Assignment 1

Special Topics in Computer Vision

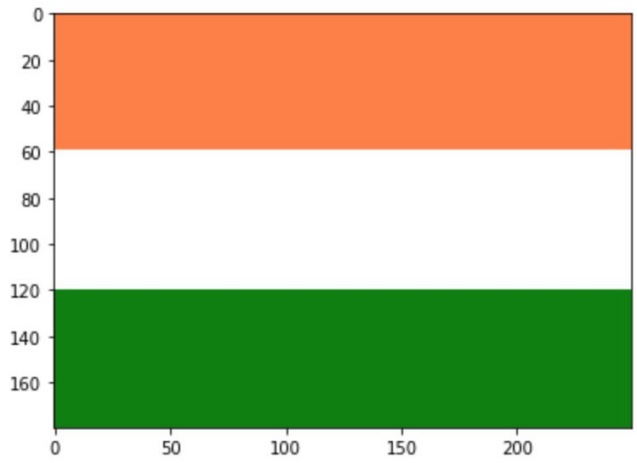
Gagandeep Singh: 2016037

Question 1:

To create a flag, I created a 2D matrix of shape 180x250x3 and initialized it with white colour.

```
In [9]: for i in range(length):
            for j in range(width):
                if i in range(0,60):
                    flagImage[i][j]=[255, 128, 64]
                elif i in range(60,120):
                    flagImage[i][j]=[255,255,255]
                else:
                    flagImage[i][j]=[0, 128, 0]
```

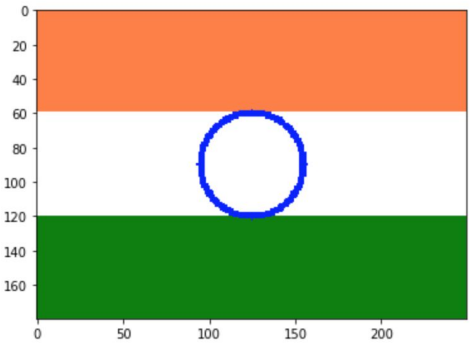
The above code helps to colour the flag into 3 section, i.e., saffron, white and green.



Next I took the center to be (90,125) with a radius of 30. Using the circles equation, i.e., (x-90)^2 + (x-125)^2=30^2 to create the center circle.

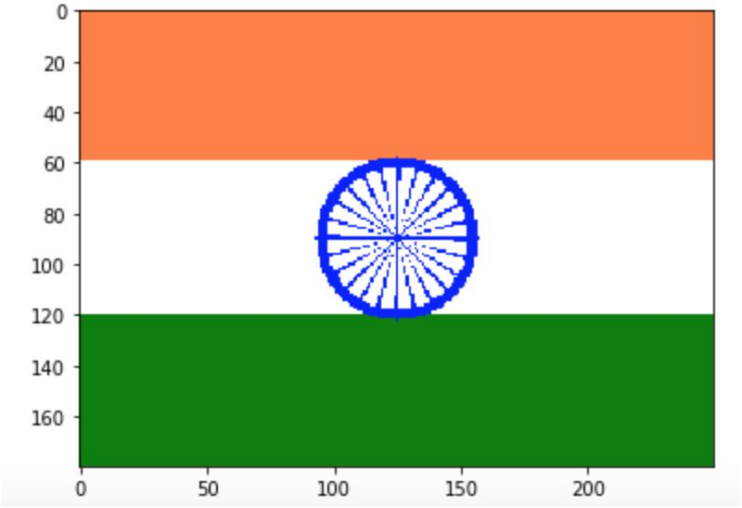
In [14]:
```
center_x=90
center_y=125
radius=30
```

In [8]:
```
for i in range(length):
    for j in range(width):
        val=pow(center_x-i, 2) + pow(center_y-j, 2)
        if(val<=pow(radius+2, 2) and val>=pow(radius-2, 2)):
            flagImage[i][j]=[0,0,255]
```



To create the spokes, for each point in the I checked its angle with the center and the distance with the center. If angle is a multiple of 15 and distance is less than 30, I colour that pixel blue. This gives the following output.

In [14]:
```
for i in range(60,120):
    for j in range(95,156):
        if j==125 or i==90:
            flagImage[i][j]=[0,0,255]
            continue
        rad=math.sqrt(pow(i-90,2)+pow(j-125,2))
        k=round(np.degrees(math.atan((90-i)/(125-j))))
        if rad<=30 and (k%15==0 or k%15==1 or k%15==2):
#             print(k)
            flagImage[i][j]=[0,0,255]
```


0
20
40
60
80
100
120
140
160
0
50
100
150
200

Question 2:

To create a HOG feature vector of an image I created a gradient vector for each pixel in the image which contains the magnitude and the direction of the gradient of each pixel.

In [11]:
```
gradient=[[0 for c in range(columns)] for r in range(rows)]
for r in range(rows):
    for c in range(columns):
        h_edge=0
        v_edge=0
        if c==0:
            h_edge=255-grayImage[r][c+1]
        elif c==columns-1:
            h_edge=255-grayImage[r][c-1]
        else:
            h_edge=abs(grayImage[r][c-1]-grayImage[r][c+1])

        if r==0:
            v_edge=255-grayImage[r+1][c]
        elif r==rows-1:
            v_edge=255-grayImage[r-1][c]
        else:
            v_edge=abs(grayImage[r-1][c]-grayImage[r+1][c])
        magnitude=sqrt(pow(h_edge,2)+pow(v_edge,2))
        angle=0
        if h_edge==0:
            angle=90
        else:
            angle=np.degrees(math.atan(v_edge/h_edge))
        gradient[r][c]=(magnitude,angle)
```

I divided the image into 16 x 16 patches to create the histogram for each patch.

In [14]:
```
histogram=[[0 for j in range(16)]for i in range(16)]
i=0
while i<256:
    j=0
    while j<256:
        arr=[0 for m in range(9)]
        for r in range(i,i+16):
            for c in range(j,j+16):
                mag=gradient[r][c][0]
                ang=gradient[r][c][1]
                print(mag,ang)
                ang%=180
                if ang%20==0:
                    ind=int(ang/20)
                    arr[ind]+=mag
                else:
                    ind=int(ang/20)
                    if ind==8:
                        val1=(180-ang)/20
                        val2=(ang-160)/20
                        arr[8]+=mag*val1
                        arr[0]+=mag*val2
                    else:
                        angle1=20*ind
                        angle2=angle1+20
                        val1=(angle2-ang)/20
                        val2=(ang-angle1)/20
                        arr[ind]+=mag*val1
                        arr[ind+1]+=mag*val2
        histogram[int(i/16)][int(j/16)]=arr
        j+=16
    i+=16
```

All the histograms were then normalized into a feature vector which was used to train the machine learning model.

In [16]:
```
# normalizeFeature=[[0 for j in range(16)] for i in range(16)]
normalizedFeature=[]
for r in range(15):
    for c in range(15):
        arr=[]
        for i in range(r,r+2):
            for j in range(c,c+2):
                for k in histogram[r][c]:
                    arr.append(k)
        s=0
        for i in arr:
            s+=pow(i,2)
        denom=sqrt(s)
        for i in arr:
            normalizedFeature.append(i/denom)
```

Then a KNN machine learning model was trained to predict if an image is a car or dog.
The data had around 20 images with 10 each of a dog and a car. The data was split into training and testing data. KNN model was trained on training data. When testing on test data it predicted all the images correctly.

Then the model was fed with additional 6 images, 5 of which were classified correctly.

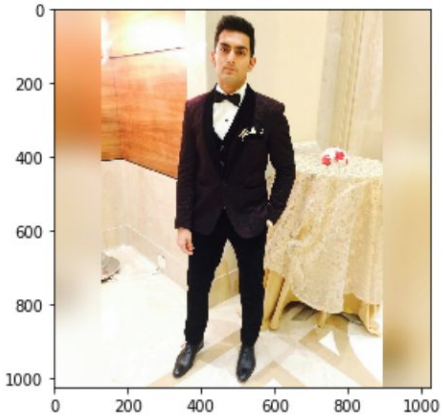
```
In [164]: from sklearn.metrics import classification_report,confusion_matrix

In [208]: from sklearn.neighbors import KNeighborsClassifier

In [225]: knn = KNeighborsClassifier(n_neighbors=1)

In [226]: knn.fit(list(X_train),y_train)

Out[226]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                               metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                               weights='uniform')
```

Question 3:

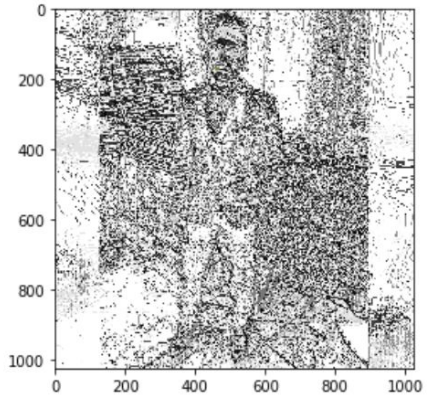
Initially an LBP feature was created for each pixel in the image.

```
In [38]: for i in range(1,rows-1):
             for j in range(1,columns-1):
                 count=0
                 if img[i][j]>=img[i-1][j-1]:
                     count+=pow(2,0)
                 if img[i][j]>=img[i][j-1]:
                     count+=pow(2,1)
                 if img[i][j]>=img[i+1][j-1]:
                     count+=pow(2,2)
                 if img[i][j]>=img[i+1][j]:
                     count+=pow(2,3)
                 if img[i][j]>=img[i+1][j+1]:
                     count+=pow(2,4)
                 if img[i][j]>=img[i][j+1]:
                     count+=pow(2,5)
                 if img[i][j]>=img[i-1][j+1]:
                     count+=pow(2,6)
                 if img[i][j]>=img[i-1][j]:
                     count+=pow(2,7)

                 print(count)
                 magnitude[i][j]=count
255
```

Input



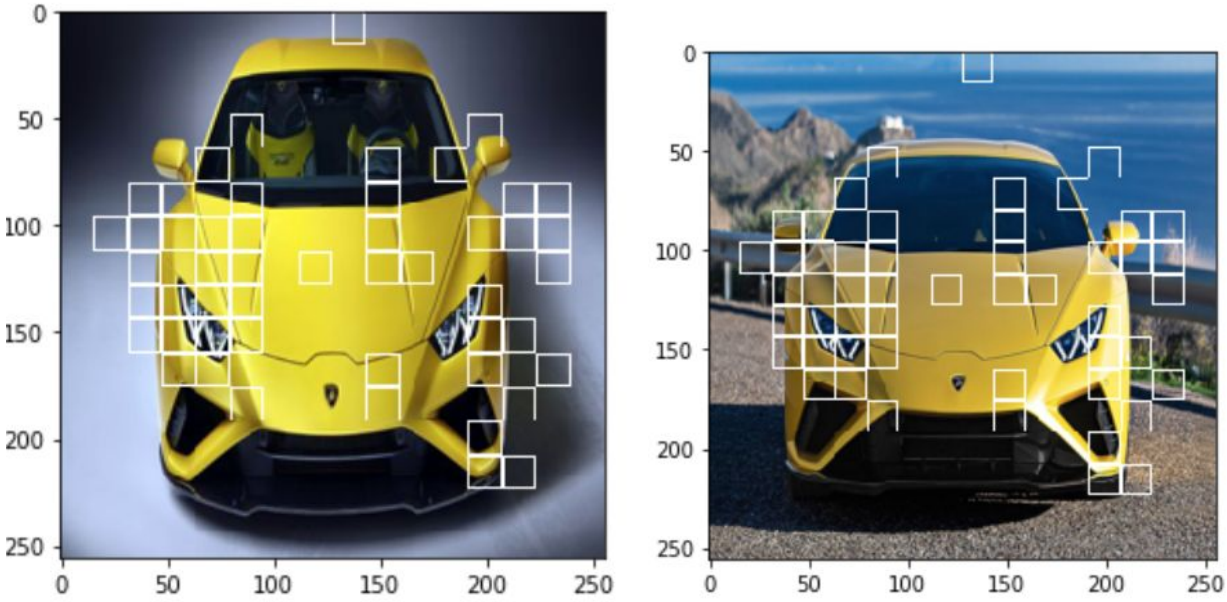
Output



Then the image was then divided into patches of 16 x 16 pixels and for each patch histogram was calculated.

```
In [48]: histograms=[]
         r=0
         while r<1024:
             c=0
             while c<1024:
                 patch=[0 for x in range(256)]
                 for i in range(r,r+64):
                     for j in range(c,c+64):
                         patch[magnitude[i][j]]+=1
                 histograms.append(patch)
                 c+=64
             print(len(histograms))
             r+=64
```

Two similar images were taken and for each of them histograms were calculated. For each patch in one image a distance was calculated with each patch in another image. For each pair of patch the closest patch was taken. Following was the result.

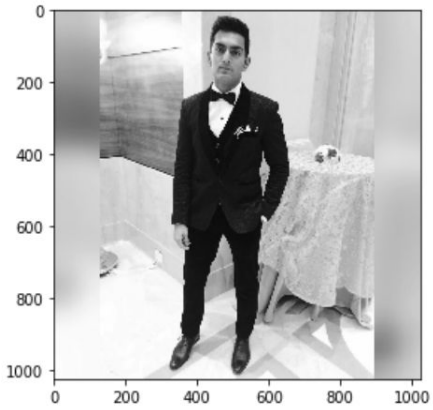


Question 4: Otsu' s binarization

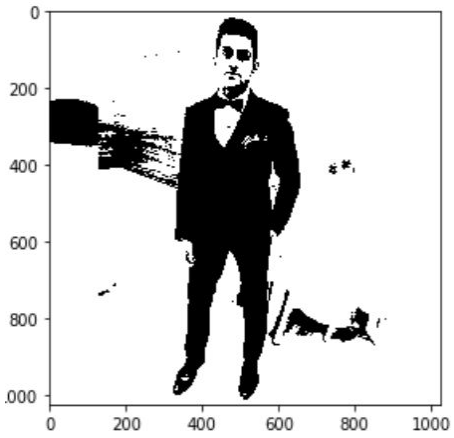
In [123]:
```
ind=-1
variance=float('inf')
currSum=0
for i in range(len(countPixels)):
    currSum+=countPixels[i]
    W1=currSum/numOfPixels
    W2=(numOfPixels-currSum)/numOfPixels
    V1=0
    V2=0
    mean1=0
    mean2=0
    s1=0
    s2=0
    for j in range(i+1):
        mean1+=countPixels[j]*j
        s1+=countPixels[j]
    for j in range(i+1,len(countPixels)):
        mean2+=countPixels[j]*j
        s2+=countPixels[j]
    if s1!=0:
        mean1=mean1/s1
    if s2!=0:
        mean2=mean2/s2

    for j in range(i+1):
        V1+=pow(j-mean1,2)*countPixels[j]
    for j in range(i+1,len(countPixels)):
        V2+=pow(j-mean2,2)*countPixels[j]
    if s1!=0:
        V1=V1/s1
    if s2!=0:
        V2=V2/s2

    var=(W1*V1)+(W2*V2)
    if var<variance:
        variance=var
        ind=i
```

The input image was converted into a grayscale image. An histogram was created showing the count of each number of pixels with value 0-255. The above code helps divide the histogram into two sections which have a minimum variance.

Input



Output

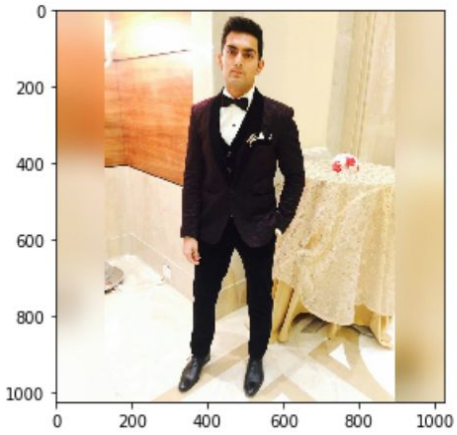


Question 5: K-means clustering

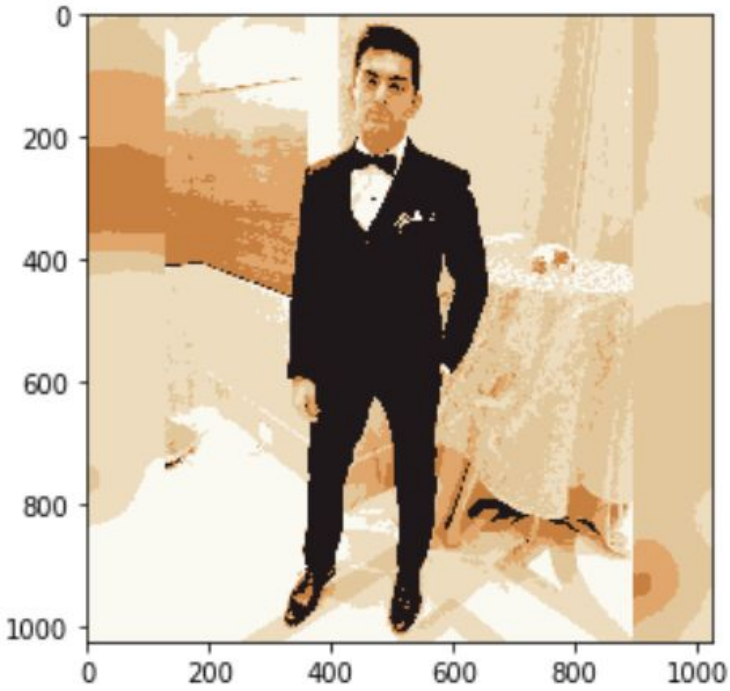
k=6

In [42]:
```
for i in range(20):
    print("iteration ",i)
    count_arr=[0 for i in range(6)]
    red_sum=[0 for i in range(6)]
    green_sum=[0 for i in range(6)]
    blue_sum=[0 for i in range(6)]
    for r in range(rows):
        for c in range(columns):
            red=image[r][c][0]
            green=image[r][c][1]
            blue=image[r][c][2]
            ind=-1
            minDist=float('inf')
            for j in range(6):
                dist=sqrt(pow(red_centroids[j]-red,2)+pow(green_centroids[j]-green,2)+pow(blue_centroids[j]-blue,2))
                if dist<minDist:
                    minDist=dist
                    ind=j
            result[r][c]=ind
            count_arr[ind]+=1
            red_sum[ind]+=red
            green_sum[ind]+=green
            blue_sum[ind]+=blue
    new_red=[int(red_sum[i]/count_arr[i]) for i in range(6)]
    new_green=[int(green_sum[i]/count_arr[i]) for i in range(6)]
    new_blue=[int(blue_sum[i]/count_arr[i]) for i in range(6)]
    if check(red_centroids,new_red) and check(green_centroids,new_green) and check(blue_centroids,new_blue):
        break
    else:
        red_centroids=[j for j in new_red]
        green_centroids=[j for j in new_green]
        blue_centroids=[j for j in new_blue]
```

Input



Output

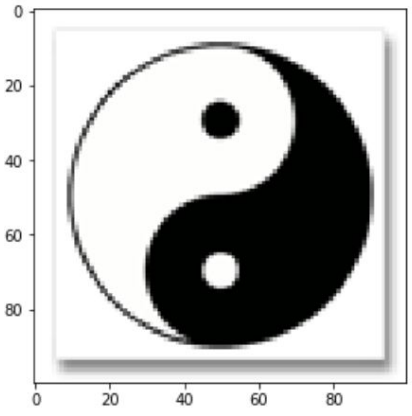


Question 6:

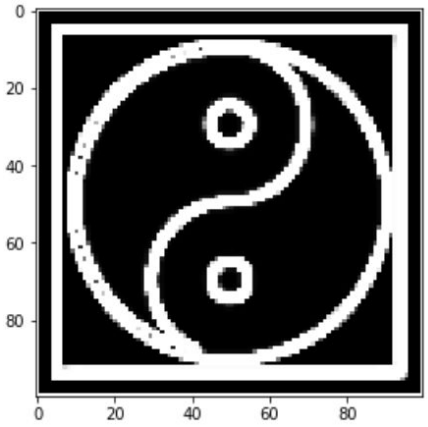
For each pixel in the image, a sobel filter was created to find the edges in the image.

```
In [191]: for x in range(1, width-1):  # ignore the edge pixels for simplici
              for y in range(1, height-1): # ignore edge pixels for simplici

                  # initialise Gx to 0 and Gy to 0 for every pixel
                  Gx = 0
                  Gy = 0

                  # top left pixel
                  p = img.getpixel((x-1, y-1))
                  r = p[0]
                  g = p[1]
                  b = p[2]

                  # intensity ranges from 0 to 765 (255 * 3)
                  intensity = r + g + b

                  # accumulate the value into Gx, and Gy
                  Gx += -intensity
                  Gy += -intensity

                  # remaining left column
                  p = img.getpixel((x-1, y))
                  r = p[0]
                  g = p[1]
                  b = p[2]

                  Gx += -2 * (r + g + b)

                  p = img.getpixel((x-1, y+1))
                  r = p[0]
                  g = p[1]
                  b = p[2]

                  Gx += -(r + g + b)
                  Gy += (r + g + b)

                  # middle pixels
                  p = img.getpixel((x, y-1))
                  r = p[0]
```

Input



Output



Using hough transform, circles were detected in the image.

```
In [118]: for r in range(rows):
              for c in range(20,80):
                  maxRadX=int(min(r,rows-r-1))
                  maxRadY=int(min(c,columns-c-1))
                  radius=min(maxRadX,maxRadY)
                  print(r,c,radius)
                  for rad in range(radius):
                      count=0
                      for i in range(r-rad,r+rad+1):
                          for j in range(c-rad,c+rad+1):
                              if int(sqrt(pow(r-i,2)+pow(c-j,2)))==rad and graySobel[i][j]==255:
                                  count+=1
                      if count>=50:
                          circles.append((r,c,rad,count))
```

The following circles were detected by the above code. There were more circles detected but they did not cross the threshold.