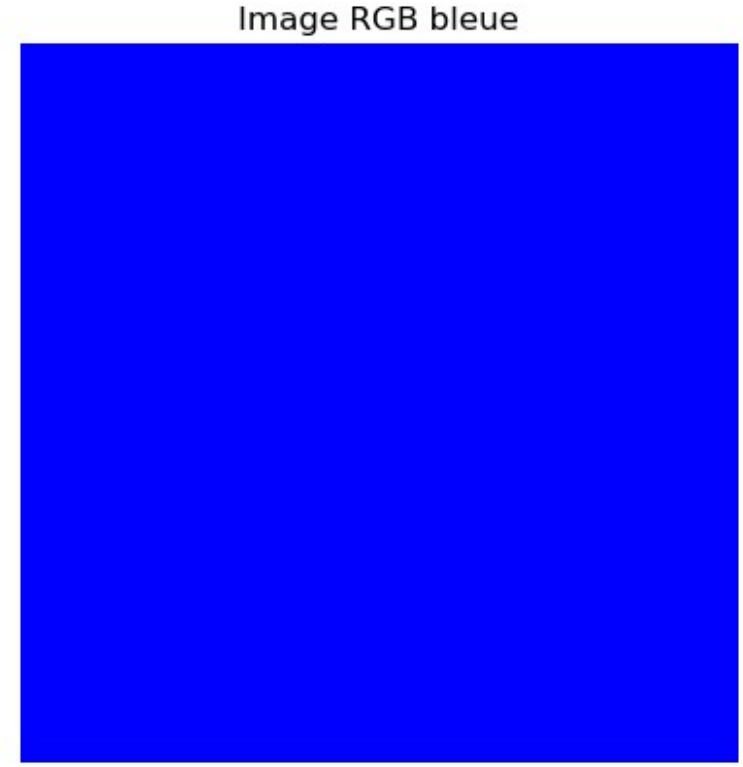
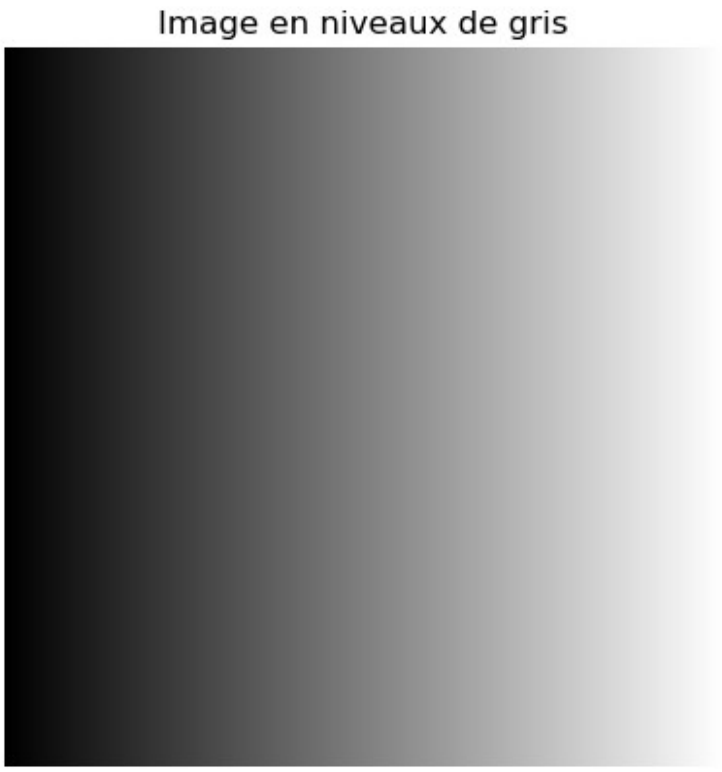
```python
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Image RGB bleue
img_rgb = np.zeros((256, 256, 3), dtype=np.uint8)
img_rgb[:] = (255, 0, 0)  # Bleu en BGR (OpenCV)
plt.imshow(cv2.cvtColor(img_rgb, cv2.COLOR_BGR2RGB))
plt.title("Image RGB bleue")
plt.axis('off')
plt.show()
```



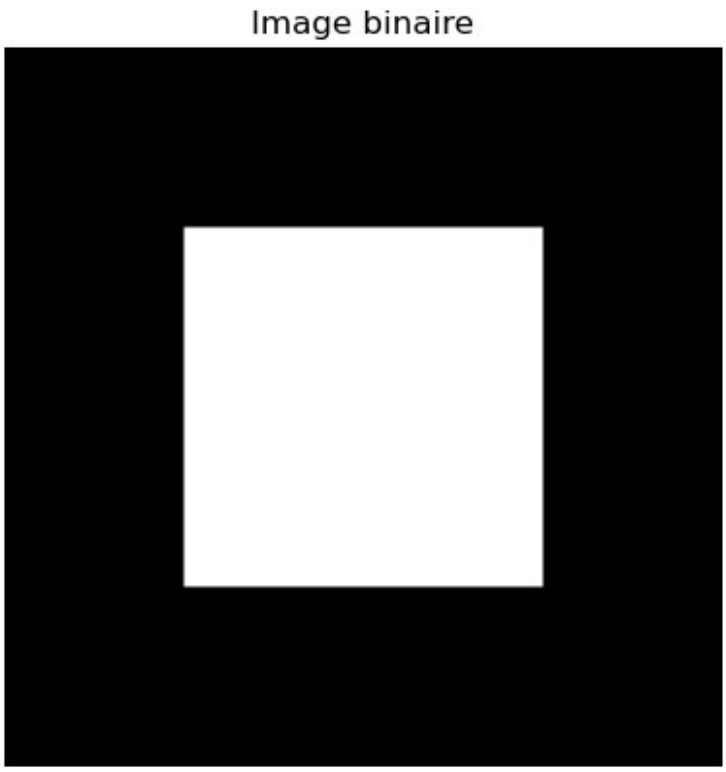

```python
#### b) Image en niveaux de gris

# Dégradé horizontal
img_gray = np.tile(np.arange(256, dtype=np.uint8), (256, 1))
plt.imshow(img_gray, cmap='gray')
plt.title("Image en niveaux de gris")
plt.axis('off')
plt.show()
```




```
#### c) Image binaire

# Carré blanc sur fond noir
img_bin = np.zeros((256, 256), dtype=np.uint8)
img_bin[64:192, 64:192] = 255
plt.imshow(img_bin, cmap='gray')
plt.title("Image binaire")
plt.axis('off')
plt.show()
```




```
## Partie 2 : Analyse d’Images Réelles

# Dans cette partie, nous allons charger une image réelle, afficher
ses caractéristiques, analyser son histogramme, la normaliser,
modifier certains pixels, puis appliquer différentes transformations
classiques du traitement d’image.

import cv2
import numpy as np
import matplotlib.pyplot as plt

# Charger une image du dataset (adapter le chemin si besoin)
img = cv2.imread('../1.jpg')  # Remplace par le chemin réel de ton
image
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title("Image du dataset")
plt.axis('off')
plt.show()
```

Image du dataset

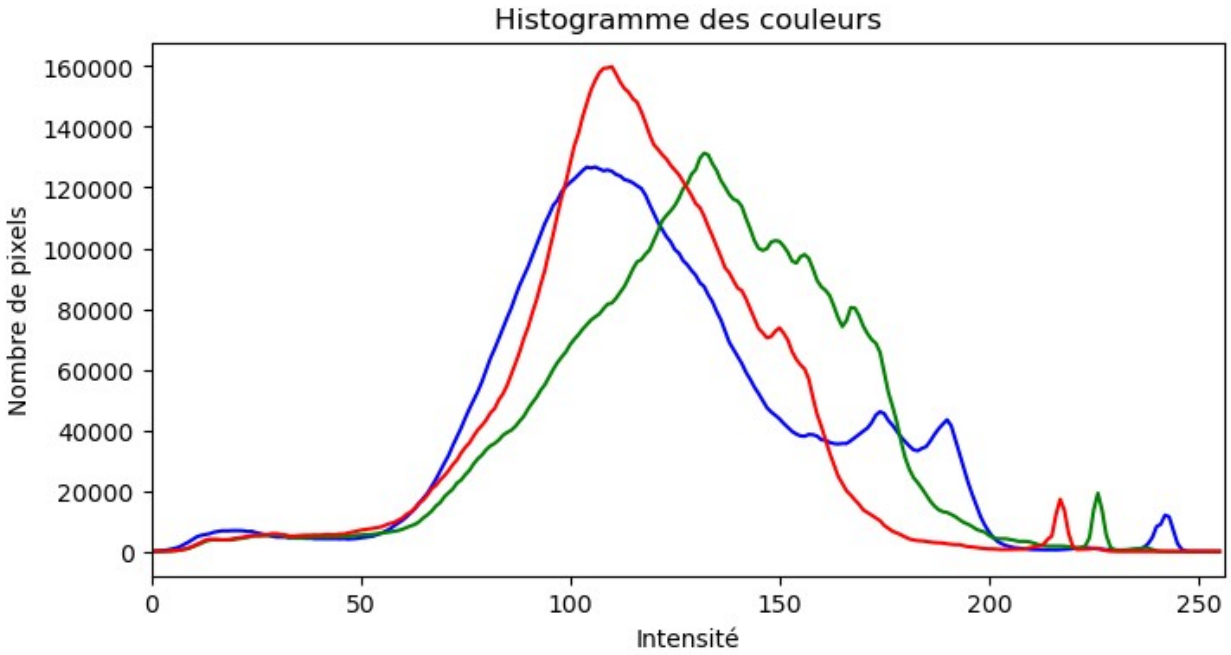
```
### 2.1 Caractéristiques de l'image
# Affichons les dimensions, le type et la taille de l'image.

print(f"Dimensions : {img.shape}")
print(f"Type : {img.dtype}")
print(f"Taille (nombre de pixels) : {img.size}")
```

```
Dimensions : (3072, 3072, 3)
Type : uint8
Taille (nombre de pixels) : 28311552
```

```
### 2.2 Histogramme des couleurs
# Visualisons la répartition des intensités pour chaque canal de
couleur.

colors = ('b', 'g', 'r')
plt.figure(figsize=(8,4))
for i, col in enumerate(colors):
    hist = cv2.calcHist([img], [i], None, [256], [0, 256])
    plt.plot(hist, color=col)
    plt.xlim([0, 256])
plt.title('Histogramme des couleurs')
plt.xlabel('Intensité')
plt.ylabel('Nombre de pixels')
plt.show()
```




```
### 2.3 Normalisation de l'image
# On ramène les valeurs de pixels entre 0 et 1.

img_norm = img.astype(np.float32) / 255.0
print(f"Min: {img_norm.min()}, Max: {img_norm.max()}")
```

```
Min: 0.0, Max: 1.0
```

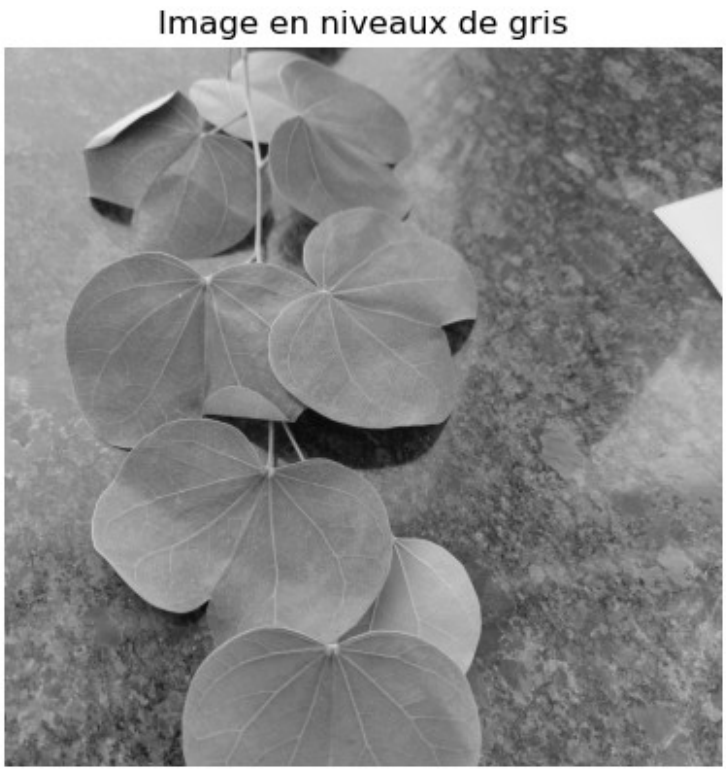
```
### 2.4 Modification d'un ensemble de pixels
# On colore un carré rouge en haut à gauche de l'image.

img_mod = img.copy()
img_mod[0:50, 0:50] = [0, 0, 255]  # Rouge en BGR
plt.imshow(cv2.cvtColor(img_mod, cv2.COLOR_BGR2RGB))
plt.title("Carré rouge ajouté")
plt.axis('off')
plt.show()
```


```
### 2.5 Transformations classiques

# Nous allons maintenant appliquer différentes transformations :
# - Conversion en niveaux de gris
# - Seuillage et détection de contours
# - Débruitage (filtres)
# - Opérations morphologiques
# - Compression

# Conversion en niveaux de gris
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(img_gray, cmap='gray')
plt.title("Image en niveaux de gris")
plt.axis('off')
plt.show()
```

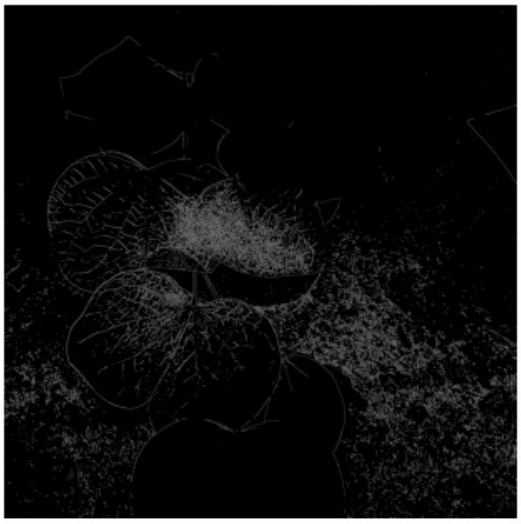



```
# Seuillage et détection de contours
_, img_thresh = cv2.threshold(img_gray, 127, 255, cv2.THRESH_BINARY)
edges = cv2.Canny(img_gray, 100, 200)

plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.imshow(img_thresh, cmap='gray')
plt.title("Seuillage binaire")
plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(edges, cmap='gray')
plt.title("Contours (Canny)")
plt.axis('off')
plt.show()
```

Seuillage binaire

Contours (Canny)



```
# Débruitage avec filtres
img_blur = cv2.GaussianBlur(img, (7, 7), 0)
img_median = cv2.medianBlur(img, 5)

plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.imshow(cv2.cvtColor(img_blur, cv2.COLOR_BGR2RGB))
plt.title("Flou gaussien")
plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(cv2.cvtColor(img_median, cv2.COLOR_BGR2RGB))
plt.title("Filtre médian")
plt.axis('off')
plt.show()
```

Flou gaussien

Filtre médian

```
# Opérations morphologiques
kernel = np.ones((5, 5), np.uint8)
img_dilate = cv2.dilate(img_thresh, kernel, iterations=1)
img_erode = cv2.erode(img_thresh, kernel, iterations=1)

plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.imshow(img_dilate, cmap='gray')
plt.title("Dilatation")
plt.axis('off')
plt.subplot(1,2,2)
plt.imshow(img_erode, cmap='gray')
plt.title("Erosion")
plt.axis('off')
plt.show()
```


```
# Compression avec/sans perte
cv2.imwrite('image_jpeg_qualite90.jpg', img,
[int(cv2.IMWRITE_JPEG_QUALITY), 90])  # Avec perte
cv2.imwrite('image_png_sans_perte.png', img,
[int(cv2.IMWRITE_PNG_COMPRESSION), 0])  # Sans perte
```

```
True
```

```
## Partie 2 : Classification d'Images avec Machine Learning (Scikit-
Learn)

# Dans cette partie, nous allons :
# - Extraire des caractéristiques simples à partir des images (par
exemple, histogrammes de couleurs ou de niveaux de gris)
# - Construire un jeu de données (features + labels)
```

```
# - Entraîner un classifieur (par exemple, SVM ou RandomForest)
# - Évaluer les performances du modèle

import os
import cv2
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Paramètres
data_dir = "300_dataset"  # adapte ce chemin à ton dataset
samples_per_class = 10    # nombre d'images par classe

# Extraction des features : histogramme de niveaux de gris
def extract_features_and_labels(data_dir, samples_per_class=10):
    classes = [d for d in os.listdir(data_dir) if
os.path.isdir(os.path.join(data_dir, d))]
    X, y = [], []
    for label, class_name in enumerate(classes):
        class_path = os.path.join(data_dir, class_name)
        images = os.listdir(class_path)[:samples_per_class]
        for img_name in images:
            img_path = os.path.join(class_path, img_name)
            img = cv2.imread(img_path)
            if img is None:
                continue
            img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            hist = cv2.calcHist([img_gray], [0], None, [64],
[0,256]).flatten()
            hist = hist / hist.sum()  # Normalisation
            X.append(hist)
            y.append(label)
    return np.array(X), np.array(y), classes

X, y, class_names = extract_features_and_labels(data_dir,
samples_per_class)
print(f"Nombre d'images : {len(X)}, Nombre de classes :
{len(class_names)}")
```

```
Nombre d'images : 100, Nombre de classes : 10
```

```
### Séparation du jeu de données et apprentissage

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42, stratify=y)

clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

```
### Évaluation du modèle

# Nous affichons le rapport de classification et la matrice de
confusion pour évaluer les performances du modèle.

print(classification_report(y_test, y_pred, target_names=class_names))
print("Matrice de confusion :")
confusion_matrix(y_test, y_pred)
```

```
                    precision    recall  f1-score   support

            Nilgiri       0.00      0.00      0.00         3
          Sonmohar       0.14      0.33      0.20         3
            Pimpal       0.75      1.00      0.86         3
            Kashid       0.33      0.67      0.44         3
            Karanj       0.25      0.33      0.29         3
Indian Rubber Tree       0.00      0.00      0.00         3
   Vilayati Chinch       0.00      0.00      0.00         3
              Apta       0.00      0.00      0.00         3
               Vad       1.00      0.33      0.50         3
        Sita Ashok       0.00      0.00      0.00         3

          accuracy                           0.27        30
         macro avg       0.25      0.27      0.23        30
      weighted avg       0.25      0.27      0.23        30

Matrice de confusion :

/home/ayoub/.local/lib/python3.13/site-packages/sklearn/metrics/
_classification.py:1565: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/home/ayoub/.local/lib/python3.13/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined
and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))
/home/ayoub/.local/lib/python3.13/site-packages/sklearn/metrics/_class
ification.py:1565: UndefinedMetricWarning: Precision is ill-defined
and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is",
len(result))

array([[0, 1, 0, 0, 1, 0, 0, 1, 0, 0],
       [0, 1, 0, 1, 1, 0, 0, 0, 0, 0],
       [0, 0, 3, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 2, 0, 0, 0, 0, 0, 0],
```

```
       [0, 0, 0, 2, 1, 0, 0, 0, 0, 0],
       [0, 1, 0, 1, 0, 0, 0, 1, 0, 0],
       [0, 1, 0, 0, 1, 1, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0, 1, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 1, 0, 1, 1],
       [0, 2, 0, 0, 0, 0, 0, 1, 0, 0]])

## Partie 4 : Réseaux de Neurones Simples (MLP)

# Dans cette partie, nous allons :
# - Utiliser un perceptron multicouche (MLP) pour la classification d'images,
# - Comparer ses performances à celles du RandomForest,
# - Discuter des avantages/inconvénients.

# Nous utiliserons les mêmes features (histogrammes de niveaux de gris) pour comparer équitablement.
```