INDIRA GANDHI DELHI TECHNICAL UNIVERSITY

COMPUTER VISION ASSIGNMENT 2

Name : Siya Pathak
Enrollment No:06601032021
Course : BTech
Branch :IT-1

Image Processing using Python

Aim:
To get familiar with image handling and basic image analysis using Python.
To implement basic image feature extraction methods.
Q1: Image Reading:

```python
import cv2

image_path = '/content/drive/MyDrive/cv_image.jpeg'
image = cv2.imread(image_path)
print(image)
```

```
[[[16 13 22]
  [17 14 23]
  [18 15 24]
  ...
  [21 15 38]
  [21 15 38]
  [21 15 38]]

 [[16 13 22]
  [17 14 23]
  [18 15 24]
  ...
  [21 15 38]
  [21 15 38]
  [21 15 38]]

 [[16 13 22]
  [17 14 23]
  [18 15 24]
  ...
  [21 15 38]
  [21 15 38]
  [21 15 38]]

 ...

 [[14 11 26]
  [14 11 26]
  [14 11 26]
  ...
  [19 12 39]
  [19 12 39]
```

Q2: Image Collection:

```
[2] import os
    import cv2
    import random
```

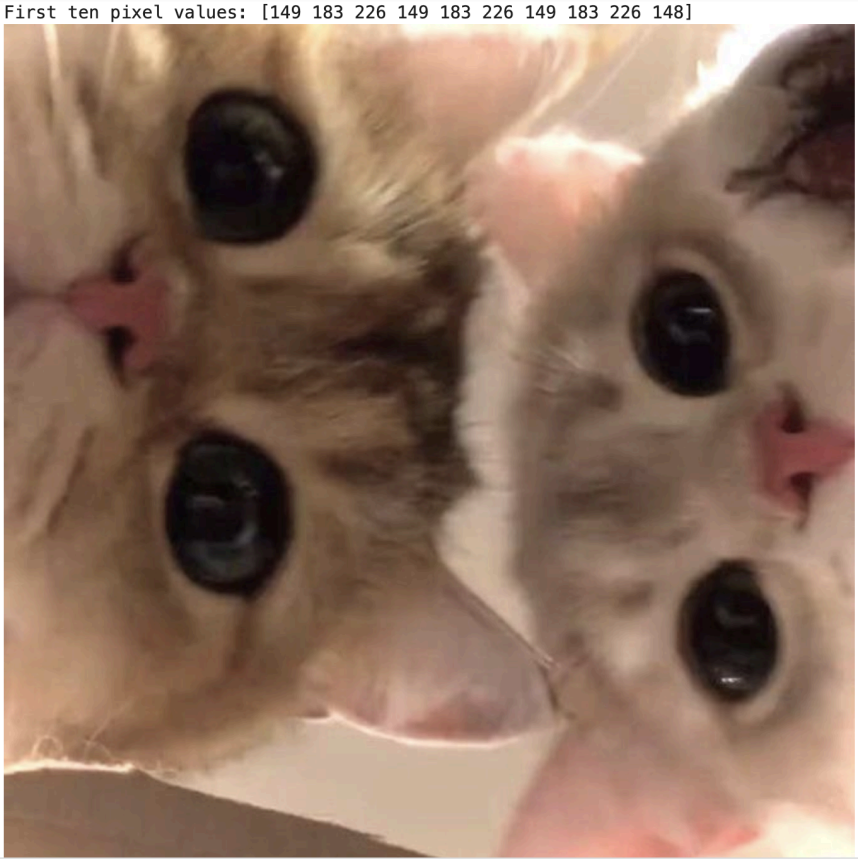
```
path="/content/drive/MyDrive/cv"
data=[]
for filename in os.listdir(path):
  if filename.endswith("jpeg"):
    img = cv2.imread("/content/drive/MyDrive/cv/"+filename)
    data.append(img)
    if img is not None:
          print("Resolution of " + filename + ": ", img.shape[1], "x", img.shape[0])
    else:
          print("Unable to read", filename)
    dimensions = img.ndim
    print("Number of dimensions:", dimensions)
    rand_x = random.randint(0, img.shape[1] - 1)
    rand_y = random.randint(0, img.shape[0] - 1)
    pixel_values = img[rand_y, rand_x]
    print("Pixel values at (", rand_x, ",", rand_y, "):", pixel_values)
```

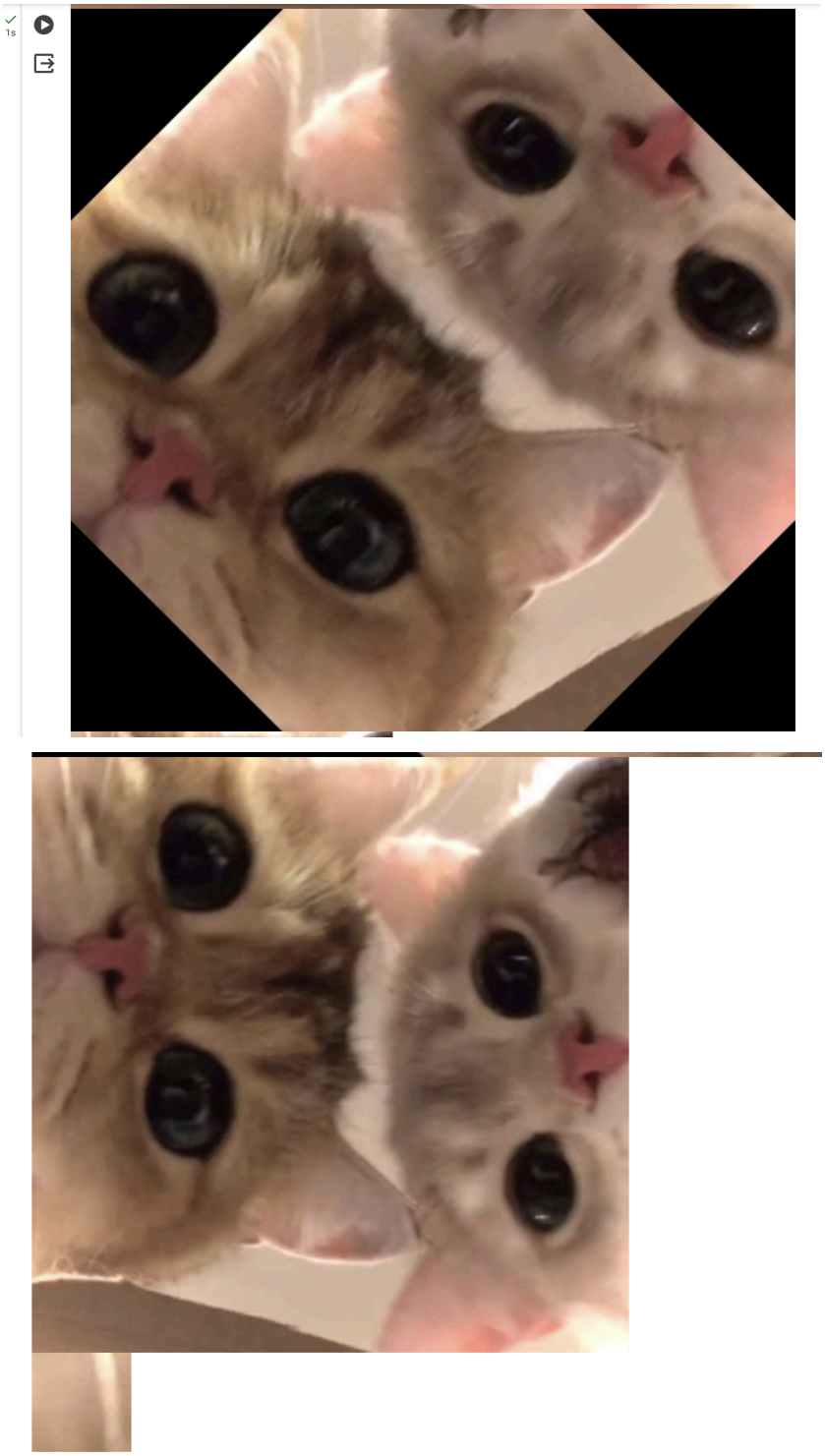
```
Resolution of image_2.jpeg:  675 x 675
Number of dimensions: 3
Pixel values at ( 375 , 482 ): [103 117 153]
Resolution of image_3.jpeg:  736 x 736
Number of dimensions: 3
Pixel values at ( 380 , 562 ): [192 176 199]
Resolution of image_4.jpeg:  736 x 736
Number of dimensions: 3
Pixel values at ( 440 , 385 ): [162 174 180]
Resolution of image_1.jpeg:  694 x 1024
Number of dimensions: 3
Pixel values at ( 64 , 188 ): [138 138 138]
Resolution of image_5.jpeg:  238 x 460
Number of dimensions: 3
Pixel values at ( 66 , 387 ): [158 158 158]
```

Q3: Image Manipulation:

```
from google.colab.patches import cv2_imshow
random_img = random.choice(data)
# Rotation by 45 degrees
rows, cols, _ = random_img.shape
M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 1)
rotated_img = cv2.warpAffine(random_img, M, (cols, rows))

# Resize the image to a predefined resolution (e.g., 300x300)
resized_img = cv2.resize(random_img, (300, 300))

# Flatten the image and print the first ten pixel values
flattened_img = random_img.flatten()
print("First ten pixel values:", flattened_img[:10])

# Select a small region in the image (e.g., top-left corner with dimensions 50x50)
small_region = random_img[:50, :50]

# Display the images (optional)
cv2_imshow(random_img)
cv2_imshow(rotated_img)
cv2_imshow(resized_img)
cv2_imshow(small_region)
```

```
First ten pixel values: [149 183 226 149 183 226 149 183 226 148]
```



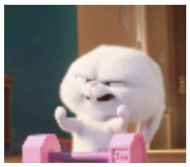


Q4: Basic Image Handling:

```python
import numpy as np

original_folder = "/content/drive/MyDrive/cv"
new_folder = "/content/drive/MyDrive/copy_cv"
#make directory
os.makedirs(new_folder, exist_ok=True)

original_images = os.listdir(original_folder)
#copy images
for img_name in original_images:
    img_path = os.path.join(original_folder, img_name)
    new_img_path = os.path.join(new_folder, img_name)
    os.system(f"cp '{img_path}' '{new_img_path}'")
#oprations
for img_name in os.listdir(new_folder):
    img_path = os.path.join(new_folder, img_name)
    img = cv2.imread(img_path)
    dimensions = img.ndim
    print(f"Number of dimensions of {img_name}: {dimensions}")
    #Resize
    resized_img = cv2.resize(img, (100, 100))
    # Display the resized image
    cv2_imshow(resized_img)
```

```
Number of dimensions of image_2.jpeg: 3
```

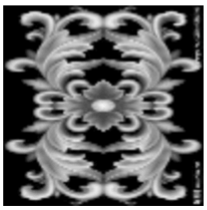
```
Number of dimensions of image_3.jpeg: 3
```



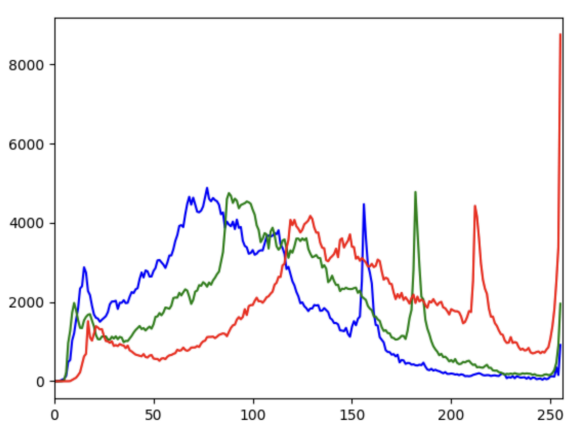
```
Number of dimensions of image_4.jpeg: 3
```

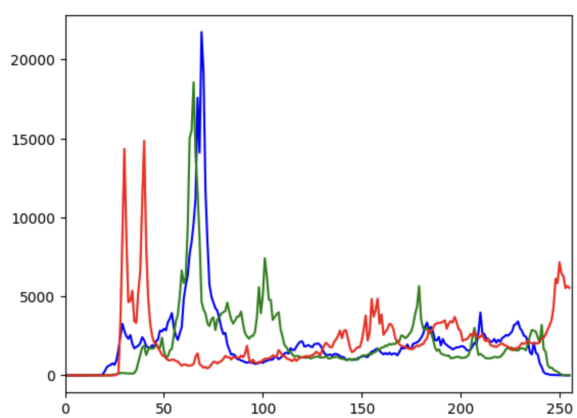
```
Number of dimensions of image_1.jpeg: 3
```

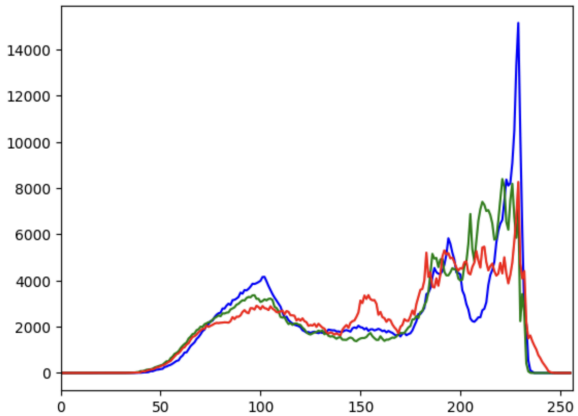
```
Number of dimensions of image_5.jpeg: 3
```

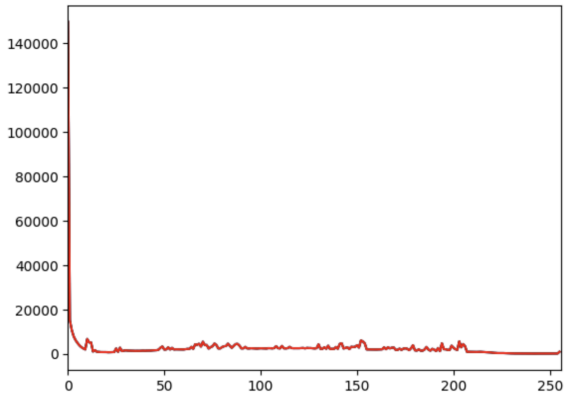


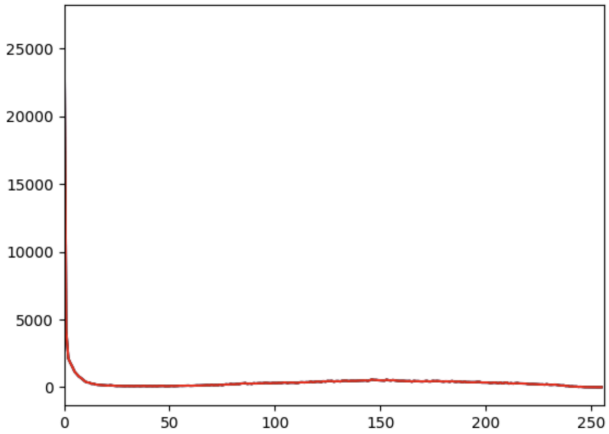
```
import matplotlib.pyplot as plt
for img_name in os.listdir(new_folder):
    img_path = os.path.join(new_folder, img_name)
    img = cv2.imread(img_path)
    for i, col in enumerate(['b', 'g', 'r']):
        hist = cv2.calcHist([img], [i], None, [256], [0, 256])
        plt.plot(hist, color = col)
        plt.xlim([0, 256])

    plt.show()
    #the coloured images have widespread distribution of r, g and b
    #black and white images have only r and that too in very low intensities
```






14000
12000
10000
8000
6000
4000
2000
0
0
50
100
150
200
250

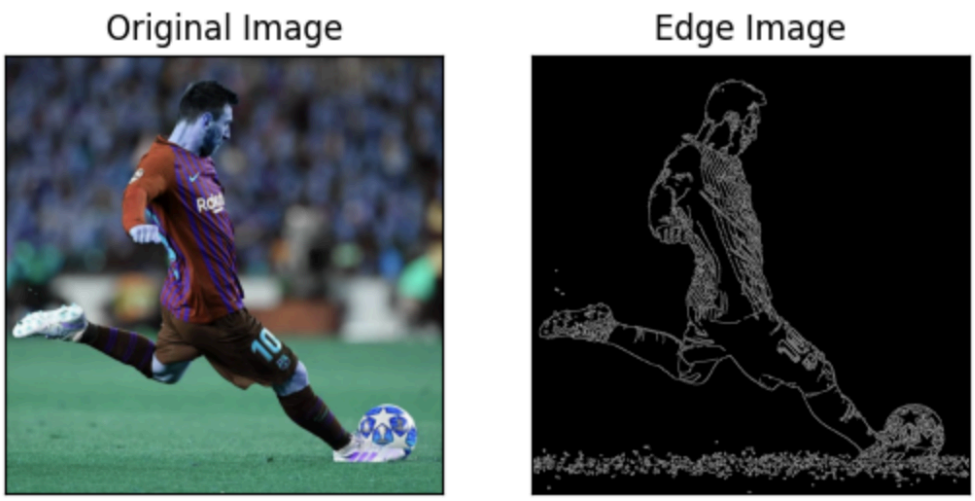

140000
120000
100000
80000
60000
40000
20000
0
0
50
100
150
200
250


25000
20000
15000
10000
5000
0
0
50
100
150
200
250

Q5: Edge Detection:

```
import cv2
from matplotlib import pyplot as plt
image_path="/content/messi.jpeg"
img = cv2.imread(image_path)
edges = cv2.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
```

```
(Text(0.5, 1.0, 'Edge Image'), ([], []), ([], []))
```



Q6: Corner Detection:

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

# Read the image
image_path = "/content/Wallpaper.jpeg"
img = cv2.imread(image_path)
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Perform Harris corner detection
dst = cv2.cornerHarris(gray_img, 2, 3, 0.04)

# Threshold for corner detection
threshold = 0.01 * dst.max()

# Mark detected corners on the image
img[dst > threshold] = [0, 0, 255]   # Red color for detected corners

# Display the result
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Harris Corner Detection')
plt.axis('off')
plt.show()
```

Q7: Keypoint Detection:

```python
import cv2
from matplotlib import pyplot as plt

# Read the image
image_path = "/content/messi.jpeg"
img = cv2.imread(image_path)
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Initialize SIFT detector
sift = cv2.SIFT_create()

# Detect keypoints
keypoints = sift.detect(gray_img, None)

# Draw keypoints on the image
img_with_keypoints = cv2.drawKeypoints(img, keypoints, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

# Display the result
plt.imshow(cv2.cvtColor(img_with_keypoints, cv2.COLOR_BGR2RGB))
plt.title('Keypoint Detection using SIFT')
plt.axis('off')
plt.show()
```

Q8: Image Classification:

[4]

```python
#loading the dataset
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
(X_train, y_train), (X_test,y_test) = datasets.cifar10.load_data()
X_train.shape
```

```
(50000, 32, 32, 3)
```

[17]

```python
#downsampling
import numpy as np
def downsample(X, y, num_samples_per_class):
    downsampled_X = []
    downsampled_y = []
    for class_label in np.unique(y):
        class_indices = np.where(y == class_label)[0]
        selected_indices = np.random.choice(class_indices, num_samples_per_class, replace=False)
        downsampled_X.append(X[selected_indices])
        downsampled_y.append(y[selected_indices])
    return np.concatenate(downsampled_X), np.concatenate(downsampled_y)
# Downsample the training set
X_train_downsampled, y_train_downsampled = downsample(X_train, y_train, 500)

# Downsample the test set
X_test_downsampled, y_test_downsampled = downsample(X_test, y_test, 100)

#reshape downsampled dataset
X_train_downsampled_flat = X_train_downsampled.reshape(X_train_downsampled.shape[0], -1)
X_test_downsampled_flat = X_test_downsampled.reshape(X_test_downsampled.shape[0], -1)
# Convert y_train_downsampled to 1D array
y_train_downsampled_flat = y_train_downsampled.ravel()
y_test_downsampled_flat = y_test_downsampled.ravel()
```

[18]

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
knn = KNeighborsClassifier(n_neighbors=1, p = 1)
knn.fit(X_train_downsampled_flat,y_train_downsampled_flat)
y_pred = knn.predict(X_test_downsampled_flat)
print(classification_report(y_test_downsampled_flat, y_pred))
average_accuracy = accuracy_score(y_test_downsampled_flat, y_pred)
print("Average Accuracy:", average_accuracy)
```

```
              precision    recall  f1-score   support

           0       0.30      0.44      0.36       100
           1       0.55      0.16      0.25       100
           2       0.23      0.34      0.27       100
           3       0.17      0.12      0.14       100
           4       0.19      0.37      0.25       100
           5       0.31      0.24      0.27       100
           6       0.21      0.21      0.21       100
           7       0.44      0.20      0.28       100
           8       0.34      0.53      0.41       100
           9       0.46      0.17      0.25       100

    accuracy                           0.28      1000
   macro avg       0.32      0.28      0.27      1000
weighted avg       0.32      0.28      0.27      1000

Average Accuracy: 0.278
```

Q9: Hyperparameter Tuning: Perform hyperparameter tuning using your downsampled training dataset. You should perform 5-fold cross validation as discussed in the link on training data to tune hyperparameters: (1) K-value, take possible values as 1,3,5; and (2) Distance metric as L1, L2. [5pt]

```python
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_neighbors': [1, 3, 5],  # K-value
    'p': [1, 2]  # Distance metric (L1 and L2)
}

# Initialize KNN classifier
knn = KNeighborsClassifier()

# Initialize GridSearchCV with 5-fold cross-validation
grid_search = GridSearchCV(estimator=knn, param_grid=param_grid, cv=5, scoring='accuracy')

# Perform grid search
grid_search.fit(X_train_downsampled_flat, y_train_downsampled_flat)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Get the best model
best_model = grid_search.best_estimator_

# Predict on test data using the best model
y_pred = best_model.predict(X_test_downsampled_flat)

# Calculate accuracy
accuracy = accuracy_score(y_test_downsampled_flat, y_pred)
print("Accuracy with Best Model:", accuracy)
```

```
Best Hyperparameters: {'n_neighbors': 1, 'p': 1}
Accuracy with Best Model: 0.278
```