```
import numpy as np
import cv2 as cv

from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
img = cv.imread('/content/drive/MyDrive/Colab
Notebooks/InputUngradedAssignment.jpg', -1)

print(type(img))
```

```
<class 'numpy.ndarray'>
```

We are reading the image from the given address and storing it in 'img' object. One of the dependencies of OpenCV is numpy. There are various modes in which the original image can be read. Here we have used the option '-1', which caused the image to be read in colour. Similarly we can use '0' to upload the image in greyscale. Images are stored as numpy arrays. This can be verified by printing the data type for the img object as we can see below

```
print(img.shape)
```

```
(2136, 4624, 3)
```

Each image can be thought of as a three dimensional matrix. The three dimensions being length, breadth and colour. On checking the dimensions of the numpy array representation of the image we see that its length, breadth and colour dimensions are: 2136, 4624 and 3. Here 3 refers to the values of the three primary colours: Red, Blue and Green (**RGB**).

```
print(img)
```

```
[[[103 110 119]
  [104 111 120]
  [104 111 120]
  ...
  [119 125 130]
  [119 125 130]
  [119 125 130]]

 [[103 110 119]
  [103 110 119]
  [103 110 119]
  ...
  [118 124 129]
  [118 124 129]
  [117 123 128]]

 [[103 110 119]
```

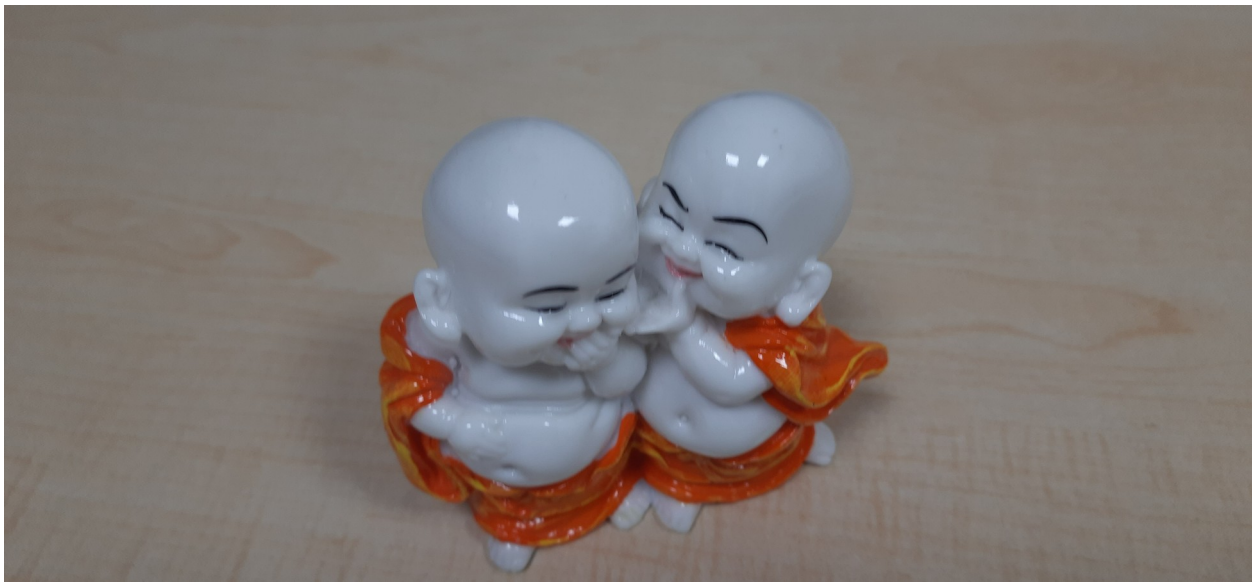
```
  [103 110 119]
  [102 109 118]
  ...
  [117 123 128]
  [116 122 127]
  [116 122 127]]

 ...

 [[100 109 122]
  [ 99 108 121]
  [ 99 108 121]
  ...
  [ 99 107 114]
  [105 113 120]
  [106 114 121]]

 [[101 110 123]
  [100 109 122]
  [100 109 122]
  ...
  [ 98 106 113]
  [108 116 123]
  [110 118 125]]

 [[100 109 122]
  [100 109 122]
  [100 109 122]
  ...
  [ 97 105 112]
  [111 119 126]
  [113 121 128]]]
```

If we print the img object as a numpy array we see that the numpy array is of the form:

```
[
  [ [234, 232, 21], [32, 34, 123], ... [0, 43, 98] ],
  [ [37, 74, 83], [90, 43, 98], ... [24, 39, 219] ],
  ...
  [[A, B, C], [X, Y, Z], ... [P, Q, R]]
  ...
]
```

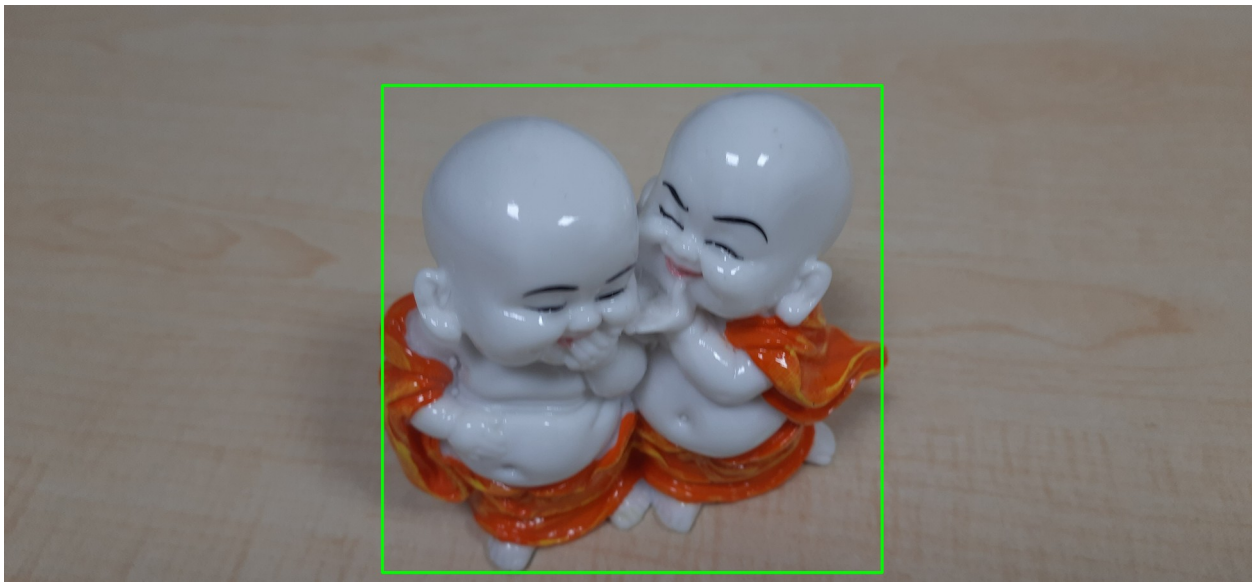
Where the innermost list represents the value for Red, Blue and Green colours for each pixel. And the consecutive outer lists represents the breadth and length of the image pixels respectively. However, for opencv, the pixels colour are not in [Red, Green, Blue] orientation, they are in stead in [Blue, Green, Red] orientation.

```
#cv.imshow('Picture', img)


from google.colab.patches import cv2_imshow

cv2_imshow(img)
```



Generally we are able to output the image in a separate window using the `cv.imshow('Picture_label', img)` command, however as it causes Jupyter notebooks to crash, so the alternative `cv2_imshow(img)` has been used from **google.colab.patches**. This command has similar functionality as `cv.imshow()`.

```
img = cv.rectangle(img, (1400, 300), (3250, 2100), (0, 255, 0), 10)

cv2_imshow(img)
```

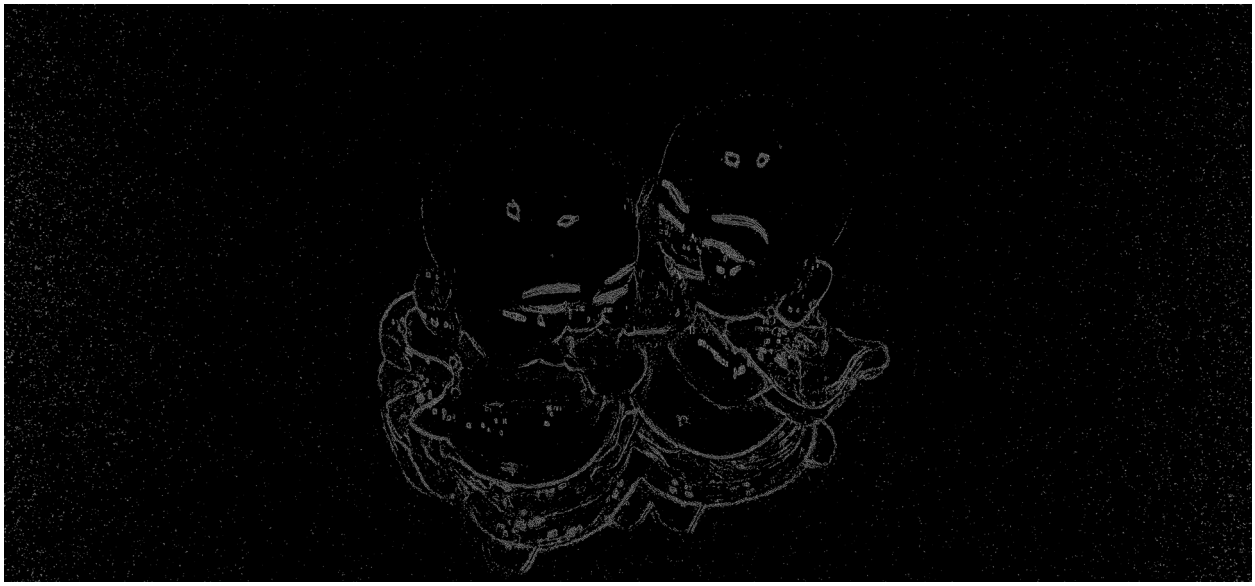


The bounding box for the foreground object in the image has been drawn after some trial and error tries. the `cv.rectangle()` method has been used to draw said bounding box. The method takes two tuple parameters, each of (*X_coordinate*, *Y_coordinate*) form. The first tuple is the left-hand upper corner of the bounding box and the second coordinates are that of the lower right-hand side of the bounding box. Other than that we can customize what the colour of the bounding box will be using a (Blue, Green, Red) pixel value tuple. We can also determine what the thickness of the line of the box will be. Here the thickness is taken as 10.

```
img_bw = cv.imread('/content/drive/MyDrive/Colab
Notebooks/InputUngradedAssignment.jpg', 0)

cv2_imshow(img_bw)
```

We take a separate instance of the image and read the image in greyscale thus getting a black and white version of the original version.

```
minVal = 25
maxVal = 30

edges = cv.Canny(img_bw, minVal, maxVal)
cv2_imshow(edges)
```



Canny Edge Detection algorithm contains a set of steps that are performed to transform the image into "line drawing" signifying possible edges in the image itself. minVal and maxVal are the two thresholding values that will determine which intensities of the transformed image we treat as edges and which ones we ignore.

```
img2 = cv.imread('/content/drive/MyDrive/Colab
Notebooks/InputUngradedAssignment.jpg', -1)
img2flat = np.float32(img2.reshape((-1, 3)))

criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 10, 1.0)
attempts = 10
K = 8

_,label,center=cv.kmeans(img2flat, K, None, criteria, attempts,
cv.KMEANS_PP_CENTERS)
```

Following is a brief description of input and output parameters for the `cv.kmeans()` function as it appears in the OpenCV Documentation

##Input parameters

samples : It should be of np.float32 data type, and each feature should be put in a single column.

nclusters(K) : Number of clusters required at end

criteria : It is the iteration termination criteria. When this criteria is satisfied, algorithm iteration stops. Actually, it should be a tuple of 3 parameters. They are `( type, max_iter, epsilon )`: type of termination criteria. It has 3 flags as below:

- cv.TERM_CRITERIA_EPS - stop the algorithm iteration if specified accuracy, epsilon, is reached.
- cv.TERM_CRITERIA_MAX_ITER - stop the algorithm after the specified number of iterations, max_iter.
- cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER - stop the iteration when any of the above condition is met.
- *max_iter* - An integer specifying maximum number of iterations.
- *epsilon* - Required accuracy

attempts : Flag to specify the number of times the algorithm is executed using different initial labellings. The algorithm returns the labels that yield the best compactness. This compactness is returned as output.

flags : This flag is used to specify how initial centers are taken. Normally two flags are used for this : cv.KMEANS_PP_CENTERS and cv.KMEANS_RANDOM_CENTERS.

##Output parameters

compactness : It is the sum of squared distance from each point to their corresponding centers.

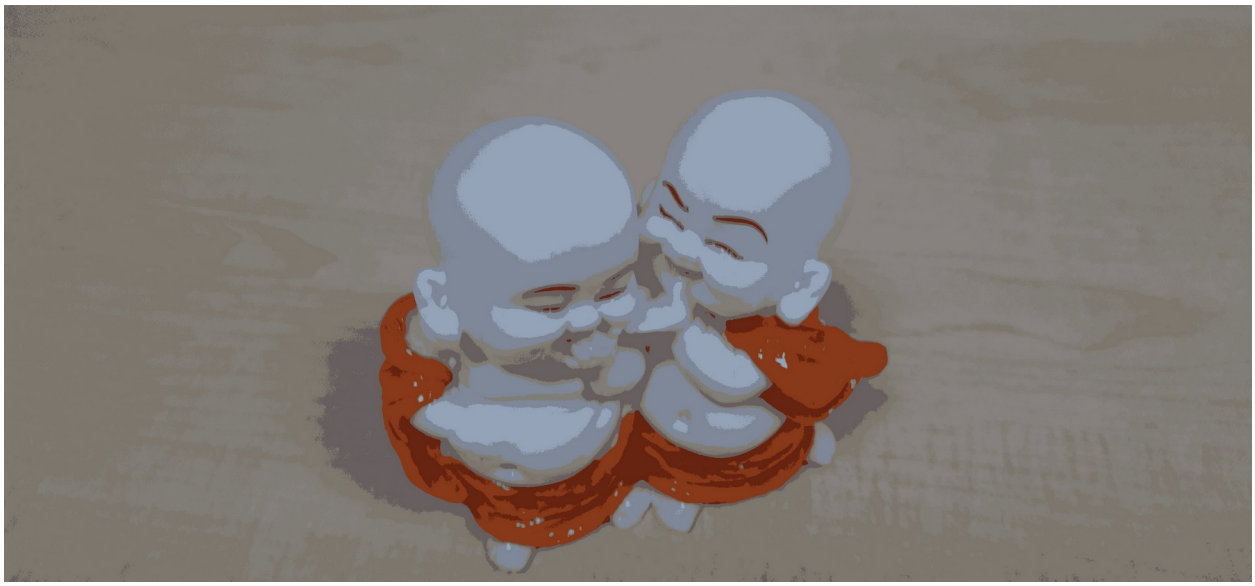
labels : This is the label array (same as 'code' in previous article) where each element marked '0', '1'.....

centers : This is array of centers of clusters.

```
center = np.uint8(center)
res = center[label.flatten()]
res2 = res.reshape((img2.shape))
cv.imwrite('quantized.jpg', res2)

True
```

We are changing the center back into its original datatype, that being 8-bit unsigned integer (otherwise range of values will be outside 0-255)

```
result = cv.imread('quantized.jpg', -1)

cv2_imshow(result)
```



###Sources:

https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

https://docs.opencv.org/3.4/d1/d5c/tutorial_py_kmeans_opencv.html