Program 1: Image Splitting

In [ ]:

```python
import cv2
import numpy
import imageio as iio
from matplotlib import pyplot as plt
from matplotlib import image as mpimg


img = cv2.imread('your image here')
h, w, channels = img.shape
half = w//2

left_part = img[:, :half]


right_part = img[:, half:]


half2 = h//2

top = img[:half2, :]
bottom = img[half2:, :]
plt.title("Image")
plt.xlabel("X pixel scaling")
plt.ylabel("Y pixels scaling")

cv2.imwrite('top.jpg', top)
plt.imshow(top)
plt.show()
cv2.imwrite('bottom.jpg', bottom)
plt.imshow(bottom)
plt.show()
cv2.imwrite('right.jpg', right_part)
plt.imshow(right_part)
plt.show()
cv2.imwrite('left.jpg', left_part)
plt.imshow(left_part)
plt.show()
cv2.waitKey(0)

plt.title("Sheep Image")
plt.xlabel("X pixel scaling")
plt.ylabel("Y pixels scaling")

image = mpimg.imread("/content/IMG-20190808-WA0091-1.jpg")
plt.imshow(image)
plt.show()
```

Program 2: Image Transformation

In [ ]:

```python
import cv2
import numpy as np

FILE_NAME = 'volleyball.jpg'
try:
    # Read image from disk.
    img = cv2.imread(FILE_NAME)

    # Get number of pixel horizontally and vertically.
    (height, width) = img.shape[:2]

    # Specify the size of image along with interpolation methods.
    # cv2.INTER_AREA is used for shrinking, whereas cv2.INTER_CUBIC
    # is used for zooming.
    res = cv2.resize(img, (int(width / 2), int(height / 2)), interpolation = cv2.INTER_CUBIC)

    # Write image back to disk.
    cv2.imwrite('result.jpg', res)

except IOError:
    print ('Error while reading files !!!')


import cv2
import numpy as np

FILE_NAME = 'volleyball.jpg'
try:
    # Read image from the disk.
    img = cv2.imread(FILE_NAME)

    # Shape of image in terms of pixels.
    (rows, cols) = img.shape[:2]

    # getRotationMatrix2D creates a matrix needed for transformation.
    # We want matrix for rotation w.r.t center to 45 degree without scaling.
    M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 1)
    res = cv2.warpAffine(img, M, (cols, rows))

    # Write image back to disk.
    cv2.imwrite('result.jpg', res)
except IOError:
    print ('Error while reading files !!!')



import cv2
import numpy as np

FILE_NAME = 'volleyball.jpg'
# Create translation matrix.
# If the shift is (x, y) then matrix would be
# M = [1 0 x]
# [0 1 y]
# Let's shift by (100, 50).
M = np.float32([[1, 0, 100], [0, 1, 50]])

try:

    # Read image from disk.
    img = cv2.imread(FILE_NAME)
    (rows, cols) = img.shape[:2]

    # warpAffine does appropriate shifting given the
    # translation matrix.
    res = cv2.warpAffine(img, M, (cols, rows))

    # Write image back to disk.
    cv2.imwrite('result.jpg', res)

except IOError:
    print ('Error while reading files !!!')
```

Program 3: Erosion and Dilation

In [ ]:

```
# Python program to demonstrate erosion
import cv2
import numpy as np
# Reading the input image
img = cv2.imread('your image here', 0)
# Taking a matrix of size 5 as the kernel
kernel = np.ones((5, 5), np.uint8)
img_erosion = cv2.erode(img, kernel, iterations=1)
cv2.imshow('Input', img)
cv2.imshow('Erosion', img_erosion)

subtracted = cv2.subtract(img, img_erosion)

# TO show the output

img_dilation = cv2.dilate(img, kernel, iterations=1)

cv2.imshow('Input', img)
cv2.imshow('Erosion', img_erosion)
cv2.imshow('Sub', subtracted)
cv2.imshow('Dilation', img_dilation)

cv2.waitKey(0)
# To close the window
cv2.waitKey(0)
cv2.destroyAllWindows()
cv2.waitKey(0)
```

Program 4A : Edge Detection

In [10]:

```
import cv2

# Read the original image

img = cv2.imread(image)
# Display original image
cv2.imshow('Original', img)
cv2.waitKey(0)

# Convert to graycsale
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
# Blur the image for better edge detection
img_blur = cv2.GaussianBlur(img_gray, (3,3), 0)

# Sobel Edge Detection
sobelx = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=5) # Sobel Edge Detection on the X axis
sobely = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=5) # Sobel Edge Detection on the Y axis
sobelxy = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=1, ksize=5) # Combined X and Y Sobel Edge Detecti
# Display Sobel Edge Detection Images
cv2.imshow('Sobel X', sobelx)
cv2.waitKey(0)
cv2.imshow('Sobel Y', sobely)
cv2.waitKey(0)
cv2.imshow('Sobel X Y using Sobel() function', sobelxy)
cv2.waitKey(0)

# Canny Edge Detection
edges = cv2.Canny(image=img_blur, threshold1=100, threshold2=200) # Canny Edge Detection
# Display Canny Edge Detection Image
cv2.imshow('Canny Edge Detection', edges)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

Program 4B: Texture Extraction

In [14]:

```
import numpy as np
from skimage.io import imread
from skimage import io
from skimage.color import rgb2gray
import imageio
import skimage
import cv2

image = cv2.imread('your image here')
io.imshow(image)
image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
io.imshow(image)
co_matrix = skimage.feature.graycomatrix(image, [5], [0], levels=256, symmetric=True, normed=True)
contrast = skimage.feature.graycoprops(co_matrix, 'contrast')
correlation = skimage.feature.graycoprops(co_matrix, 'correlation')
energy = skimage.feature.graycoprops(co_matrix, 'energy')
homogeneity = skimage.feature.graycoprops(co_matrix, 'homogeneity')
print("Contrast:", contrast)
print("Correlation:", correlation)
print("Energy:", energy)
print("Homogeneity:", homogeneity)
```

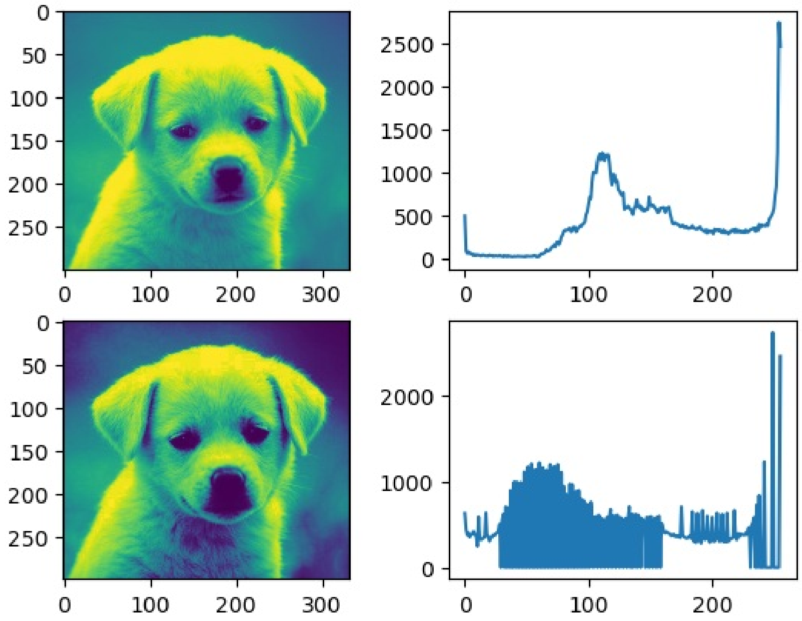
```
Contrast: [[195.07962549]]
Correlation: [[0.95501834]]
Energy: [[0.01990392]]
Homogeneity: [[0.27464869]]
```

Program 5A: Enhancing

In [2]:

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread( 'your image here',0)
hist1 = cv2.calcHist([img],[0],None,[256],[0,256])
img_2 = cv2.equalizeHist(img)
```

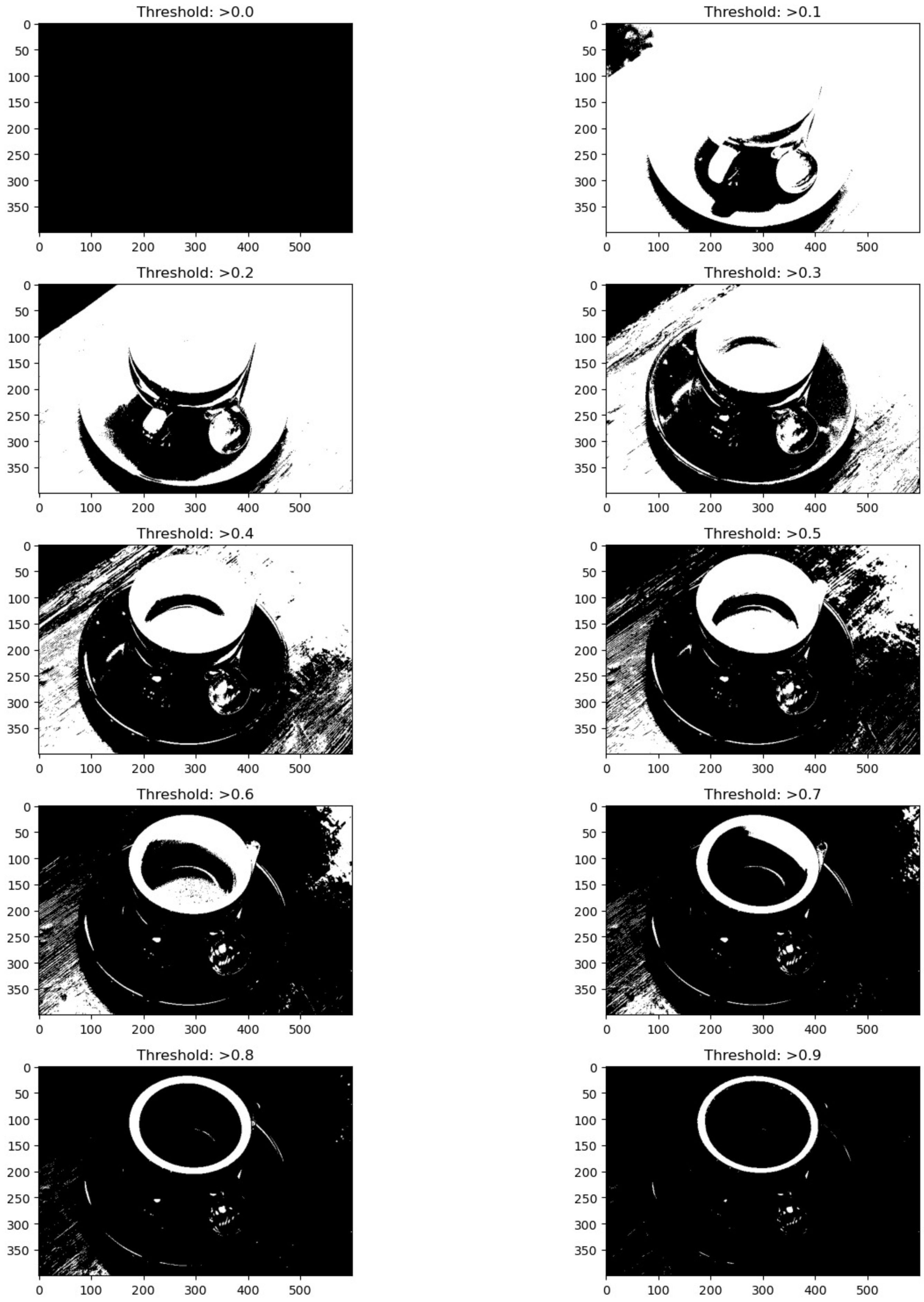
```
hist2 = cv2.calcHist([img_2],[0],None,[256],[0,256])
plt.subplot(221),plt.imshow(img);
plt.subplot(222),plt.plot(hist1);
plt.subplot(223),plt.imshow(img_2);
plt.subplot(224),plt.plot(hist2);
```



Program 5B : Segmentation

In [3]:

```
# Importing Necessary Libraries
# Displaying the sample image - Monochrome Format
from skimage import data
from skimage import filters
from skimage.color import rgb2gray
import matplotlib.pyplot as plt

# Sample Image of scikit-image package, need not include your own image!!!
coffee = data.coffee()
gray_coffee = rgb2gray(coffee)

# Setting the plot size to 15,15
plt.figure(figsize=(15, 15))

for i in range(10):

  # Iterating different thresholds
  binarized_gray = (gray_coffee > i*0.1)*1
  plt.subplot(5,2,i+1)

  # Rounding of the threshold
  # value to 1 decimal point
  plt.title("Threshold: >"+str(round(i*0.1,1)))

  # Displaying the binarized image
  # of various thresholds
  plt.imshow(binarized_gray, cmap = 'gray')

plt.tight_layout()
```



In [ ]:

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js