"Taille du descripteur final après concaténation : {final\_descriptor.shape[0]}")**

Taille du descripteur final après concaténation : 1690



**Comparez SIFT, HOG et LBP : quelles sont leurs différences ?**

**SIFT (Scale-Invariant Feature Transform) :** Détecte des points clés distinctifs et les décrit avec des gradients locaux, Invariant aux transformations d'échelle, rotation, illumination et bruit.

**HOG (Histogram of Oriented Gradients) :** Décrit la distribution des gradients en calculant des histogrammes locaux de gradients, Partiellement invariant aux transformations d'illumination et de rotation.

**LBP (Local Binary Pattern) :** Encode la texture en comparant l'intensité des pixels voisins d'un pixel central, Robuste aux variations d'illumination mais sensible aux rotations et déformations.

**Quel descripteur choisiriez-vous pour la détection de visages ? Pourquoi ?**

Pour la détection de visages en temps réel, HOG ou LBP sont préférables. Pour la reconnaissance (identification), SIFT peut être plus précis.

**Quel descripteur est le plus robuste aux transformations ?**

**SIFT** est le plus robuste, mais **LBP et HOG sont plus rapides** et adaptés aux tâches temps réel comme la détection de visages.

**Conclusion :**

* **Détection faciale** : **LBP** (rapide) ou **HOG** (avec SVM).
* **Reconnaissance faciale** : **SIFT** (plus précis mais lent).
* **Robustesse aux transformations** : **SIFT** > **HOG** > **LBP**.